INSTITUTO TECNOLÓGICO DE CIUDAD JUÁREZ

APUNTES

NOMBRE DE LA ASIGNATURA

PROGRAMACIÓN BÁSICA

CARRERA

INGENIERÍA MECATRÓNICA

CLAVE DE LA ASIGNATURA

MTD-1024

PERIODO DE ELABORACIÓN

MARZO / 2022 - AGOSTO / 2022

ELABORÓ

LUDIVINA TORAL AGUILAR

CONTRIBUCIÓN ACADÉMICA

El desarrollo de algoritmos es un tema fundamental en el diseño de programas por lo cual el alumno debe tener buenas bases que le sirvan para poder desarrollar de manera fácil y rápida sus programas.

Estos apuntes servirán de apoyo al catedrático del Instituto Tecnológico de ciudad Juárez, en su labor cotidiana de enseñanza y al estudiante de mecatrónica le facilitará desarrollar su capacidad analítica y creadora, para de esta manera mejorar su destreza en la elaboración de aplicaciones de la vida ordinaria y de aplicaciones de la ingeniería.

CONTENIDO

CONTRIBUCIÓN ACADÉMICA ii
CONTENIDO iii
LISTA DE TABLAS vii
LISTA DE FIGURAS viii
INTRODUCCIÓN ix
1. Conceptos Básicos y Algoritmos 1
1.1 Clasificación del Software de: Sistemas y Aplicación 1
1.1.1 Organización Física de una Computadora 2
1.2 Representación de Algoritmos 4
1.2.1 Definición de Lenguaje 4
1.2.2 Lenguajes de Programación 4
1.2.3 Lenguaje Maquina 4
1.2.4 Lenguaje de Bajo Nivel 4
1.2.5 Lenguaje de Alto Nivel 4
1.2.6 Definición de Algoritmo 5
1.2.7 Tipos de Algoritmos 5
1.2.8 Lenguajes Algorítmicos 5
1.2.9 Tipos de Lenguajes Algorítmicos 5
1.2.10 Metodología para la Solución de Problemas por Medio de computadora 5
1.2.11 Tipos de Datos 7
1.2.11.1 Datos Simples 8
1.2.11.2 Estructurados (Def. por el usuario) 9
1.2.12 Identificadores 9
1.2.13 Palabras Reservadas, Constantes y Variables 10
1.2.14 Expresiones, Operadores y Operandos 12
Ejercicios de Evaluación de Expresiones 16
1.3 Diseño de Algoritmos 19
1.3.1 Simbología de Diagrama de Flujo 19
1.3.2 Estructuras Algorítmicas 20
1.3.2.1 Estructuras secuenciales 20
1.3.2.2 Estructuras Selectivas 20
1.3.2.3 Estructuras Repetitivas 22
1.3.3 Ejemplo de Diseño de Diagrama de Flujo 24
Ejercicios de Diseño de Diagrama de Flujo, Estructura Secuencial 25
Ejercicios de Diseño de Diagrama de Flujo, Estructuras Selectivas y Repetitivas 28
Ejercicios Propuestos Para Solucionar por el Alumno Capitulo | 36
2 INTRODUCCION A LA PROGRAMACION 37
2.1 Conceptos Básicos 37
2.1.1 Reglas para Nombrar un Archivo Fuente 38
2.2 Características del Lenguaje de Programación 38
2.3 Estructuras Básica de un Programa 40

2.3.1 Entrada (cin)....41
2.3.2 Declaración de una Variables....42
2.4 Elementos del Lenguaje....43
2.4.1 Tipos de Datos Básicos en C++....43
2.4.2 Operadores Básicos en C++....43
2.4.3 Operadores Básicos en C++....43
2.4.4 Operadores de Asignación de C++....44
2.5 Traducción de un Programa: Compilación, Enlace, Ejecución y Errores....44
2.5.1 La Compilación y sus Fases....44
2.5.2 Errores de Sintaxis y Lógicos....45
Ejercicios Resueltos Capitulo 2....47
Ejercicios Propuestos Para Solucionar por el Alumno Capitulo 2....52
3 CONTROL DE FLUJO....54
3.1 Estructuras Secuenciales....54
3.2 Estructuras Selectivas: if, if-else, switch....55
3.2.1 La Sentencias if....55
3.2.2 La Sentencia: Condición Doble if-else....55
3.2.3 La Sentencia: Condiciones Multiples switch....56
3.3 Estructuras Iterativas: while, for, do while....57
3.3.1 Repetición: El Bucle while....57
3.3.2 Repetición: El Bucle for....58
3.3.3 Repetición: El Bucle do while....59
Ejercicios Resueltos Capitulo 3....61
Ejercicios if, if else, switch....64
Ejercicios for....66
Ejercicios while....67
Ejercicios do while....69
Ejercicios Propuestos Para Solucionar por el Alumno Capitulo 3....70
4 ARREGLOS, ESTRUCTURAS Y UNIONES....72
4.1 Arreglo....72
4.2 Unidimensionales: Conceptos Básicos, Operaciones y Aplicaciones....72
4.2.1 Declaración de un Array Unidimensional....73
4.2.2 Inicialización de Arrays Unidimensionales....73
4.2.3 Llenado de Arrays Unidimensionales....74
Ejercicios Resueltos Capitulo 4....76
Ejercicios Arrays Unidimensionales....76
4.3 Arrays Multidimensionales....84
4.3.1 Declaración de un Array Multidimensionales....84
4.3.2 Inicialización de Arrays Multidimensionales....85
4.3.3 Llenado de Arrays Multidimensionales....85
Ejercicios Arrays Multidimensionales....87
Ejercicios Propuestos Para Solucionar por el Alumno Capitulo 4....90

5 MODULOS 91
5.1 Declaración y Uso de Módulos 92
5.1.1 Prototipos de las Funciones 92
5.1.2 Diseño o Definición de Función 92
5.1.3 Llamada a la Función 93
5.2 Paso de Parámetros o Argumentos 94
5.2.1 Parámetros de una Función 94
5.2.2 Paso de Parámetros por Valor 94
5.2.3 Paso de Parámetros por Referencia 94
5.2.4 Diferencia entre el paso de Parámetros por Valor y por Referencia 95
5.3 Implementación de Funciones 96
Ejercicios resueltos Capitulo 5 99
Ejercicios Propuestos para Solucionar por el Alumno Capitulo 5 111
6 PUNTEROS 112
6.1 Concepto de Puntero (Apuntador) 112
6.2 Direcciones y Referencias 113
6.3 Declaración de Apuntadores 115
6.4 Inicialización (iniciación) de Punteros 116
6.5 Asignación de Apuntadores 117
6.6 Desreferenciación de Apuntadores 117
6.7 Apuntadores para Paso de Parámetros por Referencia 118
Ejercicios Resueltos Capitulo 6 119
Ejercicios Propuestos Para Solucinar por el Alumno Capitulo 6 122
7 ARCHIVOS 123
7.1 Conceptos Básicos 123
7.1.1 La Jerarquía de Datos 123
7.2 Definición de Ficheros 124
7.3 Utilidad de los Ficheros 124
7.4 El Formato 125
7.5 Tipos de Ficheros 125
7.6 Programación con Ficheros 126
7.6.1 Ficheros en C++ 126
7.6.2 Escritura de un Fichero Tipo Texto 127
Ejercicios Resueltos Capitulo 7 128
7.6.3 Lectura de un Fichero Tipo Texto 130
Ejercicios Propuestos para Solucionar por el Alumno Capitulo 7 133
Fuentes de Información Consultadas 134

ANEXO 1 Reporte del Alumno Ejercicios Propuestos 135
Solución Ejercicios Propuestos Capitulo1 136
Solución Ejercicios Propuestos Capitulo 2 139
Solución Ejercicios Propuestos Capitulo 3 143
Solución Ejercicios Propuestos Capitulo 4 152
Solución Ejercicios Propuestos Capitulo 5 156
Solución Ejercicios Propuestos Capitulo 6 159
Solución Ejercicios Propuestos Capitulo 7 161
ANEXO 2 Recursos de Evaluación 163
Evaluación Capitulo 1 164
Evaluación Capitulo 2 169
Evaluación Capitulo 3 173
Evaluación Capitulo 4 176
Evaluación Capitulo 5 178
Evaluación Capitulo 6 180
Evaluación Capitulo 7 181
ANEXO 3 Instrumentación Didáctica y Programa de Estudio de la asignatura: Programación Básica.....182
ANEXO 4 PROYECTO INTEGRADOR 210
PROYECTO INTEGRADOR DE ALGORITMOS (DIAGRAMAS DE FLUJO)
PROYECTO INTEGRADOR DE INTRODUCCION A LA PROGRAMACION
PROYECTO INTEGRADOR DE CONTROL DE FLUJO
PROYECTO INTEGRADOR DE ARREGLOS, ESTRUCTURAS Y UNIONES
PROYECTO INTEGRADOR DE MODULOS
PROYECTO INTEGRADOR DE PUNTEROS
PROYECTO INTEGRADOR DE FICHEROS

LISTA DE TABLAS

Tabla 1.1 Operadores Lógicos y Tabla de Verdad 15

Tabla 1.2 Precedencia de los Operadores Aritméticos, Relacionales y Lógicos 15

Tabla 3.1 Sintaxis en C++ de Estructuras Selectivas y Repetitivas 60

LISTA DE FIGURAS

Fig. 1.1 Proceso de Información en una Computadora 1

Fig. 1.2 Organización Física de una Computadora 2

Fig. 1.3 Tipos de Datos.....8

Fig. 1.4 Simbología de Diagramas de Flujo 19

Fig. 2.1 Proceso de Transformación de un Algoritmo en un Programa Ejecutable 37

Fig. 2.2 Nombre de un Archivo en Código Fuente 38

Fig. 2.3 Características de C y C++ 39

Fig. 2.4 Estructura Básica de un Programa 40

Fig. 2.5 Ejemplo de un Programa 42

Fig. 2.6 Tipos de Datos Básicos en C++.....43

Fig. 2.7 Operadores Básicos en C++ 43

Fig. 2.8 Operadores de Incremento y Decremento 43

Fig. 2.9 Operadores de Asignación de C++ 44

Fig. 2.10 Etapas en la Ejecución de un Programa 45

Fig. 4.1 Representación Grafica de un Array 73

Fig 4.2 Estructura de Dos Dimensiones.....84

INTRODUCCION

El contenido de estos apuntes abarca todos los temas y subtemas de la asignatura de Programación Básica. La asignatura se organiza en siete capítulos, Comprende los conceptos básicos de la programación y escribe expresiones aritméticas y lógicas en un lenguaje de programación. Así como el uso y funcionamiento de las estructuras secuenciales, selectivas, arreglos unidimensionales y multidimensionales, punteros y archivos en el desarrollo de aplicaciones. Es capaz de aplicarlos al construir y desarrollar aplicaciones de software que requieran dichas estructuras.

En el primer capítulo se estudian los conceptos básicos para introducir al estudiante en la programación con la finalidad de obtener las bases conceptuales para abordar las siguientes temáticas. También, se desarrolla el análisis y desarrollo de algoritmos, con la finalidad de dar solución a problemas reales utilizando el razonamiento lógico.

El segundo capítulo, tiene la finalidad de obtener y aplicar herramientas necesarias para diseñar e implementar soluciones en un lenguaje de programación, utilizando los conceptos adquiridos.

El tercer capítulo tiene como objetivo que el alumno identifique, comprenda, seleccione e implemente la estructura de control más adecuada a un problema específico, así como el diseño de bloques de códigos reutilizables, dado que es común encontrar en la práctica problemas cuyas operaciones están condicionadas o deban ejecutarse un número repetido de veces.

El cuarto capítulo tiene la finalidad de implementar arreglos para una gran variedad de propósitos que proporcionan un medio conveniente de agrupar variables relacionadas y organizar datos de una manera que puedan ser fácilmente procesados.

El quinto capítulo, módulos, se trata como un bloque que ayuda al re uso del software, que incluso es adaptable al establecer módulos que pueden ser llamados con diferentes parámetros.

El sexto capítulo, punteros, se define el concepto, declaración y uso de punteros, llamadas por valor y referencia.

El séptimo capítulo es Archivos que pueden ser secuenciales o directos, el manejo de operaciones con archivos y la aplicación de estos junto con el resto de las otras estructuras de programación.

El contenido de estos apuntes tiene el propósito de lograr un mayor interés de aprendizaje en los estudiantes. La enseñanza-aprendizaje no debe ser solamente informativa si no también formativa, para que el estudiante desarrolle sus propias capacidades de aprendizaje.

CARTA DE RECONOCIMIENTO DEL AUTOR DE LOS DERECHOS A FAVOR DEL TECNM

Ciudad Juárez, Chihuahua, 30-AGOSTO-2022

TECNOLÓGICO NACIONAL DE MÉXICO
PRESENTE

Bajo protesta de decir verdad, C. Ludivina Toral Aguilar, personal docente adscrito al Campus Cd. Juárez del Tecnológico Nacional de México, manifiesto que en cumplimiento de mis actividades relacionadas con el Año Sabático elaboré la obra titulada "Apuntes de Programación Básica".

Con base en lo anterior, y con fundamento en los artículos 83 de la Ley Federal del Derecho de Autor y 46 de su Reglamento, reconozco que el Tecnológico Nacional de México es titular de los derechos patrimoniales sobre la misma y le corresponden las facultades relativas a la divulgación, integridad de la obra y de colección, conservando el derecho a figurar como autor.

Asimismo, respondo por la autoría y originalidad de la citada obra, y relevo de toda responsabilidad al Tecnológico Nacional de México de cualquier demanda o declaración que llegara a formular alguna persona física o moral que considere que con esta obra es afectado en alguno de los derechos protegidos por la Ley en cita, asumiendo todas las consecuencias legales y económicas.

A T E N T A M E N T E

LUDIVINA TORAL AGUILAR
AUTOR

1. CONCEPTOS BÁSICOS Y ALGORITMOS

1.1 Clasificación del Software de: Sistemas y Aplicación

El software es la parte logica de computadora, que permite el manejo de los recursos y la realizacion de tareas especificas, tambien denominados programas.

Se clasifica en 2 partes:

1. SOFTWARE DE SISTEMAS
2. SOFWARE DE APLICACIÓN

1. SOFTWARE DE SISTEMAS: Son aquellos programas que permiten la administracion de la parte fisica o los recursos de la computadora, es la que interactua entre el usuario y los componentes hardware del ordenador. Se clasifican el Sistemas Operativos Monousuarios y Multiusuarios.
2. SOFTWARE DE APLICACION:Son aquellos programas que nos ayudan a tareas especificas como edicion de textos, imagenes, calculos, etc. tambien conocidos como aplicaciones.

Una computadora, es un dispositivo electrónico utilizado para procesar información y obtener resultados. Los datos y la información se pueden introducir en la computadora por la entrada (input) y a continuación se procesan para producir una salida (output, resultados), como se observa en la figura 1.1.

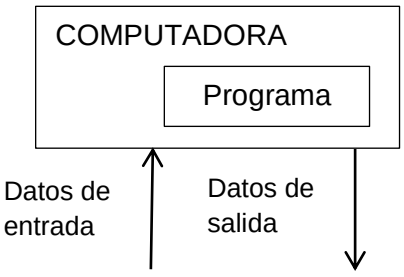


Fig. 1.1 Proceso de Información en una Computadora

Programa, es el conjunto de instrucciones escritas de algún lenguaje de programación y que ejecutadas secuencialmente resuelven un problema específico.

1.1.1 Organización Física de una Computadora (HARDWARE)

La mayoría de las computadoras, grandes o pequeñas, están organizadas como se muestra en la figura 1.2. Constan fundamentalmente de tres componentes principales: Unidad Central de Proceso (UCP) o procesador (compuesta de la UAL, Unidad Aritmética y Lógica, y la UC, Unidad de Control); la memoria principal o central y el programa.

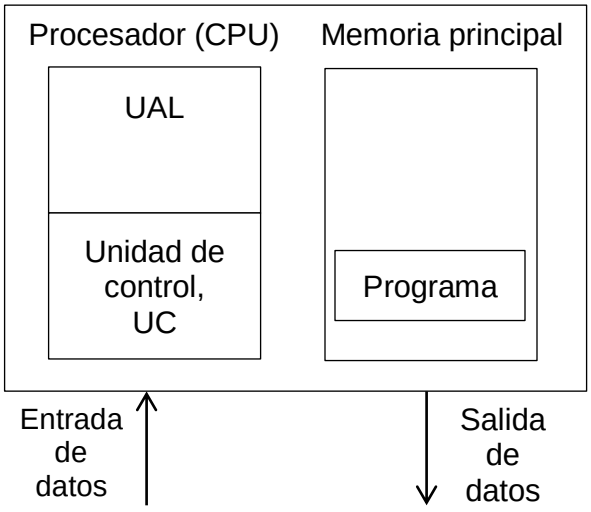


Fig 1.2 Organización Física de una Computadora

La *Unidad Central de Proceso*, **UCP** (*Central Processing Unit,* **CPU**, en inglés), dirige y controla el proceso de información realizado por la computadora. La **UCP** procesa o manipula la información almacenada en memoria; puede recuperar información desde memoria (esta información son datos o instrucciones: programas).

La **UCP** consta de dos componentes: *unidad de control* (**UC**) y *unidad aritmética-lógica* (**UAL)** (Figura 1.2). La **unidad de control** (*Control Unit*, **CU**) coordina las actividades de la computadora y determina qué operaciones se

deben realizar y en qué orden; asimismo controla y sincroniza todo el proceso de la computadora. La **unidad aritmético-lógica** (*Arithmetic-Logic Unit*, **ALU**) realiza operaciones aritméticas y lógicas, tales como suma, resta, multiplicación, división y comparaciones.

En la memoria principal se almacenan:
• Los datos enviados para procesarse desde los dispositivos de entrada.
• Los programas que realizarán los procesos.
• Los resultados obtenidos preparados para enviarse a un dispositivo de salida.

En la memoria principal se pueden distinguir dos tipos de memoria: **RAM** y **ROM**. La memoria **RAM** (**R**andom **A**ccess **M**emory, Memoria de acceso aleatorio) almacena los datos e instrucciones a procesar. Es un tipo de memoria volátil (su contenido se pierde cuando se apaga la computadora); esta memoria es, en realidad, la que se suele conocer como memoria principal o de trabajo; en esta memoria se pueden escribir datos y leer de ella. La memoria **ROM** (**R**ead **O**nly **M**emory, Memoria de sólo lectura) es una memoria permanente en la que no se puede escribir (viene pregrabada por el fabricante); es una ***memoria de sólo lectura***. Los programas almacenados en ROM no se pierden al apagar la computadora y cuando se enciende, se lee la información almacenada en esta memoria. Al ser esta memoria de sólo lectura, los programas almacenados en los chips ROM no se pueden modificar y suelen utilizarse para almacenar los programas básicos que sirven para arrancar la computadora.

Dispositivos de Entrada: Como su nombre lo indica, sirven para introducir datos (información) en la computadora para su proceso. Los datos se leen de los dispositivos de entrada y se almacenan en la memoria central o interna. Ejemplos: teclado, scanners (digitalizadores de rastreo), mouse (ratón), trackball (bola de ratón estacionario), joystick (palancas de juego), lápiz óptico.

Dispositivos de Salida: Regresan los datos procesados que sirven de información al usuario. Ejemplo: monitor, impresora.

1.2 Representación de Algoritmos: Gráfica y Pseudocódigo

1.2.1 Definicion de Lenguaje

Es una serie de símbolos que sirven para transmitir uno o más mensajes (ideas) entre dos entidades diferentes. A la transmisión de mensajes se le conoce comúnmente como **comunicación**.

1.2.2 Lenguajes de Programación

Es un conjunto de símbolos, caracteres y reglas (programas) que le permiten a las personas comunicarse con la computadora. Los lenguajes de programación tienen un conjunto de instrucciones que nos permiten realizar operaciones de entrada/salida, calculo, manipulación de textos, lógica/comparación y almacenamiento/recuperación. Los lenguajes de programación se clasifican en:

1.2.3 Lenguaje Maquina

Son aquellos cuyas instrucciones son directamente entendibles por la computadora y no necesitan traducción posterior para que la CPU pueda comprender y ejecutar el programa. Las instrucciones en lenguaje maquina se expresan en términos de la unidad de memoria mas pequeña el bit (dígito binario 0 o 1).

1.2.4 Lenguaje de Bajo Nivel (Ensamblador)

En este lenguaje de bajo nivel, las instrucciones se escriben en códigos alfabéticos conocidos como mnemotécnicos para las operaciones y direcciones simbólicas.

1.2.5 Lenguaje de Alto Nivel

Los lenguajes de programación de alto nivel (BASIC, pascal, cobol, frotran, etc.) son aquellos en los que las instrucciones o sentencias a la computadora son escritas con palabras similares a los lenguajes humanos (en general en ingles), lo que facilita la escritura y comprensión del programa.

1.2.6 Definición de Algoritmo

La palabra algoritmo se deriva de la traducción al latín de la palabra árabe alkhowarizmi, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX. Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.

1.2.7 Tipos de Algoritmos

Cualitativos: Son aquellos en los que se describen los pasos utilizando palabras.

Cuantitativos: Son aquellos en los que se utilizan cálculos numéricos para definir los pasos del proceso.

1.2.8 Lenguajes Algorítmicos

Es una serie de símbolos y reglas que se utilizan para describir de manera explícita un proceso.

1.2.9 Tipos de Lenguajes Algorítmicos

Gráficos: Es la representación gráfica de las operaciones que realiza un algoritmo (diagrama de flujo).

No Gráficos: Representa en forma descriptiva las operaciones que debe realizar un algoritmo (pseudocodigo).

1.2.10 Metodología para la Solución de Problemas por Medio de Computadora

1 Definición del Problema

Esta fase está dada por el enunciado del problema, el cual requiere una definición clara y precisa. Es importante que se conozca lo que se desea que realice la computadora; mientras esto no se conozca del todo no tiene mucho caso continuar con la siguiente etapa.

2 Análisis del Problema

Una vez que se ha comprendido lo que se desea de la computadora, es necesario definir:

Los datos de entrada.

Cuál es la información que se desea producir (salida)

Los métodos y fórmulas que se necesitan para procesar los datos.

Una recomendación muy práctica es el que nos pongamos en el lugar de la computadora y analicemos que es lo que necesitamos que nos ordenen y en que secuencia para producir los resultados esperados.

3 Diseño del Algoritmo

Algoritmo.- Es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.

Las características de un buen algoritmo son:

Debe tener un punto particular de inicio.

Debe ser definido, no debe permitir dobles interpretaciones.

Debe ser general, es decir, soportar la mayoría de las variantes que se puedan presentar en la definición del problema.

Debe ser finito en tamaño y tiempo de ejecución.

Diagrama de flujo.- Es una representación gráfica de un algoritmo que se caracteriza por usar símbolos gráficos, para expresar de forma sencilla y clara el orden lógico en el que se realizan las acciones de un algoritmo.

4 Codificación

La codificación es la operación de escribir la solución del problema (de acuerdo a la lógica del diagrama de flujo o pseudocodigo), en una serie de instrucciones detalladas, en un código reconocible por la computadora, la serie de instrucciones detalladas se le conoce como código fuente, el cual se escribe en un lenguaje de programación o lenguaje de alto nivel. (Aquí se hace el programa en el lenguaje de programación, visual c++ lenguaje a utilizar)

5 Prueba y Depuración

Los errores humanos dentro de la programación de computadoras son muchos y aumentan considerablemente con la complejidad del problema. El proceso de identificar y eliminar errores, para dar paso a una solución sin errores se le llama depuración.

La depuración o prueba resulta una tarea tan creativa como el mismo desarrollo de la solución, por ello se debe considerar con el mismo interés y entusiasmo.

Resulta conveniente observar los siguientes principios al realizar una depuración, ya que de este trabajo depende el éxito de nuestra solución.

6 Documentación

Es la guía o comunicación escrita es sus variadas formas, ya sea en enunciados, procedimientos, dibujos o diagramas.
A menudo un programa escrito por una persona, es usado por otra. Por ello la documentación sirve para ayudar a comprender o usar un programa o para facilitar futuras modificaciones (mantenimiento).

7 Mantenimiento

Se lleva acabo después de terminado el programa, cuando se detecta que es necesario hacer algún cambio, ajuste o complementación al programa para que siga trabajando de manera correcta. Para poder realizar este trabajo se requiere que el programa este correctamente documentado.

1.2.11 Tipos De Datos

Todos los datos tienen un tipo asociado con ellos. Un dato puede ser un simple carácter, tal como ‘a’, un valor entero tal como 35. El tipo de dato determina la naturaleza del conjunto de valores que puede tomar una variable. Los tipos de datos se muestran en la figura 1.3.

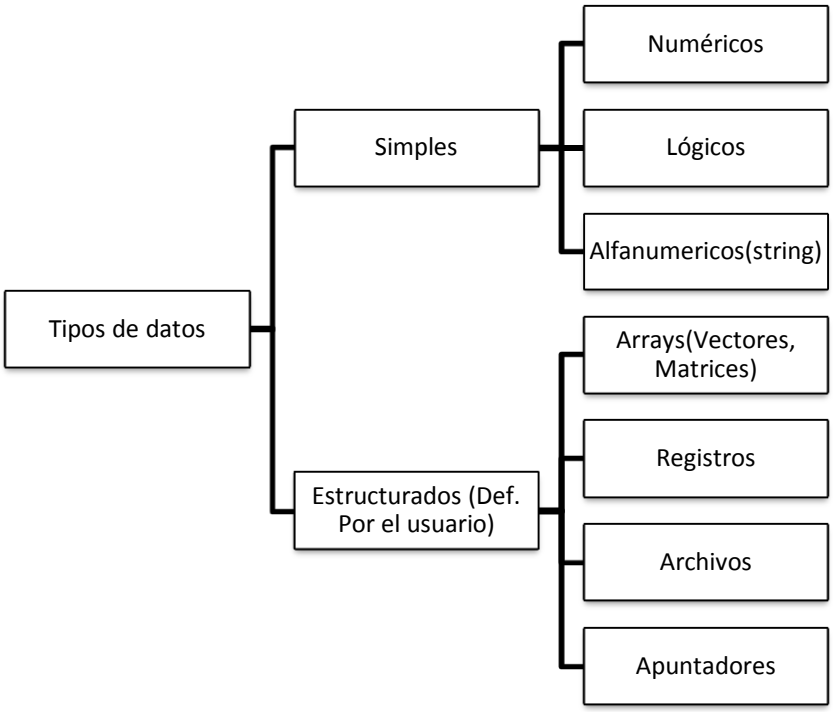


Fig. 1.3 Tipos de Datos

1.2.11.1 Tipos de Datos Simples

Datos Numéricos: Permiten representar valores escalares de forma numérica, esto incluye a los números enteros y los reales. Este tipo de datos permiten realizar operaciones aritméticas comunes.

Datos Lógicos: Son aquellos que solo pueden tener dos valores (cierto o falso) ya que representan el resultado de una comparación entre otros datos (numéricos o alfanuméricos).

Datos Alfanuméricos (String): Es una secuencia de caracteres alfanuméricos que permiten representar valores identificables de forma descriptiva, esto incluye nombres de personas, direcciones, etc. Es posible representar números como alfanuméricos, pero estos pierden su propiedad matemática, es decir no es posible hacer operaciones con ellos. Este tipo de datos se representan encerrados entre comillas.

1.2.11.2 Estructurados (Def. por el usuario)

Arrays: Son otra estructura de datos estática, pero con mayor beneficio que las variables simples.

Registro: Es una forma específica de estructura de datos que reúne una serie de campos de diversos tipos de datos que definen una estructura.

Archivos: Es una secuencia de datos almacenados en un medio persistente que están disponibles para ser utilizados por un programa.

Apuntadores: Es una variable que contiene la dirección de memoria de otra variable.

1.2.12 Identificadores

Un identificador es una secuencia de caracteres que sirve para identificar una posición en la memoria de la computadora, que nos permite accesar a su contenido. Los identificadores representan los datos de un programa (constantes, variables, tipos de datos).

Ejemplo:
Nombre
Num_hrs Calif2
nombre_clase Indice Día_Mes_Año
elemento_mayor Cantidad_Total Fecha_Compra_Casa Habitacion12

El identificador puede ser de cualquier longitud; sin embargo la mayoría de los compiladores pueden imponerle alguna restricción; en cualquier caso le recomendamos no sea muy largo pensando en la escritura y posibles modificaciones.

Algunos lenguajes son *sensible a las mayúsculas/minisculas*, por consiguiente, se reconoce como distintos los identificadores ALFA y alfa (le recomendamos que utilice siempre el mismo estilo al escribir sus identificadores). Un consejo que puede servir de posible regla puede ser:
1. Escribir identificadores de variables en letras minúsculas.
2. Constantes en mayúsculas.
3. Funciones con tipo de letra mixto: mayúscula/minúscula.

Reglas Básicas de Formación de Identificadores

1. Secuencia de letras o dígitos; el primer carácter puede ser una letra o un subrayado (compiladores de Borland, entre otros).
2. Los identificadores son sensibles a las mayúsculas y minúsculas:
minum es distinto de MiNum.
3. Los identificadores pueden tener cualquier longitud, pero sólo son significativos los 32 primeros.
4. Los identificadores no pueden ser palabras reservadas, tales como if, switch o else.

1.2.13 Palabras Reservadas, Constantes y Variables

Palabras Reservadas

Las palabras reservada (*keyword* o *reserved word*), son palabras clave, tienen un significado especial para el compilador de cualquier lenguaje de programación. Una palabra reservada no se puede utilizar como nombre de identificador, objeto o función.

Constantes y Variables

Constante: Una constante es un dato numérico o alfanumérico que no cambia durante la ejecución del programa.

Ejemplo: pi = 3.1416

Variable: Es un espacio en la memoria de la computadora que permite almacenar temporalmente un dato durante la ejecución de un proceso, su contenido puede cambia durante la ejecución del programa. Para poder reconocer una variable en la memoria de la computadora, es necesario darle un nombre con el cual podamos identificarla dentro de un algoritmo.

Ejemplo: área = pi * radio*radio
Las variables son: el radio, el área y la constate es pi

Clasificación de las Variables

- Por su Contenido
  - Numéricas
  - Lógicas
  - Alfanuméricas (String) Variables
- Por su Uso
  - De Trabajo
  - Contadores
  - Acumuladores

Por su Contenido

Variable Numéricas: Son aquellas en las cuales se almacenan valores numéricos, positivos o negativos, es decir almacenan números del 0 al 9, signos (+ y -) y el punto decimal.
Ejemplo:
iva=0.16 pi=3.1416 costo=2500

Variables Lógicas: Son aquellas que solo pueden tener dos valores (cierto o falso) estos representan el resultado de una comparación entre otros datos.

Variables Alfanuméricas: Está formada por caracteres alfanuméricos (letras, números y caracteres especiales).
Ejemplo:
letra="a" apellido="lopez" direccion="Av. Libertad #190"

Por su Uso

Variables de Trabajo: Variables que reciben el resultado de una operación matemática completa y que se usan normalmente dentro de un programa.
Ejemplo:
suma=a+b/c

Contadores: Se utilizan para llevar el control del número de ocasiones en que se realiza una operación o se cumple una condición. Con los incrementos generalmente de uno en uno.

Acumuladores: Forma que toma una variable y que sirve para llevar la suma acumulativa de una serie de valores que se van leyendo o calculando progresivamente.

Variables Centinelas: También llamados “banderas”, los centinelas son variables, normalmente de tipo lógicas (boolean), conservan un estado hasta que un evento requiera cambiarlo para finalizar un ciclo.

1.2.14 Expresiones, Operadores y Operandos

Expresiones

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales.
Por ejemplo: a+(b + 3)/c.

Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas. Una expresión consta de operadores y operandos. Según sea el tipo de datos que manipulan, se clasifican las expresiones en:

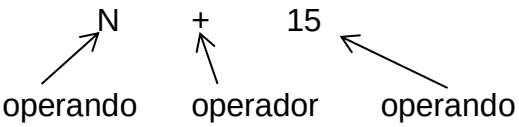
- Aritméticas
- Relaciónales
- Lógicas

Operadores y Operandos

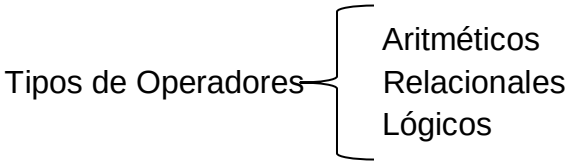
Operadores: Son símbolos que expresan operaciones que se aplican a uno o varios operandos y que devuelven un valor.

Operando: Puede ser una constante, una variable o el resultado de una función.

Expresión con un operador y operandos



Tipos de Operadores: Aritmeticos, Relacionales, Logicos



Operadores Aritméticos:

Los operadores aritméticos permiten la realización de operaciones matemáticas con los valores (variables y constantes).
Operadores aritméticos básicos: Suma +, Resta -, Multiplicación *, División /, Módulo %.

Prioridad de los Operadores Aritméticos

Todas las expresiones entre paréntesis se evalúan primero. Las expresiones con paréntesis anidados se evalúan de dentro a fuera, el paréntesis más interno se evalúa primero.
Dentro de una misma expresión los operadores se evalúan en el siguiente orden.

1.- () paréntesis
2.- *, /, % Multiplicación, división, modulo.
3.- +, - Suma y resta.

Los operadores en una misma expresión con igual nivel de prioridad se evalúan de izquierda a derecha.

Ejemplos:

4 + 2 * 5 = 14	4+10=14
23 * 2 / 5 = 9.2	46 / 5 = 9.2
3 + 5 * (10 - (2 + 4)) = 23	3 + 5 * (10 - 6) = 3 + 5 * 4 = 3 + 20 = 23
3.5 + 5.09 - 14.0 / 40 = 5.09	3.5 + 5.09 - 3.5 = 8.59 - 3.5 = 5.09
2.1 * (1.5 + 3.0 * 4.1) = 28.98	2.1 * (1.5 + 12.3) = 2.1 * 13.8 = 28.98

Operadores Relaciónales:

Se utilizan para establecer una relación entre dos valores.

Compara estos valores entre si y esta comparación produce un resultado de certeza o falsedad (verdadero o falso).

Los operadores relaciónales comparan valores del mismo tipo (numéricos o cadenas).

Tienen el mismo nivel de prioridad en su evaluación.

Los operadores relaciónales tiene menor prioridad que los aritméticos.

> Mayor que
< Menor que
> = Mayor o igual que
< = Menor o igual que
!= Diferente que
== Igualdad
= Asignación

Operadores Lógicos:

Estos operadores se utilizan para establecer relaciones entre valores lógicos.

Estos valores pueden ser resultado de una expresión relacional. En la tabla 1.1 se muestra el valor de verdad de una proposición compuesta, para cada combinación de verdad que se pueda asignar.

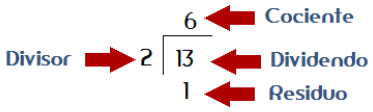
Los operadores lógicos básicos son tres:
And Y (&&)
Or O (||)
Not Negación (!)

Tabla 1.1 Operadores Lógicos y Tabla de Verdad

Operador And (&&)	Operador Or(||)	Operador Not (!)
T AND T =T	T OR T =T	T NOT F
T F =F	T F =T	F T
F T =F	F T =T	
F F =F	F F =F	

Como usar el mod (%).- El mod te da como resultado el residuo de una división entera ejemplo:

13%2= 1



En la tabla 1.2 se muestra la precedencia para todos los operadores.

Tabla 1.2 Precedencia de los Operadores Aritméticos, Relacionales y Lógicos

OPERADOR	PRECEDENCIA	NOMBRE
( )	Mayor	Parentesis
potencia		Potencia
*, / , %		Multiplicación División Mod
+, -		Suma, Resta
<, >, <=, >=		Relacionales
==, !=		Igualdad, Diferente que
&&,||,!	Menor	And,Or,Not
=		Asignación

Ejercicios Resueltos Capitulo 1

Ejercicios de Evaluación de Expresiones

1.1 Resuelve las siguientes operaciones utilizando las reglas de precedencia, donde:

Resuelve las siguientes operaciones utilizando las reglas de precedencia, donde: W = 5 , X = 7, Y = 3, Z = 9	
A = Y - Z * X + W / 3	A= 3-9*7+5/3 A= 3-63+5/3 A= 3-63+1.67 A= -60+1.67 A=-58.33
A = X * (Z – Y)/ W	A=7*(9-3)/5 A=7*6/5 A=42/5 A=8.4
A = Z * W – X + Y / Z	A=9*5-7+3/9 A=45-7+3/9 A=45-7+0.333 A=38+0.333 A=38.333
A = Z + W % Y	A=9+5%3 A=9+2 A=11
A = (4 * Y + Z % W) * X	A=(4*3+9%5)*7 A=(12+9%5)*7 A=(12+4)*7 A=16*7 A=112

1.2 Expresa las siguientes fórmulas para que las entienda la computadora.

Expresa las siguientes fórmulas para que las entienda la computadora.	
$A = \pi r^2$	A=3.1416*r*r
$x = \frac{4ac + (a + c) - 2ab}{2ab}$	X=(4*a*c+(a+c)-2*a*b)/(2*a*b)
$X = 5(y^2 + z^3 - 5zw + 3)$	X=5*(y*y+z*z*z-5*z*w+3)

1.3 Supongamos que tenemos la siguiente fórmula: Z = 4 <= 2 == 6!= 8 > 10, evaluarla y obtener el valor de Z.

Supongamos que tenemos la siguiente fórmula: Z = 4 <= 2 == 6 != 8 > 10, evaluarla y obtener el valor de Z.			
ACTIVIDAD		**OPERACIÓN**	**RESULTADO**
1.	Realiza la comparación de mayor precedencia de la izquierda.	Z = 4 <= 2 == 6 != 8 > 10	Z = 0 == 6 != 8 > 10
2.	Realiza la comparación de mayor precedencia.	Z = 0 == 6 != 8 > 10	Z = 0 == 6 != 0
3.	Realiza la comparación de mayor precedencia de la izquierda.	Z = 0 == 6 != 0	Z = 0 != 0
4.	Realiza la comparación.	Z = 0 != 0	**Z = 0**

1.4 Supongamos que tenemos la siguiente fórmula: Z = 8 == (9 + (1!= 0)) > 3 * 5 evaluarla y obtener el valor de Z.

Supongamos que tenemos la siguiente fórmula: Z = 8 == (9 + (1 != 0)) > 3 * 5 evaluarla y obtener el valor de Z.			
ACTIVIDAD		**OPERACIÓN**	**RESULTADO**
1.	Realiza la operación dentro del paréntesis más interno.	Z = 8 == (9 + (1 != 0)) > 3 * 5	Z = 8 == (9 + 1) > 3 * 5
2.	Realiza la operación dentro del paréntesis.	Z = 8 == (9 + 1) > 3 * 5	Z = 8 == 10 > 3 * 5
3.	Realiza la multiplicación.	Z = 8 == 10 > 3 * 5	Z = 8 == 10 > 15
4.	Realiza la comparación de mayor precedencia.	Z = 8 == 10 > 15	Z = 8 == 0
5.	Realiza la comparación.	Z = 8 == 0	**Z = 0**

1.3 Diseño de Algoritmos

Diagrama de flujo.- Es una representación gráfica de un algoritmo que se caracteriza por usar símbolos gráficos, para expresar de forma sencilla y clara el orden lógico en el que se realizan las acciones de un algoritmo.

1.3.1 Simbologia de Diagrama de Flujo

Los símbolos utilizados han sido normalizados por el instituto norteamericano de normalización (ANSI) figura 1.4.

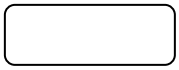
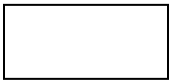
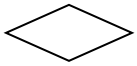
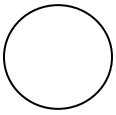
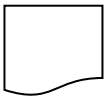
 ***Óvalo*****:** Inicio y término (Abre y/o cierra el diagrama).	 ***Rectángulo:*** Actividad (Representa la ejecución de una o más actividades o procedimientos).
 ***Rombo:*** Decisión (Formula una pregunta o cuestión).	 ***Círculo:*** Conector (Representa el enlace de actividades con otra dentro de un procedimiento).
Conector de pagina: Representa la continuidad en otra pagina	 ***salida:*** Representa la salida por impresora.
Salida: Representa salida a monitor	***Entrada***: Representa la entrada de datos por medio del teclado.

Fig 1.4 Simbología de Diagramas de Flujo.

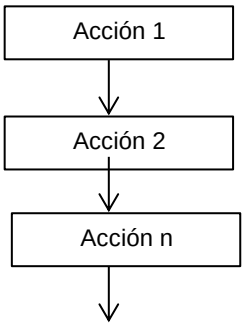
1.3.2 Estructuras Algoritmicas

Las estructuras algorítmicas son un grupo de formas de trabajo, que permiten, mediante la manipulación de variables, realizar ciertos procesos específicos que nos lleven a la solución de problemas. Estas estructuras se clasifican de acuerdo con su complejidad en: Estructuras de control secuencial, selectivas y repetitivas.

1.3.2.1 Estructuras Secuenciales

La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso.

Diagrama de flujo de la representación de una estructura secuencial:



1.3.2.2 Estructuras Selectivas: si, si sino, según sea el caso

Las estructuras selectivas comparan una variable contra otro(s) valor(es), para que en base al resultado de esta comparación, se siga un curso de acción dentro del programa. Cabe mencionar que la comparación se puede hacer contra otra variable o contra una constante, según se necesite. Existen dos tipos básicos, las simples y las múltiples.

Diagrama de flujo de la representación de un si:

Si expresión lógica es verdadera, se ejecuta la acción. Cuando resulta falsa, se termina el si y ejecuta la siguiente sentencia del diagrama.

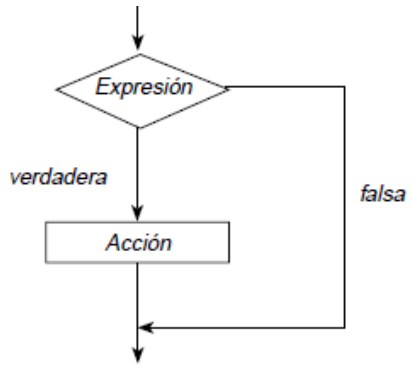


Diagrama de flujo de la representación de un si-sino:

Si expresión lógica es verdadera, se ejecuta sentencia1, si es falsa, se ejecuta sentencia2.

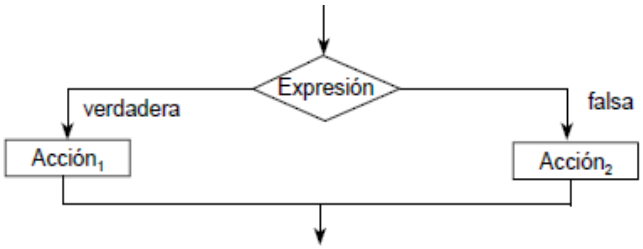
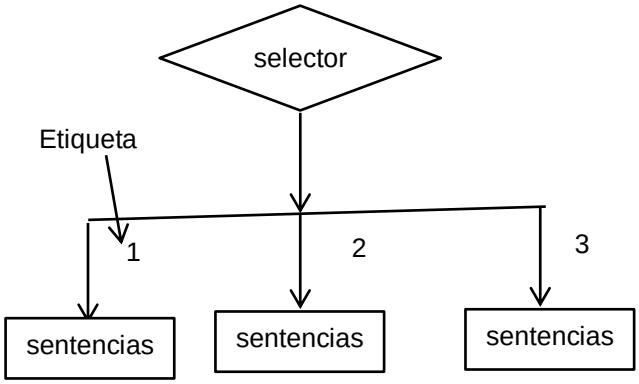


Diagrama de flujo de la representación de un según sea el caso:

La expresión de control o *selector* se evalúa y se compara con cada una de las etiquetas del **caso**. Cada *etiqueta* es un valor único, constante, y cada etiqueta debe tener un valor diferente de los otros. Si el valor de la expresión selector es igual a una de las etiquetas, entonces la ejecución comenzará con la primera sentencia de la secuencia y continuará hasta que se encuentra un fin.



1.3.2.3 Estructuras Repetitivas: mientras que, hasta que, desde, hasta que

Se llaman problemas repetitivos o cíclicos a aquellos en cuya solución es necesario utilizar un mismo conjunto de acciones que se puedan ejecutar una cantidad específica de veces. Esta cantidad puede ser fija (previamente determinada por el programador) o puede ser variable (estar en función de algún dato dentro del programa conocido como variables centinelas).

Diagrama de flujo de la representación de un mientras que:

El cuerpo del bucle se repite **mientras** que la expresión lógica (condición del bucle) sea verdadera. Cuando se evalúa la expresión lógica y resulta falsa, se termina y se sale del bucle y se ejecuta la siguiente sentencia del diagrama.

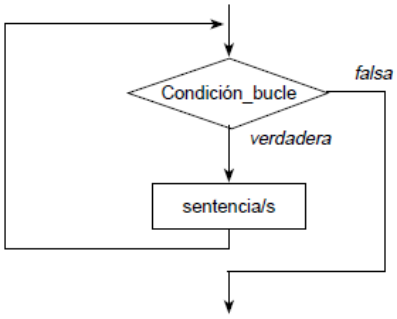


Diagrama de flujo de la representación de un hasta que:

La construcción de un **hasta que,** comienza ejecutando sentencia. Se evalúa a continuación expresión. Si expresión es verdadera, entonces se repite la ejecución de sentencia. Este proceso continúa hasta que expresiones falsa.

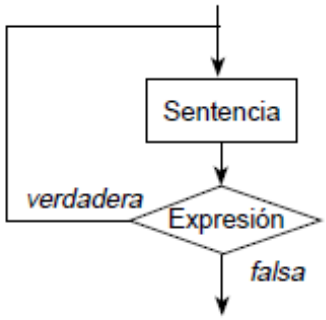
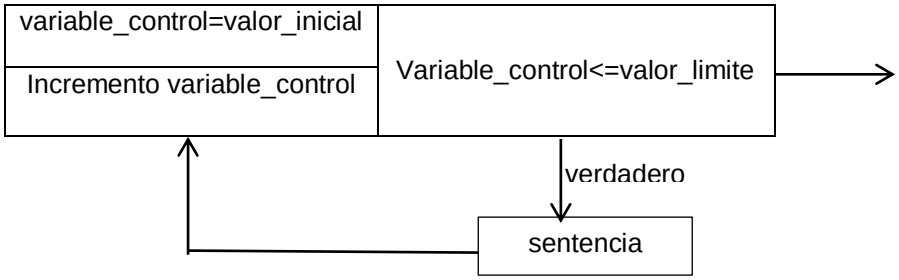


Diagrama de flujo de la representación de un desde, hasta que:

En este bucle la variable de control toma el valor inicial del bucle y se repite hasta que la variable de control llegue al valor límite superior.



1.3.3 Ejemplo de Diseño de Diagrama de Flujo

Definición del problema:
Desarrolle un algoritmo que le permita calcular el área de un triángulo en función de las longitudes de sus lados previamente leídos desde el teclado.

Análisis del problema:

Datos de Entrada: Aquí van todos los datos que necesito saber para resolver el problema y que desconozco porque el enunciado no me los da, y necesito ingresarlos por medio del dispositivo de entrada el teclado, el dato ingresado por teclado se le conoce como dato de entrada.
El dato ingresado por teclado necesitamos identificarlo mediante una variable.

Ejemplo:
Datos de Entrada:
b.- es la base del triángulo, numérica
h.- es la altura del triángulo, numérica

Proceso: Aquí van todos los cálculos matemáticos. Respetando la prioridad de los operadores aritmético.
Operadores del mismo nivel de prioridad que se encuentren en una misma expresión matemática se empieza a resolver de izquierda a derecha.

Proceso:
a=b*h/2

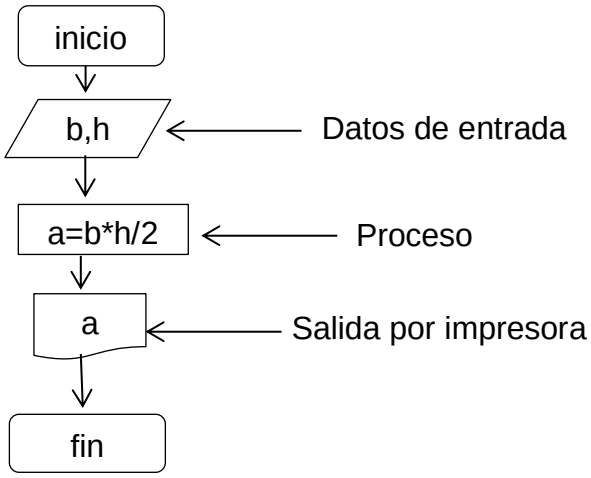
La multiplicación y la división, tienen el mismo nivel de prioridad, de izquierda a derecha primero multiplica y después divide.

La variable a, es una variable de trabajo, y recibe su valor por medio del resultado de una operación matemáticas.

Salida: Aquí va todo lo que queremos mandar a imprimir mediante un dispositivo de salida.

Salida:
a

Diseño del algoritmo (diagrama de flujo):

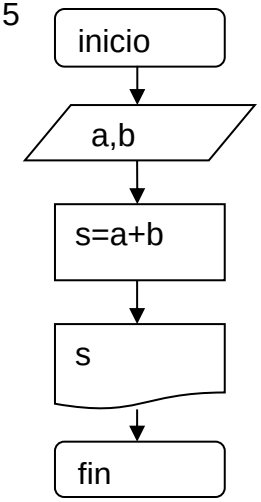


Ejercicio de Diseño de Diagramas de Flujo, Estructuras Secuenciales

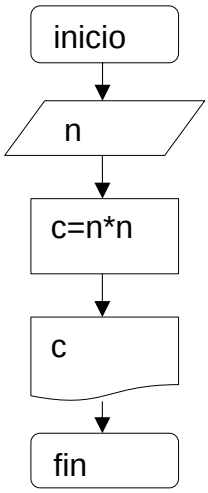
1.5. Desarrolle un algoritmo que le permita leer dos valores y escribir la suma de los dos.
1.6. Desarrolle un algoritmo que le permita leer un valor entero, calcular su cuadrado y escribir dicho resultado.
1.7 Desarrolle un algoritmo que le permita leer un valor para radio (R), calcular el área (A) de un círculo $A = \pi * r^2$ y escribir su valor.
1.8. Determinar la hipotenusa de un triángulo rectángulo conocidas las longitudes de sus dos catetos. Desarrolle los correspondientes algoritmos.
1.9. Desarrolle un algoritmo que le permita leer un valor que represente una temperatura expresada en grados Celsius y convierta dicho valor en un valor expresado en grados Fahrenheit.
1.10. Desarrolle un algoritmo que le permita calcular el área de un triángulo en función de las longitudes de sus lados previamente leídos desde el teclado.
1.11. Desarrolle un algoritmo que le permita determinar el área y volumen de un cilindro cuyo radio (R) y altura (H) se leen desde teclado.
1.12. Desarrolle un algoritmo que le permita calcular el área de un trapecio.

Resultados:

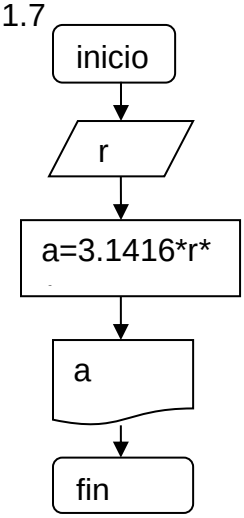
1.5



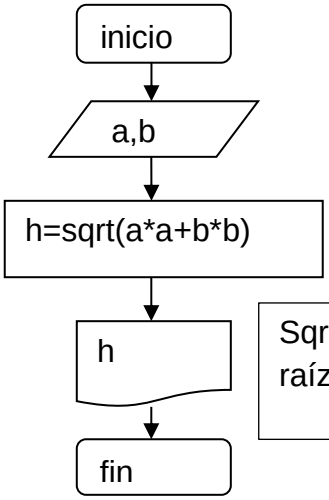
1.6



1.7

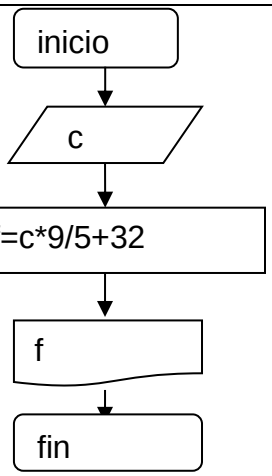


1.8

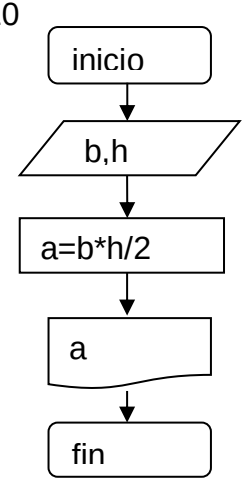


Sqrt.- calcula la raíz cuadrada

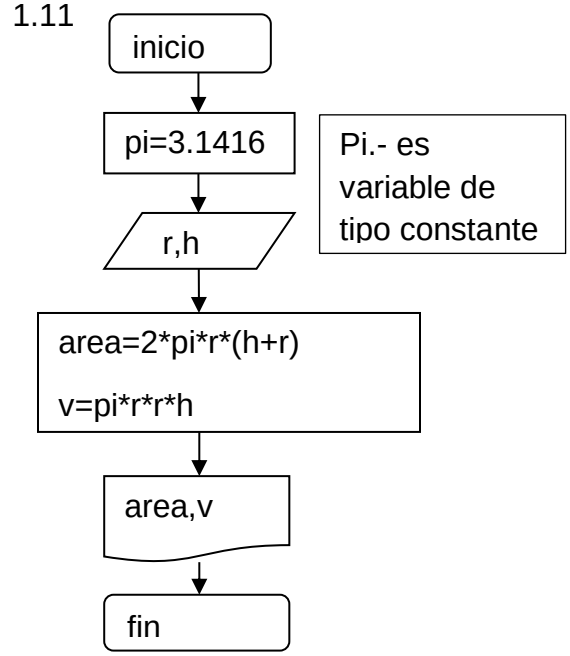
1.9



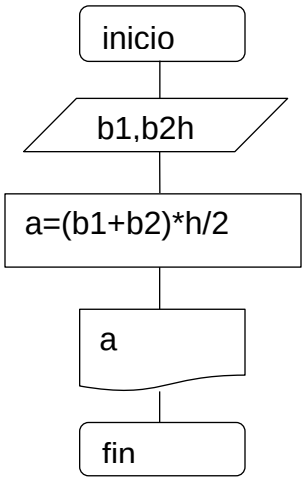
1.10



1.11

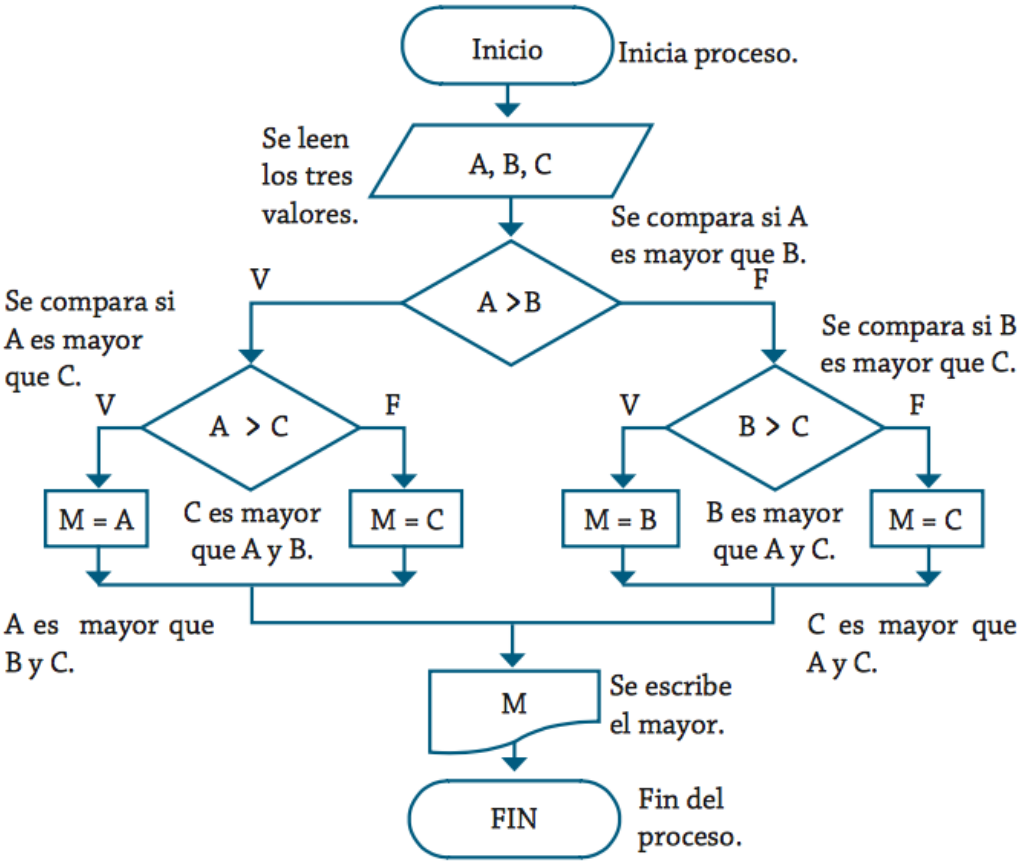

inicio
pi=3.1416
Pi.- es variable de tipo constante
r,h
area=2*pi*r*(h+r)
v=pi*r*r*h
area,v
fin

1.12


inicio
b1,b2h
a=(b1+b2)*h/2
a
fin

Ejercicios de Diseño de Algoritmos, Estructuras Selectivas y Repetitivas

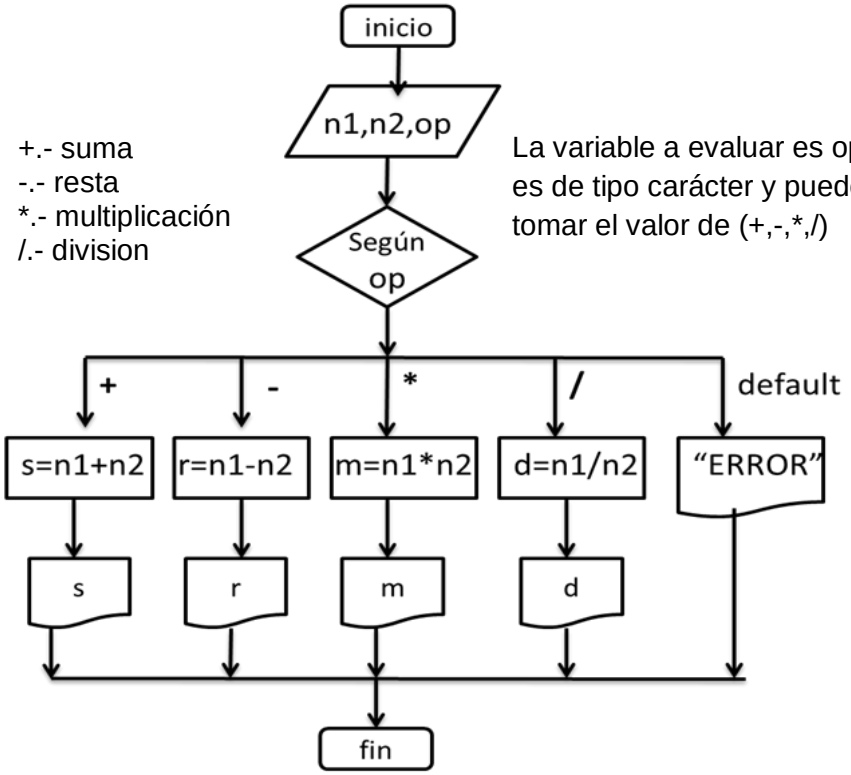
1.13 Leer tres números diferentes e imprimir el número mayor de los tres.



1.14 Realizar un programa de calculadora con las operaciones básicas
+.- suma
-.- resta
*.-multiplicación
/.- división
Teclee dos Números Y el operador deseado, realizar la operación según el operador.

+.- suma
-.- resta
*.- multiplicación
/.- division

La variable a evaluar es op y es de tipo carácter y puede tomar el valor de (+,-,*,/)



1.15 Realizar un programa de calculadora con las operaciones básicas

1.- suma

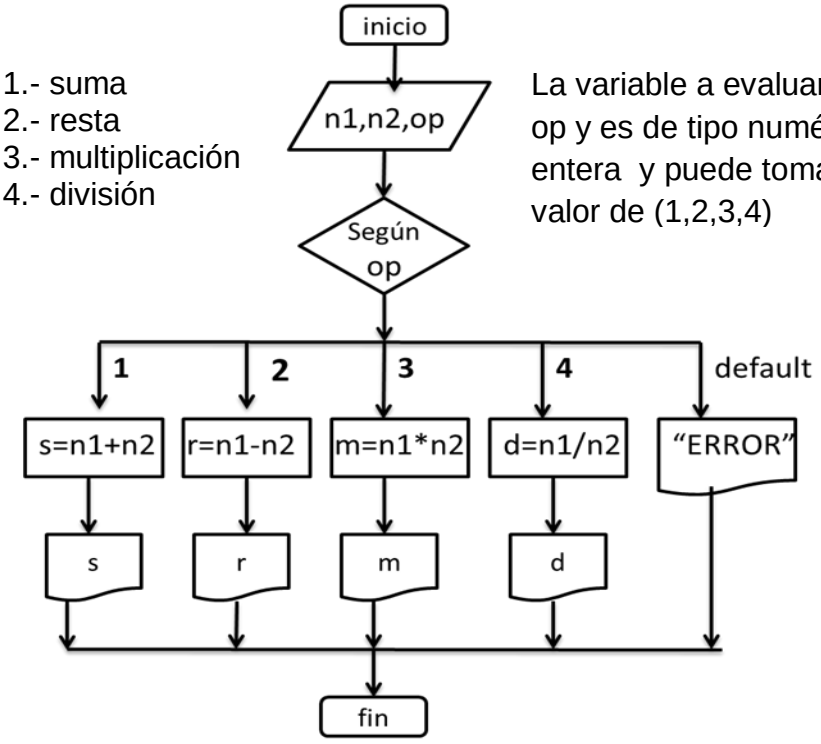
2.- resta

3.- multiplicación

4.- división

Teclee dos números y el número de la opción deseada, realizar la operación según el número de la opción

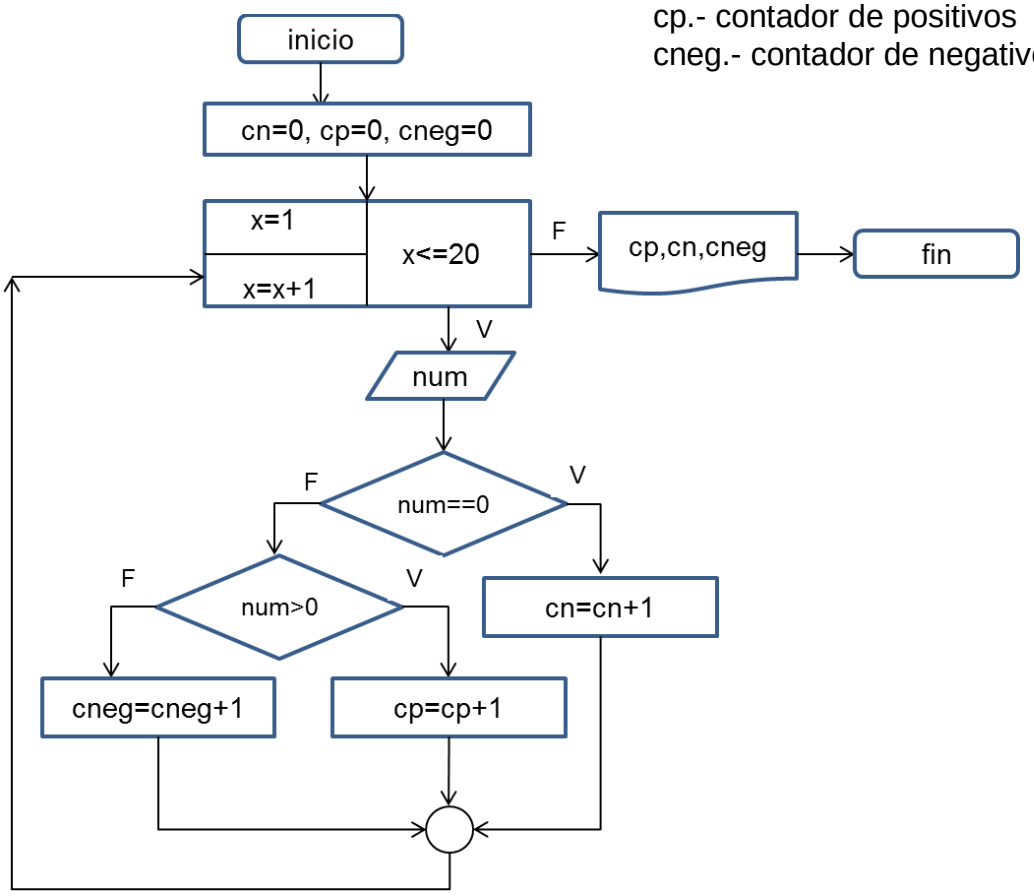
1.- suma
2.- resta
3.- multiplicación
4.- división



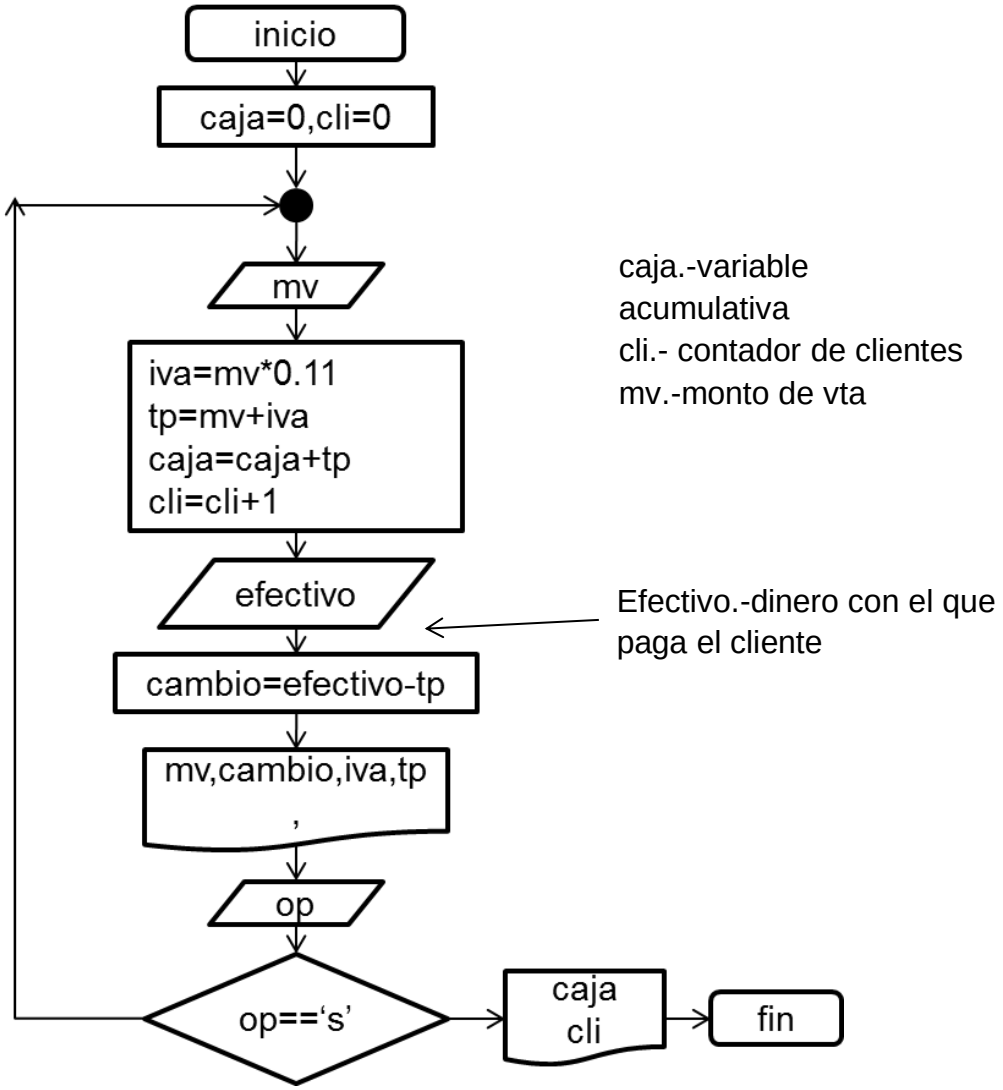
La variable a evaluar es op y es de tipo numérica entera y puede tomar el valor de (1,2,3,4)

1.16 Leer 20 números e imprimir cuantos son positivos, cuantos negativos y cuantos neutros.

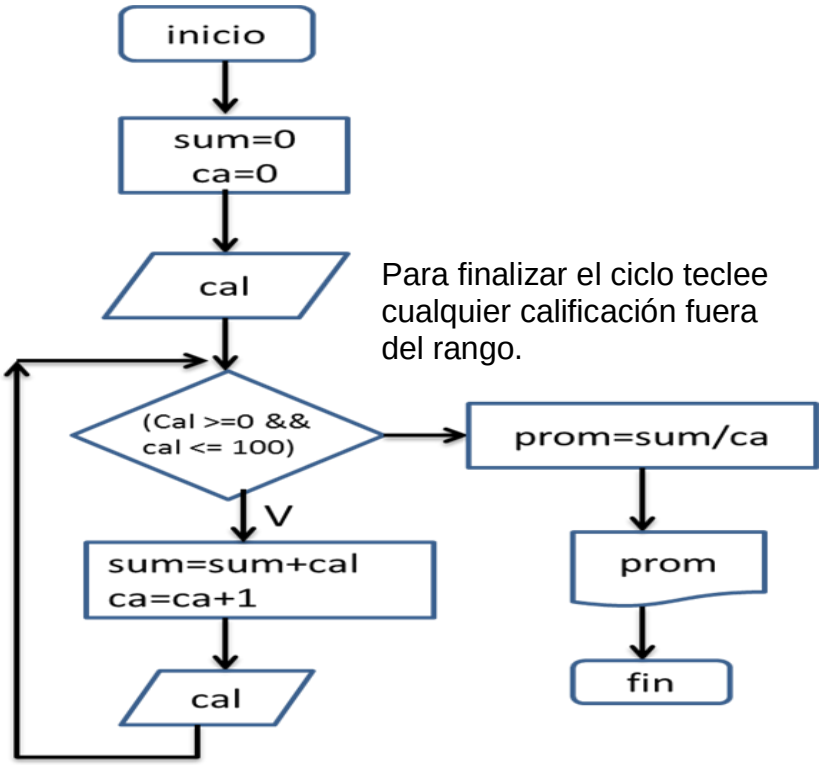
cn.- contador de neutros
cp.- contador de positivos
cneg.- contador de negativos



1.17 Suponga que tiene una tienda y desea registrar sus ventas por medio de una computadora. Diseñe un pseudocódigo que lea por cada cliente:
a).- el monto de la venta,
b).- calcule e imprima el IVA ,
c).- calcule e imprima el total a pagar,
d).- lea la cantidad con que paga el cliente,
e).-calcule e imprime el cambio.
Al final del día deberá imprimir la cantidad de dinero que debe haber en la caja.

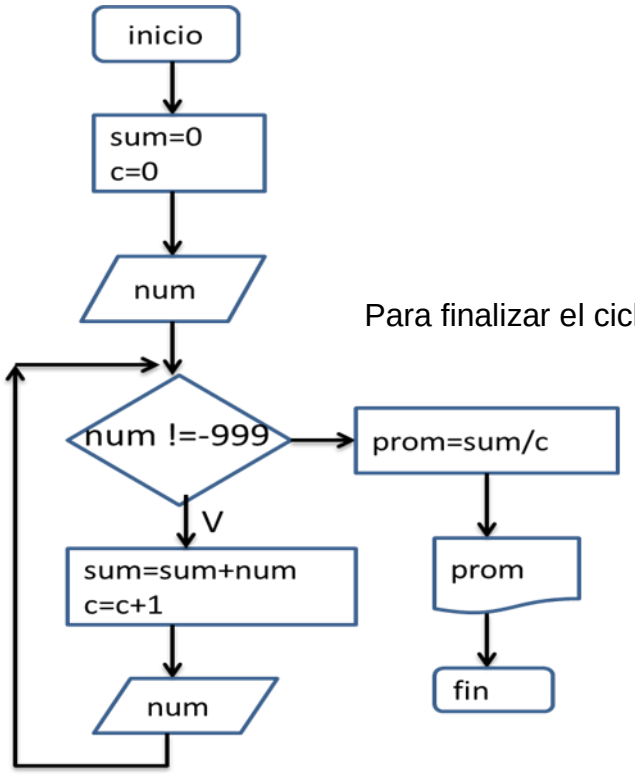


1.18 Obtener el promedio de calificaciones de un grupo de *n* alumnos.



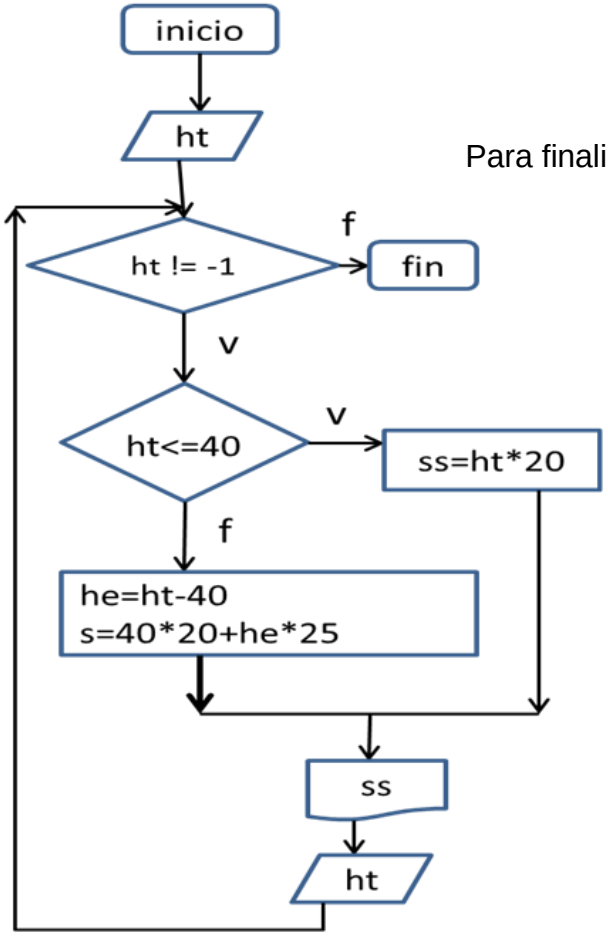
Para finalizar el ciclo teclee cualquier calificación fuera del rango.

1.19 Obtener el promedio de números de un conjunto de *n números*.



Para finalizar el ciclo teclee -999

1.20 En una empresa se requiere calcular el salario semanal de cada uno de los n obreros que laboran en ella. El salario se obtiene de la sig. forma:
Si el obrero trabaja 40 horas o menos se le paga $20 por hora
Si trabaja más de 40 horas se le paga $20 por cada una de las primeras 40 horas y $25 por cada hora extra.



Ejercicios Propuestos Para Solucionar por el Alumno Capitulo 1

1) Suponga que un individuo desea invertir su capital en un banco y desea saber cuánto dinero ganara después de un mes si el banco paga a razón de 2% mensual.

2) Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuánto deberá pagar finalmente por su compra.

3) Un maestro desea saber qué porcentaje de hombres y que porcentaje de mujeres hay en un grupo de estudiantes.

4) La presión, el volumen y la temperatura de una masa de aire se relacionan por la fórmula: masa = (presión * volumen)/(0.37 * (temperatura + 460))

5) Calcular el número de pulsaciones que una persona debe tener por cada 10 segundos de ejercicio, si la fórmula es: num. pulsaciones = (220 - edad)/10

6) Calcular el nuevo salario de un obrero si obtuvo un incremento del 25% sobre su salario anterior.

7) En un hospital existen tres áreas: Ginecología, Pediatría, Traumatología. El presupuesto anual del hospital se reparte conforme a la sig. tabla:
Área Porcentaje del presupuesto
Ginecología 40%
Traumatología 30%
Pediatría 30%
Obtener la cantidad de dinero que recibirá cada área, para cualquier monto presupuestal.

8) El dueño de una tienda compra un artículo a un precio determinado. Obtener el precio en que lo debe vender para obtener una ganancia del 30%.

9) Todos los lunes, miércoles y viernes, una persona corre la misma ruta y cronometra los tiempos obtenidos. Determinar el tiempo promedio que la persona tarda en recorrer la ruta en una semana cualquiera.

10) Tres personas deciden invertir su dinero para fundar una empresa. Cada una de ellas invierte una cantidad distinta. Obtener el porcentaje que cada quien invierte con respecto a la cantidad total invertida.

2. INTRODUCCIÓN A LA PROGRAMACIÓN

2.1 Conceptos Básicos

El procesador debe ser capaz de *interpretar* el algoritmo, lo que significa:
• debe de comprender las instrucciones de cada paso,
• y realizar las operaciones correspondientes.

Cuando el procesador es una computadora, el algoritmo se ha de expresar en un formato que se denomina *programa,* ya que el algoritmo (pseudocódigo o el diagrama de flujo) no son comprensibles por la computadora, aunque pueda entenderlos cualquier programador.
Un programa se escribe en un *lenguaje de programación* y las operaciones que conducen a expresar un algoritmo en forma de programa se llaman *programación*. Así pues, los lenguajes utilizados para escribir programas de computadoras son los *lenguajes de programación* y **programadores** son los escritores y diseñadores de programas.
El proceso de traducir un algoritmo a un lenguaje de programación se denomina **codificación**, y el algoritmo escrito en un lenguaje de programación se denomina **código fuente**.
En la realidad la computadora no entiende directamente los lenguajes de programación sino que se requiere un programa que traduzca el código fuente a otro lenguaje que sí entiende la máquina directamente, pero muy complejo para las personas; este lenguaje se conoce como **lenguaje máquina** y el código correspondiente **código máquina**. Los programas que traducen el código fuente escrito en un lenguaje de programación —tal como C++— a código máquina se denominan **traductores**.
El proceso de conversión de un algoritmo hasta un programa ejecutable comprensible por la máquina, se muestra en la figura 2.1.

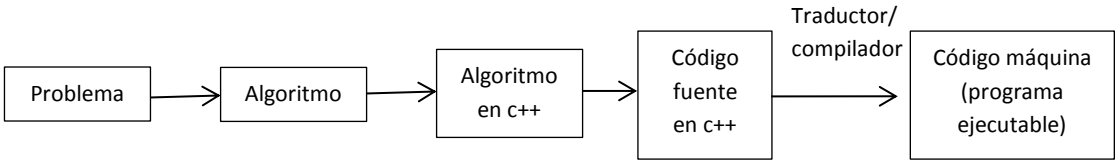


Fig 2.1 Proceso de Transformación de un Algoritmo en un Programa Ejecutable

2.1.1 Regla para Nombrar un Archivo Fuente

Al nombrar un archivo fuente se debe utilizar el sufijo adecuado al compilador C++ que esté utilizando.
De este modo se identifica el archivo como código fuente C++ y se le indica también al compilador para que lo reconozca. El sufijo consta de un punto seguido por un carácter o grupo de caracteres y se denomina *extensión*.

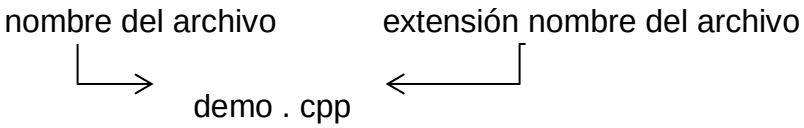


Fig 2.2. Nombre de un Archivo en Código Fuente

2.2 Características del Lenguaje de Programación

Bjarne Stroustrup de AT&T Bell Laboratories desarrolló C++ al principio de la década de los ochenta. Stroustrup diseñó C++ como un mejor C.

C++ es una extensión de C con características más potentes. Estrictamente hablando, es un superconjunto de C. Los elementos más importantes añadidos a C para crear clases C, objetos y programación orientada a objetos (C++ fue llamado originalmente “C con clases”).

C++ es un lenguaje orientado a objetos que se ha hecho muy popular y con una gran difusión en el mundo del software y actualmente constituye un lenguaje estándar para programación orientada a objetos, aunque también puede ser utilizado como lenguaje estructurado al estilo de C si se desea trabajar al modo clásico de programación estructurada, sobre todo si se desea trabajar con algoritmos y estructura de datos. La figura 2.3 muestra las relaciones entre C y C++.

2.2.1 Características de C y C++

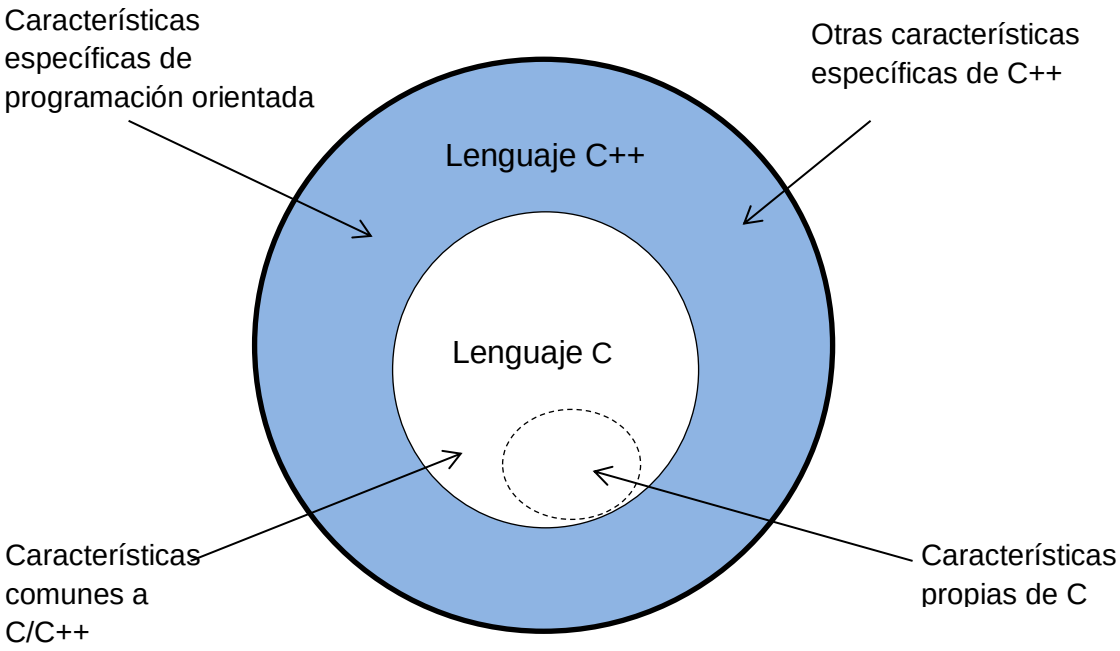


Fig 2.3 Características de C y C++

2.3 Estructura Básica de un Programa

La estructura típica completa de un programa C++ se muestra en la figura 2.4. Un ejemplo de un programa sencillo en C++.

```
1)  // El programa imprime "Bienvenido a la programación C++"
2)  #include <iostream>
3)  using namespace std;
4)  int main()
5)  {
6)     cout << "Bienvenido a la programación C++\n";
7)     return 0;
8)  }
```

Fig 2.4 Estructura Básica de un Programa

La primera línea es un *comentario*, identificado por dobles barras inclinadas (//). Los comentarios se incluyen en programas que proporcionan explicaciones a los lectores de los mismos. Son ignorados por el compilador. Para comentarios de varias líneas Los comentarios comienzan con la secuencia /* y terminan con la secuencia */.

La directiva #include de la segunda línea es necesaria para que el programa tenga salida. Se refiere a un archivo externo denominado iostream en el que se proporciona la información relativa al objeto cout. Obsérvese que los ángulos < y > no son parte del nombre del archivo; se utilizan para indicar que el archivo es un archivo de la biblioteca estándar C++.

La tercera línea es la directiva using namespace std**;** indica que todas las sentencias del programa que vienen a continuación están dentro del espacio de nombres std. Diversos componentes de programa, tales como cout, están declarados dentro de este espacio de nombres.

La cuarta línea contiene la cabecera de la función main(), obligatoria en cada programa de C++ contiene una función main() que es la función principal y que es el primer punto de entrada al programa.

La quinta y octava línea contienen sólo las llaves { y } que encierran el cuerpo de la función main() y son necesarias en todos los programas C++.

La sexta línea contiene la sentencia cout << "Bienvenido a la programación C++\n"; que indica al sistema que envía el mensaje "Bienvenido a la programación en C++\n" al objeto cout. Este objeto es el *flujo estándar de salida* que normalmente representa la pantalla de presentación de la computadora (el nombre cout es una abreviatura de *«console output»* —salida a consolas).

La salida será Bienvenido a la programación C++.

El símbolo '\n' es el símbolo de *nueva línea*. Poniendo este símbolo al final de la cadena entre comillas, indica al sistema que comience una nueva línea después de imprimir los caracteres precedentes, terminando, por consiguiente, la línea actual.

La séptima línea contiene la sentencia return 0. Esta sentencia termina la ejecución del programa y devuelve el control al sistema operativo de la computadora. El número 0 se utiliza para señalar que el programa ha terminado correctamente (*con éxito*).

La sentencia de salida de la sexta línea incluye diversos símbolos típicos de C++. El símbolo << se denomina *operador de salida* u *operador de inserción*. Inserta el mensaje en el flujo de salida. El símbolo \n incluido al final del mensaje significa *carácter de nueva línea*. Siempre que aparece en un mensaje de salida, se termina la línea actual de salida y comienza una nueva línea.
Obsérvese el punto y coma (;) al final de la sexta y séptima línea. C++ requiere que cada sentencia termine con un punto y coma. No es necesario que esté al final de una línea. Se pueden poner varias sentencias en la misma línea y se puede hacer que una sentencia se extienda sobre varias líneas.

2.3.1 Entrada (cin)

El archivo iostream define cin como un objeto que representa el flujo. Para la entrada, cin utiliza el operador >> para extraer caracteres del flujo de entrada y por ello se denomina *operador de extracción*. Al igual que cout, es un objeto inteligente. Convierte la entrada, una serie de caracteres escritos desde el teclado, en un formato aceptable a la variable donde se almacena la información. Si no se redirige explícitamente cin, la entrada procede del teclado variable.

2.3.2 Declaración de una Variable

Una *declaración* de una variable es una sentencia que proporciona información de la variable al compilador C++. Su sintaxis es:

tipo variable;
tipo.- *es el nombre de un tipo de dato conocido por C++*
variable.- *es un identificador (nombre) válido en C++.*

Ejemplos
int dNumero;
double HorasAcumuladas;
float HorasPorSemana;

Ejemplo de un programa que suma el valor de 2 números, figura 2.5.

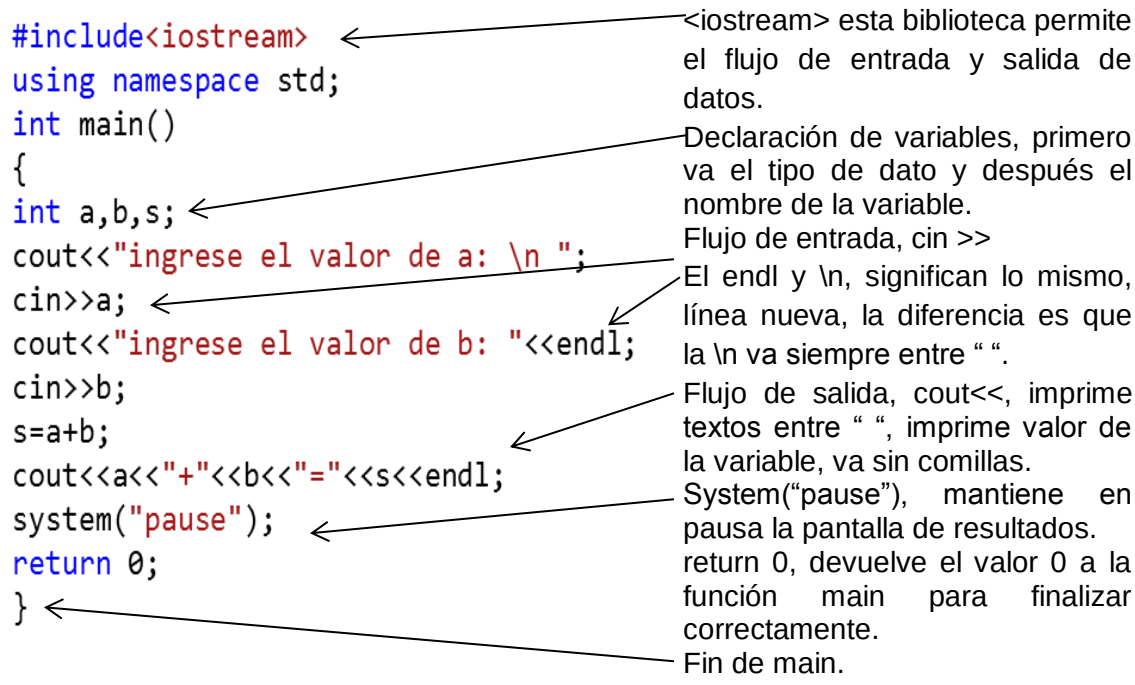


Fig 2.5 Ejemplo de un Programa

2.4 Elementos del Lenguaje

2.4.1 Tipos de Datos Básicos en C++

Tipo de dato	Descripción
int	(Números enteros)
float	(Números reales)
doublé	(Números reales más grandes que float)
bool	(Valores lógicos)
char	(Caracteres y cualquier cantidad de 8 bits)
void	(Nada. Sirve para indicar que una función no devuelve valores)

Fig 2.6 Tipos de Datos Básicos en C++

2.4.2 Operadores Básicos en C++

Aritméticos	Relacionales	Lógicos
+ suma	== Igual a	Y (and) &&
- Resta	!= Distinto	0 (or) ||
* Multiplicación	>Mayor que	Negación (Not) !
/ División	< Menor que	
% Resto de la división entera (mod).	<= Menor o igual que	
	>= Mayor o igual que	
	= Asignación	

Fig 2.7 Operadores Básicos en C++

2.4.3 Operadores de Incremento y Decremento en C++

Operador	Descripción	Ejemplo
++	Incremento **++a**	Incrementa en 1 a **a,** luego utiliza el nuevo valor de **a.** **a=a+1**
++	**a++**	Toma el valor actual de **a**, luego incrementa en 1 a **a**.
- -	Decremento **- - a**	decrementa en 1 a **a,** luego utiliza el nuevo valor de **a.** **a=a-1**
- -	**a - -**	Toma el valor actual de **a**, luego decrementa en 1 a **a**.

Fig 2.8 Operadores de Incremento y Decremento

2.4.4 Operadores de Asignación de C++

Símbolo	Uso	Descripción
=	a = b	Asigna el valor de *b* a *a*.
***=**	a *= b	Multiplica a por *b* y asigna el resultado a la variable *a*.
/=	a /= b	Divide *a* entre *b* y asigna el resultado a la variable *a*.
%=	a %= b	Fija a al resto de *a*/*b*.
+=	a += b	Suma *b* y *a* y lo asigna a la variable *a*.
-=	a -= b	Resta *b* de *a* y asigna el resultado a la variable *a*.

Fig 2.9 Operadores de Asignación de C++

2.5 Traducción de un Programa: Compilación, Enlace, Ejecución y Errores.

2.5.1 La Compilación y sus Fases

La *compilación* es el proceso de traducción de programas fuente a programas objeto.
El proceso de ejecución de un programa escrito en un lenguaje de programación y mediante un compilador suele tener los siguientes pasos:
1. Escritura del *programa fuente* con un *editor* (programa que permite a una computadora actuar de modo similar a una máquina de escribir electrónica) y guardarlo en un dispositivo de almacenamiento.
2. Introducir el programa fuente en memoria.
3. *Compilar* el programa con el compilador C++.
4. *Verificar* y *corregir errores de compilación* (listado de errores).
5. Obtención del programa *objeto*.
6. El enlazador (*linker*) obtiene el *programa ejecutable*.
7. Se ejecuta el programa y, si no existen errores, se tendrá la salida del programa.

Figura 2.10 Etapas de Ejecución de un Programa

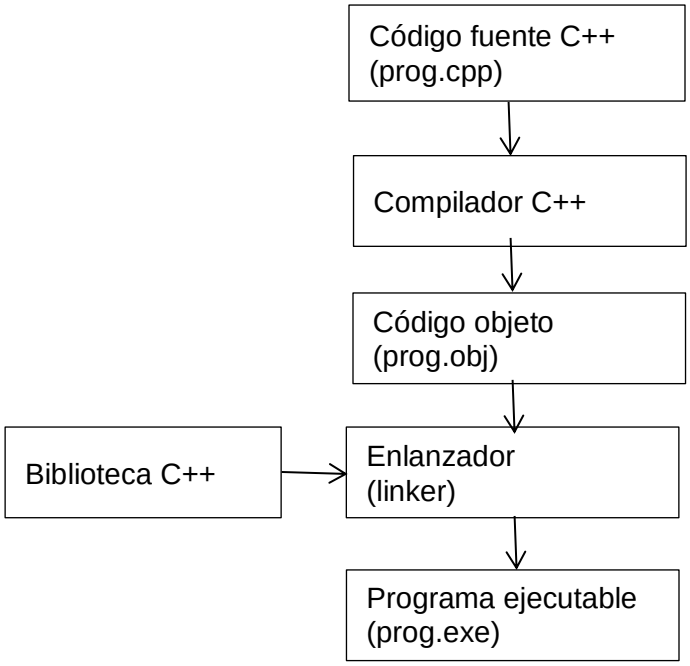


Fig 2.10 Etapas en la Ejecución de un Programa

2.5.2 Errores de Sintaxis y Lógicos

Los **errores de sintaxis** son aquellos que se producen cuando el programa viola la sintaxis, es decir, las reglas de gramática del lenguaje. Errores de sintaxis típicos son: escritura incorrecta de palabras reservadas, omisión de signos de puntuación (comillas, punto y coma...). Los errores de sintaxis son los más fáciles de fijar, ya que ellos son detectados y aislados por el compilador.

Un segundo tipo de error importante es el **error lógico**, ya que tal error representa errores del programador en el diseño del algoritmo y posterior programa. Los errores lógicos son más difíciles de encontrar y aislar, ya que no suelen ser detectados por el compilador.

Mensajes de error

Los compiladores emiten mensajes de error o de advertencia durante las fases de compilación, de enlace o de ejecución de un programa.

Los mensajes de error producidos durante la compilación se suelen producir, normalmente, por errores de sintaxis y suele variar según los compiladores; pero, en general, se agrupan en tres bloques:

Errores fatales. Son raros. Algunos de ellos indican un error interno del compilador. Cuando ocurre un error fatal, la compilación se detiene inmediatamente, se debe tomar la acción apropiada y, a continuación, se vuelve a iniciar la compilación.

Errores de sintaxis. Son los errores típicos de sintaxis, errores de línea de órdenes y errores de acceso a memoria o disco. El compilador terminará la fase actual de compilación y se detiene.

Advertencias (*warning*). No impiden la compilación. Indican condiciones que son sospechosas, pero son legítimas como parte del lenguaje.

Ejercicios Resueltos Capitulo 2

2.1 Desarrolle un algoritmo que le permita leer dos valores y escribir la suma de los dos.

```
//ejercicio 2.1
#include <iostream>
using namespace std;
void main()
{
	int a,b,s;
	cout<<"ingresa a valor de a \n";
	cin>>a;
	cout<<"ingrese el valor de b \n";
	cin>>b;
	s=a+b;
	cout<<a<<"+"<<b<<"="<<s<<endl;
	system("pause");
}
```

2.2 Desarrolle un algoritmo que le permita leer un valor entero, calcular su cuadrado y escribir dicho resultado.

```
//ejercicio 2.2
#include <iostream>
using namespace std;
void main()
{
	int a,c;
	cout<<"ingresa numero \n";
	cin>>a;
	c=a*a;
	cout<<a<<"^2="<<c<<endl;
	system("pause");
}
```

2.3 Desarrolle un algoritmo que le permita leer un valor para radio (R), calcular el área (A) de un círculo A =π*R² y escribir su valor.

```
//ejercicio 2.3
#include <iostream>
using namespace std;
void main()
{
	float a,r,pi=3.1416;
	cout<<"ingresa r: \n";
	cin>>r;
	a=pi*r*r;
	cout<<"area="<<a<<endl;
	system("pause");
}//fin de main
```

2.4 Determinar la hipotenusa de un triángulo rectángulo conocidas las longitudes de sus dos catetos. Desarrolle los correspondientes algoritmos.

```
// ejercicio 2.4
#include<iostream>
#include<math.h>
using namespace std;
void main()
{
  float a,b,h;
  cout<<"ingrese a:"<<endl;
  cin>>a;
  cout<<"ingrese b:"<<endl;
  cin>>b;
  h=sqrt(a*a+b*b);
  cout<<"hipotenusa="<<h<<endl;
  system("pause");
}//fin de main
```

2.5 Desarrolle un algoritmo que le permita leer un valor que represente una temperatura expresada en grados Celsius y convierta dicho valor en un valor expresado en grados Fahrenheit.

```
//ejercicio 2.5
#include<iostream>
using namespace std;
void main()
{
        float c;
        cout<<"ingrese los centigrados \n";
        cin>>c;
        cout<<" fahrenheit="<<(1.8*c+32)<<endl;
   system("pause");
}//fin de main
```

2.6 Leer la capacidad de un disco duro en gigas bytes (GB), convertir e imprimirlo en megas bytes (MB), kilo bytes (KB) y bytes (B).

```
//ejercicio 2.6
#include <iostream>
using namespace std;
void main()
{
        float gb,mb,kb,b;
        cout<<“ingrese la capacidad del disco duro en GB \n“;
        cin>>gb;
        mb=gb*1024;
        kb=mb*1024;
        b=kb*1024;
        cout<<“MB= “<<mb<<”\n”;
        cout<<“KB=  “<<kb<<endl;
        cout<<“B = “<<b<<endl;
} // fin main
```

2.7 Calcular el salario neto de un trabajador, en función del número de horas trabajadas y el precio de la hora trabajada, y un descuento del 20% del impuesto.

```
//ejercicio 2.7
#include <iostream>
using namespace std;
void main()
{
        float ht,ph,sb,imp,st;
        cout<<“ingrese las horas trabajadas \n”;
        cin>>ht;
        cout<<“ingrese el pago por hora \n”;
        cin>>ph;
        sb=ht*ph;
        imp=sb*0.20;
        st=sb-imp;
        cout<<“EL SALARIO TOTAL ES:     “<<st<<endl;
}  //fin main
```

2.8 calcular el área de un triángulo.

```
//ejercicio 2.8
#include<iostream>
using namespace std;
void main()
{
        float a,b,h;
        cout<<"teclee base y altura? \n";
        cin>>b>>h;
        a=b*h/2;
        cout<<"area="<<a<<endl;
}//fin
```

2.9 Un vendedor recibe un sueldo base más un 10% extra por comisión de sus ventas, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá en el mes tomando en cuenta su sueldo base y comisiones.

```
//ejercicio 2.9
#include <iostream>
using namespace std;
void main()
{
	float sb,v1,v2,v3,com,ss;
	cout<<"Ingrese sueldo base \n";
	cin>>sb;
	cout<<"Ingrese las tres ventas \n";
	cin>>v1>>v2>>v3;
	com=(v1+v2+v3)*0.10;
	ss=sb+com;
	cout<<"El sueldo total es:  "<<ss<<endl;
}//Fin main
```

Ejercicios Propuestos Para Solucionar por el Alumno Capitulo 2

1) Suponga que un individuo desea invertir su capital en un banco y desea saber cuánto dinero ganara después de un mes si el banco paga a razón de 2% mensual.

2) Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuánto deberá pagar finalmente por su compra.

3) Un maestro desea saber qué porcentaje de hombres y que porcentaje de mujeres hay en un grupo de estudiantes.

4) La presión, el volumen y la temperatura de una masa de aire se relacionan por la fórmula:
masa = (presión * volumen)/(0.37 * (temperatura + 460))

5) Calcular el número de pulsaciones que una persona debe tener por cada 10 segundos de ejercicio, si la fórmula es: num. pulsaciones = (220 - edad)/10

6) Calcular el nuevo salario de un obrero si obtuvo un incremento del 25% sobre su salario anterior.

7) En un hospital existen tres áreas: Ginecología, Pediatría, Traumatología. El presupuesto anual del hospital se reparte conforme a la sig. tabla:
Área Porcentaje del presupuesto
Ginecología 40%
Traumatología 30%
Pediatría 30%
Obtener la cantidad de dinero que recibirá cada área, para cualquier monto presupuestal.

8) El dueño de una tienda compra un artículo a un precio determinado. Obtener el precio en que lo debe vender para obtener una ganancia del 30%.

9) Todos los lunes, miércoles y viernes, una persona corre la misma ruta y cronometra los tiempos obtenidos. Determinar el tiempo promedio que la persona tarda en recorrer la ruta en una semana cualquiera.

10) Tres personas deciden invertir su dinero para fundar una empresa. Cada una de ellas invierte una cantidad distinta. Obtener el porcentaje que cada quien invierte con respecto a la cantidad total invertida.

3. CONTROL DE FLUJO

Las **estructuras de control** controlan el flujo de ejecución de un programa o función. Las estructuras de control permiten combinar instrucciones o sentencias individuales en una simple unidad lógica con un punto de entrada y un punto de salida.
Las instrucciones o sentencias se organizan en tres tipos de estructuras de control que sirven para controlar el flujo de la ejecución: *secuencia*, *selección* (*decisión*) y *repetición*, el flujo secuencial. Una **sentencia compuesta** es un conjunto de sentencias encerradas entre llaves ({ y }) que se utiliza para especificar un flujo secuencial.

3.1 Estructuras Secuenciales

El control fluye de la *sentencia*1 a la *sentencia*2, y así sucesivamente. Sin embargo, existen problemas que requieren etapas con dos o más opciones o alternativas a elegir en función del valor de una condición o expresión.

```
{
   sentencia1;
   sentencia2;
   .
   .
   .
   sentencia n;
}
```

3.2 Estructuras Selectivas: if, if-else, switch

3.2.1 La Sentencia if

En C++, la estructura de control de selección principal es una sentencia if. La sentencia if tiene dos alternativas o formatos posibles. El formato más sencillo tiene la sintaxis siguiente:

Sintaxis

```
if (Expresión)  <-- Expresión lógica que determina si la acción se ha de ejecutar
 Acción;
```

Acción se ejecuta si la expresión lógica es verdadera

La sentencia if funciona de la siguiente manera. Cuando se alcanza la sentencia if dentro de un programa, se evalúa la *expresión* entre paréntesis que viene a continuación de if. Si *Expresión* es verdadera, se ejecuta Acción; en caso contrario no se ejecuta Acción (en su formato más simple, Acción es una sentencia simple, y en los restantes formatos, es una sentencia compuesta). En cualquier caso la ejecución del programa continúa con la siguiente sentencia del programa.

3.2.2 La Sentencia: Condición Doble if-else

Un segundo formato de la sentencia if es la sentencia if-else. Este formato de la sentencia if tiene la siguiente sintaxis:

Sintaxis

```
If (Expresión)   <-- Expresión lógica que determina la acción a ejecutar
        Acción1; <-- Acción que se ejecuta si la expresión lógica es verdadera
else
        Acción2; <-- Acción que se ejecuta si la expresión lógica es falsa
```

En este formato Acción1 y Acción2 son individualmente o bien una única sentencia que termina en un punto y coma (;) o un grupo de sentencias

encerrado entre llaves. Cuando se ejecuta la sentencia if else, se evalúa Expresión. Si Expresión es verdadera, se ejecuta Acción1 y en caso contrario se ejecuta Acción2.

3.2.3 La Sentencia: Condiciones Múltiples switch

La sentencia switch es una sentencia C++ que se utiliza para seleccionar una de entre múltiples alternativas.
La sentencia switch es especialmente útil cuando la selección se basa en el valor de una variable simple o de una expresión simple denominada *expresión de control* o *selector*. El valor de esta expresión puede ser de tipo int o char, pero no de tipo doublé, sintaxis switch:

Sintaxis

```
switch (selector)
{
        case etiqueta1 : sentencias1;
                         break;
        case etiqueta2 : sentencias2;
                         break;
        .
        .
        .
        case etiqueta n : sentenciasn;
                         break;
        default: sentenciasd; // opcional
}
```

La expresión de control o *selector* se evalúa y se compara con cada una de las etiquetas de case. La expresión *selector* debe ser un tipo ordinal (por ejemplo, int, char, bool pero no float o string).
Cada *etiqueta* es un valor único, constante, y cada etiqueta debe tener un valor diferente de los otros. Si el valor de la expresión selector es igual a una de las etiquetas case —por ejemplo, *etiquetai*— entonces la ejecución comenzará con la primera sentencia de la secuencia *secuenciai* y continuará hasta que se encuentra una sentencia break (o hasta que se encuentra el final de la sentencia de control switch).

3.3 Estructuras Iterativas: while, for, do-while

Un bucle es cualquier construcción de programa que repite una sentencia o secuencia de sentencias un número de veces. La sentencia (o grupo de sentencias) que se repiten en un bloque se denomina **cuerpo** del bucle y cada repetición del cuerpo del bucle se llama **iteración** del bucle. Las dos principales cuestiones de diseño en la construcción del bucle son: ¿Cuál es el cuerpo del bucle? ¿Cuántas veces se iterará el cuerpo del bucle?

3.3.1 Repetición: El Bucle while

El bucle while indica que la ejecución de la sentencia o sentencias expresadas se repite *mientras* la condición del bucle permanece verdadera y termina cuando se hace falsa. También indica que la condición del bucle se evalúa antes de que se ejecute el cuerpo del bucle y, por consiguiente, si esta condición es inicialmente falsa, el cuerpo del bucle no se ejecutará. En otras palabras, el cuerpo de un bucle while se ejecutará *cero* o *más veces*.

Sintaxis

1 while (condición_bucle)
  sentencia; ← cuerpo

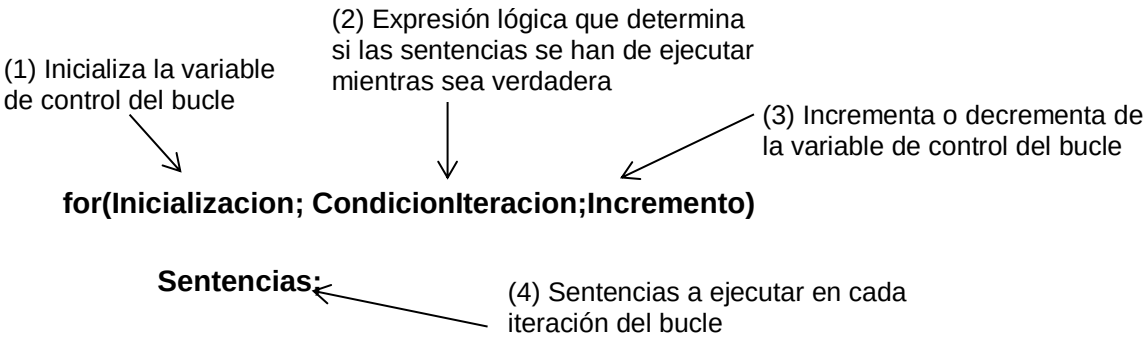
2 while (condición_bucle)
```
  {
     sentencia-1;
     sentencia-2;
     .
     .                    cuerpo
     .
     sentencia-n;
  }
```

While.- palabra reservada C++
condición_bucle.- expresión lógica o booleana
sentencia.- sentencia simple o compuesta

3.3.2 Repetición: El Bucle for

La sentencia **for** (bucle **for**) es un método para ejecutar un bloque de sentencias un número fijo de veces. El bucle for se diferencia del bucle while en que las operaciones de control del bucle se sitúan en un solo sitio: la cabecera de la sentencia.

Sintaxis



El bucle for contiene las cuatro partes siguientes:

• *Parte de inicialización*, que inicializa las variables de control del bucle. Se pueden utilizar variables de control del bucle simple o múltiple.

• *Parte de iteración,* que contiene una expresión lógica que hace que el bucle realice las iteraciones de las sentencias, mientras que la expresión sea verdadera.

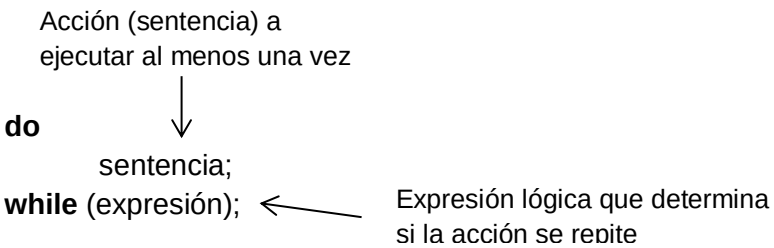
• *Parte de incremento*, que incrementa o decrementa la variable o variables de control del bucle.

• *Sentencias*, acciones o sentencias que se ejecutarán por cada iteración del bucle.

3.3.3 Repetición: El Bucle do-while

La sentencia **do-while** se utiliza para especificar un bucle condicional que se ejecuta al menos una vez. Esta situación se suele dar en algunas circunstancias en las que se ha de tener la seguridad de que una determinada acción se ejecutará una o varias veces, pero al menos una vez.

Sintaxis



La construcción **do** comienza ejecutando sentencia. Se evalúa a continuación expresión. Si expresión es verdadera, entonces se repite la ejecución de sentencia. Este proceso continúa hasta que expresión es falsa.

Cuándo se usa cada uno de los tres bucles. En C++, el bucle for es el más frecuentemente utilizado de los tres. Es relativamente fácil reescribir un bucle do-while como un bucle while, insertando una asignación inicial de la variable condicional. Sin embargo, no todos los bucles while se pueden expresar de modo adecuado como bucles do-while, ya que un bucle do-while se ejecutará siempre al menos una vez y el bucle while puede no ejecutarse. Por esta razón, un bucle while suele preferirse a un bucle do-while, a menos que esté claro que se debe ejecutar una iteración como mínimo.

En la siguiente tabla 3.1 se muestran las sintaxis en C++ de estructuras selectivas y repetitivas.

Tabla 3.1 Sintaxis en C++ de Estructuras Selectivas y Repetitivas

<table>
<tr><td colspan="3">Estructuras selectivas</td></tr>
<tr><td>Sintaxis if:

if(expresion)
{
accion;
}</td><td>Sintaxis if – else:

If(expresion)
{
accion;
}
else
{
accion;
}</td><td>Sintaxis switch:

switch(selector)
{
case etiqueta 1:
sentencias 1;
break;
case etiqueta 2:
sentencias 2;
break;
.
.case n:
sentencias n;
break;
default:
sentencias;
}</td></tr>
<tr><td colspan="3">Estructuras repetitivas</td></tr>
<tr><td>sintaxis for:

for(Inicializacion;CondicionIteracion; incremento)
{
sentencias;
}</td><td>sintaxis while:

while(expresion)
{
sentencias;

}</td><td>sintaxis do – while:

do
{
sentencias;
}while(expresion);</td></tr>
</table>

Nota importante: El uso de las { } solo se requieren si las líneas de sentencias son 2 o más.

Ejercicios Resueltos Capitulo 3

Ejercicios de Estructura Secuencial

3.1 Leer 2 números, calcular e imprimir la suma.

```
// ejercicio 3.1
#include <iostream>
using namespace std;
int main()
{
	int a,b,s;
	cout<<"ingresa a valor de A; /n";
	cin>>a;
	cout<<"ingrese el valor de B. ; "<<endl;
	cin>>b;
	s=a+b;
	cout<<a<<"+"<<b<<"="<<s<<endl;
	system("pause");
	return 0;
}
```

3.2 Lee un nombre y lo imprime, función main de tipo void.

```
//ejercicio 3.2
# include <iostream >
# include <string >
using namespace std ;
void main ()
 {
	string nombre ;
	cout << "Hola , como te llamas ?" << endl;
	cin >> nombre ;
	cout << "Hola , " << nombre << endl;
}
```

3.3 Leer la capacidad de un disco duro en gigas bytes (GB), convertir e imprimirlo en megas bytes (MB), kilo bytes (KB) y bytes (B).

```
// ejercicio 3.3
#include <iostream>
using namespace std;
void main()
{
        float gb,mb,kb,b;
        cout<<“ingrese la capacidad del disco duro en GB \n“;
        cin>>gb;
        mb=gb*1024;
        kb=mb*1024;
        b=kb*1024;
        cout<<“MB= “<<mb<<”\n”;
        cout<<“KB=  “<<kb<<endl;
        cout<<“B = “<<b<<endl;
} // fin main
```

3.4 Calcular el salario neto de un trabajador, en función del número de horas trabajadas y el precio de la hora trabajada, y un descuento del 20% del impuesto.

```
//ejercicio 3.4
#include <iostream>
using namespace std;
void main()
{
        float ht,ph,sb,imp,st;
        cout<<“ingrese las horas trabajadas \n”;
        cin>>ht;
        cout<<“ingrese el pago por hora \n”;
        cin>>ph;
        sb=ht*ph;
        imp=sb*0.20;
        st=sb-imp;
        cout<<“EL SALARIO TOTAL ES:     “<<st<<endl;
}  //fin main
```

3.5 Calcular el área de un triángulo.

```
//ejercicio 3.5
#include<iostream>
using namespace std;
void main()
{
	float a,b,h;
	cout<<"teclee base y altura? \n";
	cin>>b>>h;
	a=b*h/2;
	cout<<"area="<<a<<endl;
}//fin
```

3.6 Un vendedor recibe un sueldo base más un 10% extra por comisión de sus ventas, el vendedor desea saber cuánto dinero obtendrá por concepto de comisiones por las tres ventas que realiza en el mes y el total que recibirá en el mes tomando en cuenta su sueldo base y comisiones.

```
//ejercicio 3.6
#include <iostream>
using namespace std;
void main()
{
	float sb,v1,v2,v3,com,ss;
	cout<<"Ingrese sueldo base \n";
	cin>>sb;
	cout<<"Ingrese las tres ventas \n";
	cin>>v1>>v2>>v3;
	com=(v1+v2+v3)*0.10;
	ss=sb+com;
	cout<<"El sueldo total es:  "<<ss<<endl;
}//Fin main
```

Ejercicios if, if-else, switch:

3.7 Leer tres números diferentes e imprimir el número mayor de los tres.

```
//ejercicio 3.7
#include<iostream>
using namespace std;
void main()
{
    float a,b,c,m;
    cout<<"Ingrese el valor de a,b,c \n: ";
    cin>>a>>b>>c;
    if(a>b && a>c)
          m=a;
    if(b>a && b>c)
          m=b;
    if(c>a && c>b)
          m=c;
cout<<"El numero mayor es: " <<m<<endl;
system("pause");
}
```

3.8 Realizar un programa de calculadora con las operaciones básicas

+.- suma

-.- resta

*****.-multiplicación

/.- división

Teclee dos números y el operador deseado, realizar la operación según el operador.

```
//3.8 Calculadora basica tipo char
#include<iostream>
using namespace std;
void main()
{
          double n1,n2;
          char op;
          cout<<"Teclee valor de n1 y n2 \n";
          cin>>n1>>n2;
          cout<<"Teclee opcion para operacion \n";
          cout<<"(+) suma \n";
          cout<<"(-) resta \n";
          cout<<"(*) multiplicacion \n";
```

```
	cout<<"(/) division \n";
	cin>>op;
switch (op)
{
	case '+':
		cout<<n1<<"+"<<n2<<"="<<n1+n2<<endl;
		break;
	case '-':
		cout<<n1<<"-"<<n2<<"="<<n1-n2<<endl;
		break;
	case '*':
		cout<<n1<<"*"<<n2<<"="<<n1*n2<<endl;
		break;
	case '/':
		cout<<n1<<"/"<<n2<<"="<<n1/n2<<endl;
		break;
	default:
			cout<<"ERROR";
	}//fin de switch
system("pause");
}//fin main
```

3.9 Realizar un programa de calculadora con las operaciones básicas
1.- suma
2.- resta
3.- multiplicación
4.- división
Teclee dos números y el número de la opción deseada, realizar la operación según el número de la opción

```
//ejercicio 3.9
#include<iostream>
using namespace std;
void main()
{
float n1,n2;
int op;
cout<<"Ingrese el valor numerico de n1 y n2 ";
cin>>n1>>n2;
cout<<"Ingrese la operacion a realizar \n";
cout<<" [1] -> Suma \n";
cout<<" [2] -> Resta \n";
cout<<" [3] -> Multiplicacion \n";
```

```
cout<<" [4] -> Division \n";
cin>>op;
switch(op)
{
    case 1:
    cout<<n1<<"+"<<n2<<"="<<n1+n2<<endl;
    break;
    case 2
    cout<<n1<<"-"<<n2<<"="<<n1-n2<<endl;
    break;
    case 3:
    cout<<n1<<"*"<<n2<<"="<<n1*n2<<endl;
    break;
    case 4:
    cout<<n1<<"/"<<n2<<"="<<n1/n2<<endl;
    break;
    default:
    cout<<"ERROR"<<endl;
}
system("pause");
}
```

Ejercicio Bucle for

3.10 Leer 20 números e imprimir cuantos son positivos, cuantos negativos y cuantos neutros.

```
//ejercicio 3.10
#include<iostream>
using namespace std;
void main()
{
    int cp=0,cn=0,cneg=0,num;
    for(int i=1;i<=20;i++)
    {
        cout<<"Ingrese el numero: ";
        cin>>num;
        if(num==0)
          cn++;
        else
          if(num>0)
            cp++;
        else
```

```
            cneg++;
    }
    cout<<"El total de numeros positivos es: "<<cp<<endl;
    cout<<"El total de numeros negativos es: "<<cneg<<endl;
    cout<<"El total de numeros neutros es: "<<cn<<endl;
    system("pause");
}
```

Ejercicios Bucle while

3.11 Obtener el promedio de calificaciones de un grupo de *n* alumnos.

```
//ejercicio 3.11
#include<iostream>
using namespace std;
void main()
{
    float cal, sum=0, ca=0, prom;
    cout<<"Ingrese la calificacion, para finalizar teclee un valor fuera del
    rango\n";
    cin>>cal;
    while(cal>=0 && cal<=100)
    {
          sum=sum+cal;
          ca=ca+1;
          cout<<"Ingrese la calificacion \n";
          cin>>cal;
    }
    prom=sum/ca;
    cout<<"El promedio grupal es: "<<prom<<endl;
    system("pause");
}
```

3.12 Obtener el promedio de una serie de números (utilizar -999 para fin)

```
//ejercicio 3.12
#include<iostream>
using namespace std;
void main()
{
    float num, sum=0, c=0, prom;
    cout<<"Ingrese el numero, para salir teclee -999 \n";
```

```
    cin>>num;
    while(num!=-999)
    {
          sum=sum+num;
          c=c+1;
          cout<<"Ingrese el numero \n";
          cin>>num;
    }
    prom=sum/c;
    cout<<"El promedio de los "<<c<<" numeros ingresados es: "<<prom<<endl;
    system("pause");
}
```

Ejercicios Bucle do while

3.13 Suponga que tiene una tienda y desea registrar sus ventas por medio de una computadora. Diseñe un programa que lea por cada cliente:
a).- el monto de la venta,
b).- calcule e imprima el IVA ,
c).-calcule e imprima el total a pagar,
d).- lea la cantidad con que paga el cliente,
e).-calcule e imprime el cambio.
Al final del día deberá imprimir la cantidad de dinero que debe haber en la caja.

```
//ejercicio 3.13
#include<iostream>
using namespace std;
void main()
{
    float mv,iva,caja=0,cli=0,efec,cambio,tp;
    char op;
    do
    {
        cout<<"Teclee el monto de ventas: ";
        cin>>mv;
        iva=mv*0.16;
        tp=mv+iva;
        caja=caja+tp;//se acumula en caja
        cli=cli+1;
        cout<<"Subtotal: $"<<mv<<endl;
        cout<<"IVA: $"<<iva<<endl;
        cout<<"Total a pagar: $"<<tp<<endl;
        cout<<"Efectivo recibido: $ ";
        cin>>efec;
        cambio=efec-tp;
        cout<<"Cambio: $"<<cambio<<endl;
        cout<<"Agregar otro cliente presione s(si). Finalice con cualquier otro caracter\n";
        cin>>op;
    }while(op=='s' || op=='S');
    cout<<"Total en caja: $"<<caja<<endl;
    cout<<"Cantidad de clientes: "<<cli<<endl;
    system("pause");
}
```

Ejercicios Propuestos Para Solucionar por el Alumno Capitulo 3

1) Leer 2 números; si son iguales que los multiplique, si el primero es mayor que el segundo que los reste y si no que los sume.

2) Leer tres números diferentes e imprimir el número mayor de los tres.

3) Determinar la cantidad de dinero que recibirá un trabajador por concepto de las horas extras trabajadas en una empresa, sabiendo que cuando las horas de trabajo exceden de 40, el resto se consideran horas extras y que estas se pagan al doble de una hora normal cuando no exceden de 8; si las horas extras exceden de 8 se pagan las primeras 8 al doble de lo que se pagan las horas normales y el resto al triple.

4) Realizar un programa de calculadora con las operaciones básicas
+.- suma
-.- resta
*****.-multiplicación
/.- división
Teclee dos números y el operador deseado, realizar la operación según el operador.

5) Realizar un programa de calculadora con las operaciones básicas
1.- suma
2.- resta
3.- multiplicación
4.- división
Teclee dos números y el número de la opción deseada, realizar la operación según el número de la opción

6) Calcular el promedio general de un grupo de 15 alumnos.

7) Leer 20 números e imprimir cuantos son positivos, cuantos negativos y cuantos neutros.

8) Calcular e imprimir la tabla de multiplicar de un número cualquiera. Imprimir el multiplicando, el multiplicador y el producto.

9) Calcular e imprimir las tablas de multiplicar del 1 al 10. Imprimir el multiplicando, el multiplicador y el producto.

10) Obtener el promedio de calificaciones de un grupo de *n* alumnos.

11) Obtener el promedio de una serie de números (utilizar -999 para fin)

12) Determinar cuántos hombres y cuantas mujeres se encuentran en un grupo de *n* personas, suponiendo que los datos son extraídos alumno por alumno.

13) El Depto. de Seguridad Publica y Transito del D.F. desea saber, de los n autos que entran a la ciudad de México, cuantos entran con calcomanía de cada color. Conociendo el último dígito de la placa de cada automóvil se puede determinar el color de la calcomanía utilizando la sig. relación:

DÍGITO	COLOR
1 o 2	amarilla
3 o 4	rosa
5 o 6	roja
7 o 8	verde
9 o 0	azul

14) En un supermercado una ama de casa pone en su carrito los artículos que va tomando de los estantes. La señora quiere asegurarse de que el cajero le cobre bien lo que ella ha comprado, por lo que cada vez que toma un articulo anota su precio junto con la cantidad de artículos iguales que ha tomado y determina cuánto dinero gastara en ese artículo; a esto le suma lo que ira gastando en los demás artículos, hasta que decide que ya tomo todo lo que necesitaba. Ayúdale a esta señora a obtener el total de sus compras.

15) Suponga que tiene una tienda y desea registrar sus ventas por medio de una computadora. Diseñe un pseudocódigo que lea por cada cliente:
a).- el monto de la venta,
b).- calcule e imprima el IVA ,
c).-calcule e imprima el total a pagar,
d).- lea la cantidad con que paga el cliente,
e).-calcule e imprime el cambio.

Al final del día deberá imprimir la cantidad de dinero que debe haber en la caja.

4. ARREGLOS, ESTRUCTURAS Y UNIONES

4.1 Arreglo

Un *array* (**arreglo**) es un tipo de dato estructurado que se utiliza para localizar y almacenar elementos de un tipo de dato dado.

• Existen arrays de una dimensión, de dos dimensiones y multidimensionales.

• En C++ los arrays se definen especificando el tipo de dato del elemento, el nombre del array y el tamaño de cada dimensión del array. Para acceder a los elementos del array se deben utilizar sentencias de asignación directas, sentencias de lectura/escritura o bucles.

4.2 Unidimensionales: Conceptos Básicos, Operaciones y Aplicaciones

Un *array* o **arreglo** unidimensional es una secuencia de objetos del mismo tipo. Los objetos se llaman elementos del *array* y se numeran consecutivamente 0, 1, 2, 3... El tipo de elementos almacenados en el *array* puede ser cualquier tipo de dato de C++. Normalmente, el *array* se utiliza para almacenar tipos tales como char, int o float. Cada item del *array* se denomina *elemento*.

Los elementos de un *array* se numeran, como ya se ha comentado, consecutivamente 0, 1, 2, 3... Estos números se denominan *valores índice* o *subíndice* del *array*. El término *subíndice* se utiliza ya que se especifica igual que en matemáticas como una secuencia tal como a_0, a_1, a_2... Estos números localizan la posición del elemento dentro del *array*, proporcionando *acceso directo* al *array*. Si el nombre del *array* es a, entonces a[0] es el nombre del elemento que está en la posición 0, a[1] es el nombre del elemento que está en la posición 1, etc. En general, el elemento *i-ésimo* está en la posición *i-1*. De modo que si el *array* tiene *n* elementos, sus nombres son a[0], a[1],...,a[n-1]. Gráficamente se representa así el *array* a con seis elementos (figura 4.1).

El *array* a tiene 6 elementos: a[0] contiene 25,1, a[1] contiene 34,2, a[2] contiene 5,25, a[3] contiene 7,45, a[4] contiene 6,09 y a[5] contiene 7,54. El diagrama de la figura 4.1 representa realmente una región de la memoria de la computadora ya que un *array* se almacena siempre con sus elementos en una secuencia de posiciones de memoria contigua. Estos arrays se conocen como *unidimensionales* (una sola dimensión) y se caracterizan por tener un solo subíndice. Estos arrays se conocen también por el término *listas*.

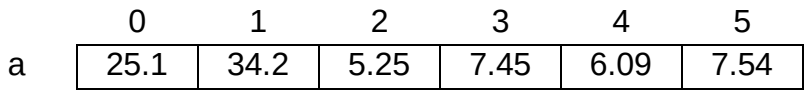


Fig 4.1 Representa Grafica de un *array* a con seis elementos

Podemos concluir que un array tiene:

Tamaño: cuantas cajas va a tener, el número de datos.
Tipo: cuál es el tipo de todos los datos del array.
Nombre: el único nombre bajo el cual vamos a dirigirnos al mismo.

4.2.1 Declaración de un Array Unidimensional

Al igual que con cualquier tipo de variable, se debe declarar un array antes de utilizarlo. Un array se declara de modo similar a otros tipos de datos, excepto que se debe indicar al compilador el *tamaño* o *longitud* del array. Para indicar al compilador el *tamaño* o *longitud* del array se debe hacer seguir al nombre, el tamaño encerrado entre corchetes. La *sintaxis* para declarar un array de una dimensión determinada es:

tipo nombreArray[numeroDeElementos];

Ejemplo:
int notas[10]; //estoy declarando un array llamado notas de 10 elementos numérico entero.

4.2.2 Inicialización de Arrays Unidimensionales

Sintaxis para declarar e inicializar al mismo tiempo:
Tipo nombre [tamaño]={Valor1, Valor2, Valor3, ...}; //Declaración e inicialización

Ejemplo:
int notas[10]={100,80,90,70,60,70,80,90,100,75};

4.2.3 Llenado de Arrays Unidimensionales

Ejemplos de llenados de arrays (arreglos):

```
//ejemplo de cómo declarar y llenar un array al mismo tiempo
  #include<iostream>
  using namespace std;
  void main()
   {
    int a[10]={10,20,30,40,50,60,70,80,90,100};
    for(int i=0;i<=9;i++)
       cout<<a[i]<<endl;
   system("pause");
  }


//ejemplo de llenado de un  array desde el teclado
    #include<iostream>
   using namespace std;
   void main()
    {
        int a[10],i;
        for(i=0;i<=9;i++)
        {
                cout<<"teclee valor del array "<<i<<endl;
                cin>>a[i];
        }
        for(i=0;i<=9;i++)
                cout<<a[i]<<endl;
        system("pause");
     }
```

```
// ejemplo de llenado de un array de forma aleatoria
#include<iostream>
include<stdlib.h> // esta librería se usa en DEV c++
#include<time.h>
using namespace std;
void main()
{
        int a[10];
        srand(time(NULL));
        for(int i=0;i<=9;i++)
        {
          a[i]=rand()%100;
          cout<<a[i]<<endl;
        }
          system("pause");
 }//fin main
```

Ejercicios Resueltos Capitulo 4

Ejercicios Arrays Unidimensionales

4.1 Calcular el promedio de 50 valores almacenados en un array. Determinar además cuantos son mayores que el promedio, imprimir el promedio, el número de datos mayores que el promedio y una lista de valores mayores que el promedio.

```
//4.1
#include<iostream>
#include<time.h>
using namespace std;
void main()
{
	srand(time(NULL));
	float a[50],prom,sum=0;
	int c=0;
	for(int i=0;i<=49;i++)
	{
	  a[i]=rand()%100;
	  sum=sum+a[i];
	}
	prom=sum/50;
	cout<<" el promedio= "<<prom<<endl<<endl;
	for(int i=0;i<=49;i++)
	  if (a[i]>prom)
		{
		  c=c+1;
		  cout<< a[i]<<"\t es > que \t"<<prom<<endl;
		}
	cout<<"total de numeros > que el promedio= "<<c<<endl;
system("pause");
}//fin main
```

4.2 Llenar dos arrays A y B de 10 elementos cada uno, sumar el elemento uno del array A con el elemento uno del array B y así sucesivamente hasta 10, almacenar el resultado en un array C, e imprimir el array resultante.

```
//4.2
#include<iostream>
#include<time.h>
using namespace std;
void main()
 {
   srand(time(NULL));
   int a[10],b[10],c[10];
   for(int i=0;i<=9;i++)
   {
            a[i]=rand()%100;
            b[i]=rand()%100;
            c[i]=a[i]+b[i];
            cout<<a[i]<<"+"<<b[i]<<"="<<c[i]<<endl;
   }
  system("pause");
}//fin main
```

4.3 Llenar un array de 10 elementos, imprimir la posición y el valor del elemento mayor almacenado en el array. Suponga que todos los elementos del array son diferentes.

```
//4.3
#include<iostream>
#include<time.h>
using namespace std;
void main()
{
 srand(time(NULL));
 int a[10],i,pos=0,mayor;
for(i=0;i<=9;i++)
{
          a[i]=rand()%100;
          cout<<a[i]<<endl;
}
mayor=a[0];
for(i=0;i<=9;i++)
 {
```

```
        if(a[i]>=mayor)
        {
          mayor=a[i];
          pos=i;
        }
 }//fin for
cout<<"mayor="<<mayor<<"\t pos="<<pos<<endl;
system("pause");
}//fin main
```

4.4 Almacenar 10 números en un array, elevar al cuadrado cada valor almacenado en el array, almacenar el resultado en otro array. Imprimir el array original y el array resultante.

```
//4.4 con rand()
#include<iostream>
#include<time.h>
using namespace std;
void main()
{
 srand(time(NULL));
 int a[10],b[10],i;
 for(i=0;i<=9;i++)
 {
   a[i]=rand()%20;
   b[i]=a[i]*a[i];
   cout<<a[i]<<"\t^2\t"<<b[i]<<endl;
 }//fin for
 system("pause");
}//fin main
```

4.5 Almacenar 10 números en un array, imprimir cuantos son ceros, cuántos son negativos, cuantos positivos. Imprimir además la suma de los negativos y la suma de los positivos.

```
//4.5
#include<iostream>
using namespace std;
void main()
{
 int a[10],i,cp=0,cneg=0,cn=0,sp=0,sneg=0;
 for(i=0;i<=9;i++)
 {
   cout<<"a["<<i<<"]=";
   cin>>a[i];
   if (a[i]==0)
    cn=cn+1;
   else
     if(a[i]>0)
            {
              cp=cp+1;
              sp=sp+a[i];
            }
            else
            {
                    cneg=cneg+1;
                    sneg=sneg+a[i];
            }
 }//fin for
 cout<<"neutros="<<cn<<endl;
 cout<<"positivos="<<cp<<" "<<"suma(+)="<<sp<<endl;
 cout<<"negativos="<<cneg<<" "<<"suma(-)="<<sneg<<endl;
system("pause");
}//fin main
```

4.6 Se tienen almacenados en la memoria dos arrays M y N de cien elementos cada uno. Hacer un programa que escriba la palabra “Iguales” si ambos arrays son iguales y “Diferentes” si no lo son. Serán iguales cuando en la misma posición de ambos arrays se tenga el mismo valor para todos los elementos.

```
//4.6 Array M y N de 10 elementos
//Escribir "iguales" si son iguales o "diferentes" si no lo son
#include<iostream>
using namespace std;
void main()
{
	int M[10],N[10],i,c=0;
	for(i=0;i<=9;i++)
	{
	    cout<<"Ingrese el valor "<<i<<" para ambos arrays (M y N)"<<":\n";
	    cin>>M[i]>>N[i];
	}
	cout<<"\n";
	cout<<"----------"<<endl;
	cout<<"M\tN\n"<<endl;
	for(i=0;i<=9;i++)
	    cout<<M[i]<<"\t"<<N[i]<<endl;
	for(i=0;i<=9;i++)
	{
	    if(M[i]==N[i])
	    c++;
	}
	if(c==10)
	    cout<<"\n-LOS ARRAYS SON IGUALES\n";
	else
	    cout<<"\n-LOS ARRAYS SON DIFERENTES\n";
	system("pause");
}
```

4.7 Se tiene el array A con 10 elementos almacenados. Diseñe un algoritmo que escriba “SI” si el array esta ordenado ascendentemente o “NO” si el array no está ordenado

```
//4.7 Array 'a' de 10 elementos
//Escribir "si" esta en orden ascendente o "no" si no esta
#include<iostream>
using namespace std;
void main()
{
	int a[10],i,c=0;
	for(i=0;i<=9;i++)
	{
	    cout<<"Ingrese el valor "<<i<<":\n";
	    cin>>a[i];
	}
	cout<<"\n";
	cout<<"----------"<<endl;
	cout<<"a\n"<<endl;
	for(i=0;i<=9;i++)
	    cout<<a[i]<<endl;
	for(i=0;i<=8;i++)
	    {
	        if(a[i]<=a[i+1])
	        c++;
	    }
	if(c==9)
	    cout<<"\nEl array SI esta en orden ascendente\n";
	else
	    cout<<"\nEl array NO esta en orden ascendente\n";
	system("pause");
}
```

4.8 Diseñe un algoritmo que lea un número cualquiera y lo busque en el array X, el cual tiene almacenados 80 elementos. Escribir la posición donde se encuentra almacenado el número en el array o el mensaje “NO” si no lo encuentra.Búsqueda secuencial.

```
//4.8 Buscar un # en un array x de 20 elementos
#include<iostream>
#include<stdlib.h>
#include<time.h>
using namespace std;
void main()
{
	cout<<"ARRAY\n";
	cout<<"-------\n";
	srand(time(NULL));
	float a[20],num;
	int i,pos=0;
	bool band=false;
	for(int i=0;i<=19;i++)
	{
	    a[i]=rand()%100;
	    cout<<a[i]<<endl;
	}
	cout<<"\nIngrese el numero a buscar dentro del array:\n";
	cin>>num;
	for(i=0;i<=19;i++)
	if(a[i]==num)
	{
			band=true;
			pos=i;
	}
	if(band==true)// o(band==1)
	  cout<<"\nEl numero "<<num<<"fue encontrado en la posicion:
	  "<<pos<<"\n"<<endl;
	else
	  cout<<"\nEl numero "<<num<<" NO fue encontrado en el array\ n"<<endl;
	system("pause");
}
```

4.9 llenar un array, utilizar el método de la burbuja para ordenarlo.

```
//4.9 ordenamiento de un array
 #include<iostream>
 #include<time.h>
 using namespace std;
 void main()
 {
        srand(time(NULL));
        int a[10],i,j,aux;
        for(i=0;i<=9;i++)
        {
                a[i]=rand()%100;
                cout<<a[i]<<endl;
        }
        //aqui inicia el ordenamiento
        for(i=0;i<=9;i++)
          for(j=0;j<=8;j++)
                if (a[j]>a[j+1])
                        {
                                aux=a[j];
                                a[j]=a[j+1];
                                a[j+1]=aux;
                        }
        //aqui se imprime el array ordenado
        cout<<"\n Array A ordenado \n\n";
        for(i=0;i<=9;i++)
                cout<<a[i]<<endl;
        system("pause");
 }
```

4.3 Arrays Multidimensionales

Los arrays vistos anteriormente se conocen como arrays *unidimensionales* (una sola dimensión) y se caracterizan por tener un solo subíndice. Estos arrays se conocen también por el término *listas*. Los *arrays multidimensionales* son aquellos que tienen más de una dimensión y, en consecuencia, más de un índice. Los arrays más usuales son los de dos dimensiones, conocidos también por el nombre de *tablas* o *matrices*. Sin embargo, es posible crear arrays de tantas dimensiones como requieran sus aplicaciones, esto es, tres, cuatro o más dimensiones.
Un array de dos dimensiones equivale a una tabla con múltiples filas y múltiples columnas (figura 4.2).

	0	1	2	3
0	0,0	0,1	0,2	0,3
1	1,0	1,1	1,2	1,3
2	2,0	2,1	2,2	2,3
3	3,0	3,1	3,2	3,3

Figura 4.2 Estructura de Dos Dimensiones

4.3.1 Declaración de un Array Multidimensional

La sintaxis para la declaración de un array de dos dimensiones es:

<tipo de datoElemento> <nombre array> [<NúmeroDeFilas<] [<NúmeroDeColumnas>]

Algunos ejemplos de declaración de tablas son:

```
char Pantalla[25][80];
int puestos[6][8];
int equipos[4][30];
int matriz[4][2];
```

4.3.2 Inicialización de Arrays Multidimensionales

Los arrays multidimensionales se pueden inicializar, al igual que los de una dimensión, cuando se declaran. La inicialización consta de una lista de constantes separadas por comas y encerradas entre llaves, como en Ejemplo:

```
int tabla1[2][3] = {51, 52, 53, 54, 55, 56};
```

o bien en los formatos:

```
int tabla[2][3]= {{51, 52, 53},
                  {54, 55, 56} };

int tabla[2][3]= {{51, 52, 53}, {54, 55, 56}};
```

4.3.3 Llenado de Arrays Multidimensionales:

```
/* llenar por teclado e imprime una matriz de 3 filas por 3 columnas */
#include<iostream>
using namespace std;
void main()
{
 int a[3][3];
 for(int i=0;i<=2;i++)
    for(int j=0;j<=2;j++){
       cout<<"ingrese valor["<<i<<","<<j<<"]=";
       cin>>a[i][j];
    }
  cout<<"\n\n\n";
  for(int i=0;i<=2;i++){
       for(int j=0;j<=2;j++)
          cout<<a[i][j]<<"\t";
  cout<<endl;
   }
 system("pause");
}//fin main
```

```
/* llenar matriz desde la declaracion*/
#include<iostream>
using namespace std;
void main()
 {
   int a[3][3]={{10,20,30}, {40,50,60}, {70,80,90}};
   for(int i=0;i<=2;i++){
     for(int j=0;j<=2;j++)
         cout<<a[i][j]<<"\t";
  cout<<endl;
 }
 system("pause");
}//fin main
```

```
// llenar aleatoriamente e imprime una matriz
#include<iostream>
#include<time.h>
using namespace std;
 void main()
 {
 srand(time(NULL));
 int a[3][3];
  for(int i=0;i<=2;i++)
  {
      for(int j=0;j<=2;j++)
        {
              a[i][j]=rand()%100;//llenado aleatorio
              cout<<a[i][j]<<"\t";
        }
   cout<<endl;
  }
 system("pause");
}//fin main
```

Ejercicios Arrays Multidimensionales

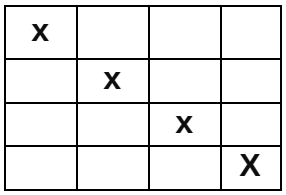
4.10 Crear una matriz de 4 * 4 filas Imprimir el mayor elemento y la fila y columna donde se almacena.

```
//4.10
#include<iostream>
#include<time.h>
using namespace std;
void main()
{
  srand(time(NULL));
  int a[4][4],mayor,fila,col;
  for(int i=0;i<=3;i++)
  {
            for(int j=0;j<=3;j++)
             {
               a[i][j]=rand()%100;//llenado aleatorio
               cout<<a[i][j]<<"\t";
             }
            cout<<endl;
  }
 mayor=a[0][0];
  for(int i=0;i<=3;i++)
    for(int j=0;j<=3;j++)
       if(a[i][j]>mayor)
                {
                  mayor=a[i][j];
                        fila=i;
                        col=j;
                 }
  cout<<"mayor= "<<mayor<<"\t";
  cout<<"fila="<<fila<<"\t"<<"columna="<<col<<endl;
  system("pause");
 }//fin main
```

4.11 Crear y cargar una matriz de 4 filas por 4 columnas. Imprimir la primera fila. Imprimir la última fila.

```
//4.11 imprimir el primer y ultimo renglon
#include<iostream>
#include<time.h>
using namespace std;
void main()
 {
  srand(time(NULL));
  int a[4][4];
  for(int i=0;i<=3;i++)
  {
            for(int j=0;j<=3;j++)
             {
                  a[i][j]=rand()%100;//llenado aleatorio
                cout<<a[i][j]<<"\t";
             }
            cout<<endl;
  }
  cout<<"\n\n";
  for(int j=0;j<=3;j++)
          cout<<a[0][j]<<"\t";//imprimr la primera fila
 cout<<"\n\n\n";
 for(int j=0;j<=3;j++)
          cout<<a[3][j]<<"\t";//imprime la última fila
 cout<<endl<<endl;
 system("pause");
 }//fin main
```

4.12 Crear y cargar una matriz de 4 filas por 4 columnas. Imprimir la diagonal principal.




```
//4.12 imprimir la diagonal principal
#include<iostream>
 using namespace std;
 void main()
 {
  int a[4][4];
  for(int i=0;i<=3;i++)
  {
            for(int j=0;j<=3;j++)
             {
                  a[i][j]=rand()%100;//llenado aleatorio
                cout<<a[i][j]<<"\t";
             }
            cout<<endl;
  }
 cout<<"\n\n";
 for(int i=0;i<=3;i++)
  {
            for(int j=0;j<=3;j++)
                      if(i==j)
                       cout<<a[i][j]<<"\t";
                      else
                       cout<<"\t";
           cout<<endl;
  }
 system("pause");
 }//fin main
```

Ejercicios Propuestos Para Solucionar por el Alumno Capitulo 4

1).Escribir un programa que llene un arreglo con los números enteros comprendidos entre 4 y 14.

2) Escribir un programa que llene un arreglo con los números pares comprendidos entre 1 y 100.

3) Desarrolle un programa en c++ que permita ingresar varios números. Almacene los números pares en un arreglo llamado numPar[] y los números impares en un arreglo llamado numImpar[]. Finalmente, deberá mostrar los numero pares ingresados y después los números impares.

4) Encontrar el número mayor de una lista.

5) Programa que ordena un array de números de forma ascendente, utilizar el método de la burbuja.

6). Diseñar un programa que determine e imprima el dia de la semana de mayor venta registradas en un array.

5. MÓDULOS

C++ se puede utilizar como *lenguaje de programación estructurada,* también conocida como *programación modular*. Por esta razón, para escribir un programa se divide éste en varios módulos, en lugar de uno solo largo. El programa se divide en muchos módulos (rutinas pequeñas denominadas *funciones*), que producen muchos beneficios: aislar mejor los problemas, escribir programas correctos más rápido y producir programas que son más fáciles de mantener.

¿Funciones, métodos o procedimientos?
En el mundo de la programación, muchos acostumbramos hablar indistintamente de estos tres términos sin embargo poseen deferencias fundamentales.

Métodos:
Los métodos y las funciones son funcionalmente idénticos, pero su diferencia radica en el contexto en el que existen. Un método también puede recibir valores, efectuar operaciones con estos y retornar valores, sin embargo en método está asociado a un objeto, básicamente un método es una función que pertenece a un objeto o clase, mientras que una función existe por sí sola, sin necesidad de un objeto para ser usada.

Procedimientos:
Los procedimientos son básicamente lo un conjunto de instrucciones que se ejecutan sin retornar ningún valor, hay quienes dicen que un procedimiento no recibe valores o argumentos, sin embargo en la definición no hay nada que se lo impida. En el contexto de C++ un procedimiento es básicamente una función void que no nos obliga a utilizar una sentencia return.

Funciones:
Las funciones son un conjunto de procedimiento encapsulados en un bloque, usualmente reciben parámetros, cuyos valores utilizan para efectuar operaciones y adicionalmente retornan un valor. Esta definición proviene de la definición de función matemática la cual posee un dominio y un rango, es decir un conjunto de valores que puede tomar y un conjunto de valores que puede retornar luego de cualquier operación.

5.1 Declaración y Uso de Módulos

5.1.1 Prototipos de las Funciones

C++ requiere que una función se declare o defina antes de su uso. La declaración de una función se denomina prototipo. Los prototipos de una función contienen la misma cabecera de la función, con la diferencia de que los prototipos terminan con un punto y coma. Específicamente un prototipo consta de los siguientes elementos: nombre de la función, una lista de argumentos encerrados entre paréntesis y un punto y coma. La inclusión del nombre de los parámetros es opcional.

Sintaxis:
valor_de_retorno nombre_funcion (lista_tipos_de_parametros);

- El prototipo declara el valor de retorno de la función, nombre de la funcion el número, tipos y orden de los parámetros (valores) que la función espera recibir.
- Normalmente, colocaremos los prototipos de funciones al principio del programa antes de la función main.

5.1.2 Diseño o Definición de Función

El diseño o definición de función, contiene las instrucciones con las que realizará su trabajo la función.

Sintaxis:

```
valor_de_retorno  nombre_funcion (lista_de_parametros)
 {
        proceso;
 }
```

5.1.3 Llamada a la Función

La llamada a una función, se realiza escribiendo el nombre de la función, seguido por la lista de parámetros entre paréntesis.
Una función invocada puede regresar el control a su invocador de tres formas:

- Si la función no regresa valor, el control se devuelve al encontrar la llave final de la misma.
- Ejecutando una instrucción *return*;
- Si la función regresa valor, se debe utilizar la siguiente instrucción para regresar el valor a su invocador: *return* valor;

Sintaxis:
Nombre_de_funcion(parametros);

5.2 Paso de Parámetros o Argumentos

5.2.1 Parámetros de una Función

C++ proporciona dos métodos para pasar variables (*parámetros*) entre funciones. Una función puede utilizar *parámetros por valor* y *parámetros por referencia*, o *puede no tener parámetros*. Esta sección examina el mecanismo que C++ utiliza para pasar parámetros a funciones y cómo optimizar el paso de parámetros, dependiendo del tipo de dato que se utiliza. Suponiendo que se tenga la declaración de una función **círculo** con tres argumentos.

void circulo(int x, int y, int diametro);
Cuando se llama a circulo se deben pasar tres parámetros a esta función.

5.2.2 Paso de Parámetros por Valor

Paso por valor (también llamado *paso por copia*) significa que cuando C++ compila la función y el código que llama a la función, la función recibe una copia de los valores de los parámetros. Si se cambia el valor de un parámetro variable local, el cambio sólo afecta a la función y no tiene efecto fuera de la función.

5.2.3 Paso de Parámetros por Referencia

Cuando una función debe modificar el valor del parámetro pasado y devolver este valor modificado a la función llamadora, se ha de utilizar el método de paso de parámetro por *referencia* o *dirección*.
En este método el compilador pasa la dirección de memoria del valor del parámetro a la función.
Cuando se modifica el valor del parámetro (la variable local), este valor queda almacenado en la misma dirección de memoria, por lo que al retornar a la función llamadora la dirección de la memoria donde se almacenó el parámetro contendrá el valor modificado. Para declarar una variable parámetro como paso por referencia, el símbolo & debe preceder al nombre de la variable.

5.2.4 Diferencia entre el Paso de Parámetros por Valor y por Referencia

Las reglas que se han de seguir cuando se utilizan parámetros valor y referencia son las siguientes:
• Los parámetros valor (declarados sin &) reciben copias de los valores de los argumentos que se les pasan;
• la asignación a parámetros valor de una función nunca cambian el valor del argumento original pasado a los parámetros;
• los parámetros referencia (declarados con &) reciben la dirección de los argumentos pasados;
• en una función, las asignaciones a parámetros referencia cambian los valores de los argumentos originales.
Por ejemplo, la escritura de una función intercambio() para intercambiar los contenidos de dos variables, requiere que los datos puedan ser modificados.
Llamada por valor Llamada por referencia

```
int primero, segundo;                          int primero, segundo;

intercambio1   (int primero,int segundo)       intercambio2 (int& primero,int& segundo)

{                                              {
...                                                  ...
}                                              }
```

Sólo en el caso de intercambio2 los valores de uno y dos se cambiarán.
Veamos una aplicación completa de ambas funciones:

```
#include <iostream>
using namespace std;
void intercambio1(int, int);
void intercambio2(int&, int&);
void main()
{
      int a, b;
      a = 10; b = 20;
      intercambio1(a, b);
      cout << a << b << endl;
      intercambio2(a, b);
      cout << a << b << endl;
}
```

```
void intercambio1(int primero, int segundo)
{
	int aux;
	aux = primero;
	primero = segundo;
	segundo = aux;
}
void intercambio2(int& primero, int& segundo)
{
	int aux;
	aux = primero;
	primero = segundo;
	segundo = aux;
}
```

La ejecución del programa producirá:

10 20
20 10

5.3 Implementación

5.3.1 Ejemplos de Implementación de Funciones

```
// funcion que suma dos números enteros, con valor de retorno y envía parámetros
#include<iostream>
using namespace std;
int suma(int,int);  //declaración de la función
void main()
{
	int a,b;
	cout<<“ingrese el valor de a, b : “;
	cin>>a>>b;
	cout<<“la suma   es: “<<suma(a,b)<<endl;
	system(“pause”);
}  // fin main

// definición de la función el valor de a lo recibe x y el valor de b lo recibe y.
int suma(int  x, int  y)
{
	return  x+y;
}
```

```
//función que calcula el área de un rectángulo, con retorno con parámetros
#include<iostream>
using namespace std;
float area(float,float);  //declaracion de la funcion
void main()
{
        float b,h;
        cout<<“ingrese el valor de la base y la altura: “;
        cin>>b>>h;
        cout<<“El area es: “<<area(b,h)<<endl;
system(“pause”);
}  // fin main

// definicion de la funcion
float area(float  b, float  h)
{
        return  b*h;
}

//función que calcula el área de un rectángulo, sin retorno con parámetros
#include<iostream>
using namespace std;
void area(float,float);  //declaracion de la funcion
void main()
{
        float b,h;
        cout<<“ingrese el valor de la base y la altura: “;
        cin>>b>>h;
        area(b,h);
}  // fin main

// definicion de la funcion
void area(float  b, float  h)
{
        cout<<“la area  es: “<<b*h<<endl;
}
```

```
//función que calcula el área de un rectángulo, sin retorno sin parámetros
#include<iostream>
using namespace std;
void area( );  //declaración de la función
void main()
{
        area( );
}  // fin main

// definición de la función
void area()
{
        float b,h;
        cout<<“ingrese el valor de la base y la altura: “;
        cin>>b>>h;
        cout<<“El area es: “<<b*h<<endl;
}
```

Ejercicios Resueltos Capitulo 5

5.1 Que muestre un menú con las opciones sumar, restar, multiplicar y dividir, el programa solicitará una opción y realizará la tarea elegida, se debe usar un switch para mostrar el menú, pedir los datos en el main y después usar funciones para realizar los cálculos.

```
//Ejercicio 5.1 calculadora con funciones
#include <iostream>
using namespace std;
void suma (float,float);//declaración de funciones
void resta(float,float);
void division(float,float);
void multi(float,float);
void main() {
    float a,b;
    char op;
    cout<<"Ingrese valor a\n";
    cin>>a;
    cout<<"Ingrese valor b\n";
    cin>>b;
    cout<<"Ingrese el símbolo según la operación deseada\n”;
    cout<<”+ suma\n”;
    cout<<”- resta\n”;
    cout<<”* multiplicación\n”;
    cout<<”/ división\n";
    cin>>op;
    switch (op)//switch para las opciones de calculadora
    {
        case'+':
        suma(a,b);//llamada de funciones
        break;
        case'-':
        resta(a,b);
        break;
        case'*':
        multi(a,b);
        break;
        case'/':
        division(a,b);
        break;
        default:
```

```
            cout<<"Error \n";
    }// fin switch
system(“pause”);
}//fin del main

//diseño de función suma
void suma(float x,float y){
    float c;
    c=x+y;
    cout<<"La suma de "<<x<<" mas "<<y<<" es igual a: "<<c<<endl;
}
//diseño de función resta
void resta(float x,float y){
    float c;
    c=x-y;
    cout<<"La resta de "<<x<<" menos "<<y<<" es igual a: "<<c<<endl;
}
//diseño de función multiplicación
void multi(float x,float y){
    float c;
    c=x*y;
    cout<<"La multiplicación de "<<x<<" por "<<y<<" es igual a: "<<c<<endl;
}
//diseño de función división
void division(float x,float y){
    float c;
    c=x/y;
    cout<<"La división de "<<x<<" entre "<<y<<" es igual a: "<<c<<endl;
}
```

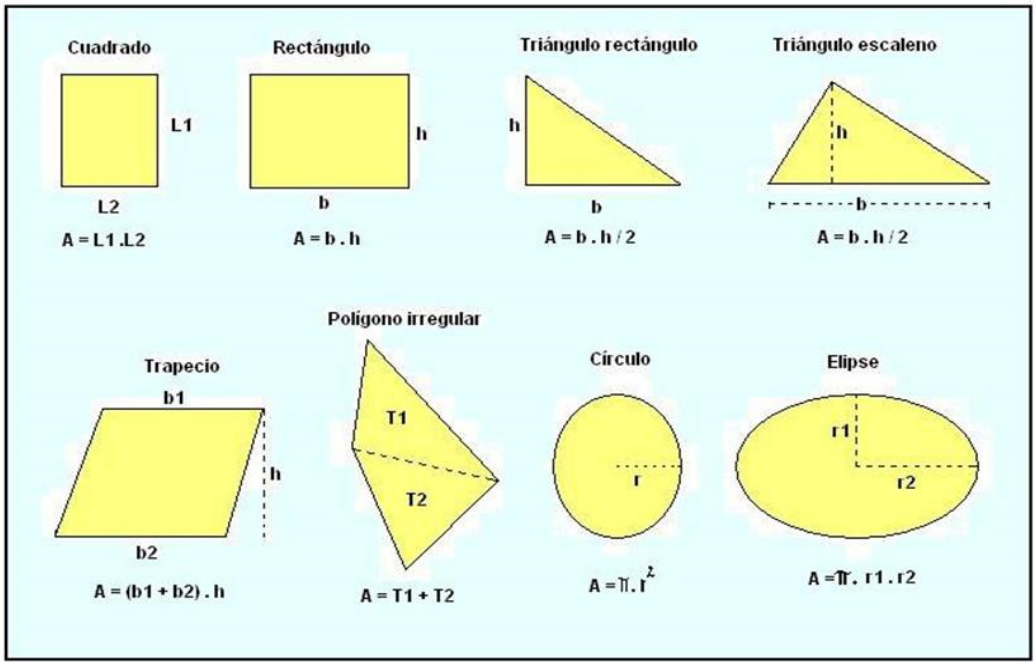
5.2 Que pida por pantalla una temperatura, muestre un menú para convertirlos a Centígrados, Fahrenheit o Kelvin y muestre el equivalente por pantalla, utilizar funciones.

```
//Ejercicio 5.2 Conversión de distintos grados con funciones
#include <iostream>
using namespace std;
void f_c();//llamada a funciones sin parametros ni retorno.
void c_f();
void f_k();
void k_f();
void c_k();
```

```
void k_c();
void main() {
    int op;
    cout<<"Elija el numero dependiendo de que conversión desea utilizar \n";
    cout<<"(1)de fahrenheit a celsius\n";
    cout<<"(2)de celsius a fahrenheit\n";
    cout<<"(3)de fahrenheit a kelvin\n";
    cout<<"(4)de kelvin a fahrenheit \n";
    cout<<"(5)de celsius a kelvin\n";
    cout<<"(6)de kelvin a celsius\n";
    cin>>op;//leer valor
    switch (op){//switch de las diferentes opciones
        case 1:
          f_c();//lamada de funciones
          break;
        case 2:
          c_f();
          break;
        case 3:
          f_k();
          break;
        case 4:
          k_f();
          break;
        case 5:
          c_k();
          break;
        case 6:
          k_c();
          break;
        default:
          cout<<"Error\n";
        }//fin switch
}//fin del main
//inicio de funciones
void f_c(){
      float f,c;
      cout<<"Ingrese la temperatura en fahrenheit\n";
      cin>>f;
      c=(f-32)/1.8;
      cout<<"Los grados fahrenheit en celsius son: "<<c<<" grados"<<endl;
}
```

```
void c_f(){
    float f,c;
    cout<<"Ingrese la temperatura en celsius\n";
    cin>>c;
    f=(c*1.8)+32;
    cout<<"Los grados celsius a Fahrenheit son "<<f<<" grados\n";
}
void f_k(){
      float f,k;
      cout<<"Ingrese la temperatura en fahrenheit\n";
      cin>>f;
      k=((f-32)/1.8)+273.15;
      cout<<"Los grados fahrenheit a celsius son "<<k<<" grados\n";
}
void k_f(){
      float k,f;
      cout<<"Ingrese la temperatura en kelvin\n";
      cin>>k;
      f=(((k-273.15)*1.8)+32);
      cout<<"Los grados kelvin a fahrenheit son "<<f<<" grados\n";
}
void c_k(){
      float c,k;
      cout<<"Ingrese la temperatura en celsius\n";
      cin>>c;
      k=c+273.15;
      cout<<"Los grados celsius a kelvin son "<<k<<" grados\n";
}
void k_c(){
    float c,k;
    cout<<"Ingrese la temperatura en kelvin\n";
    cin>>k;
    c=k-273.15;
    cout<<"Los grados kelvin a celsius son "<<c<<" grados\n";
}
```

5.3 Diseñe el siguiente programa: mediante un menú presente las opciones de cálculo de área para cada figura y mediante una función calcule el área:




```
//Ejercicio 5.3 Formulas de area de diversas figuras
#include <iostream>
using namespace std;
void cuadrado();//declaración de funciones
void rectangulo();
void trec();
void tesc();
void trapecio();
void poligono();
void circulo();
void elipse();
void main() {
    int op;//declarar variables
    cout<<"Elija el numero dependiendo de que figura sacará el área \n";
    cout<<"(1)cuadrado\n";
    cout<<"(2)rectángulo\n";
    cout<<"(3)triángulo rectángulo\n";
    cout<<"(4)triángulo escaleno \n";
    cout<<"(5)trapecio\n";
    cout<<"(6)polígono irregular\n";
    cout<<"(7)círculo\n";
```

```
    cout<<"(8)elipse\n";
    cin>>op;//lectura de valor
    switch (op){//switch para escoger opción
    case 1:
        cuadrado();//Llamada de función
        break;
    case 2:
        rectangulo();
        break;
    case 3:
        trec();
        break;
    case 4:
        tesc();
        break;
    case 5:
        trapecio();
    break;
    case 6:
        poligono();
        break;
    case 7:
        circulo();
        break;
    case 8:
        elipse();
        break;
    default:
        cout<<"Error\n";
        break;
    }
}//fin del main
//inicio de funciones.
void cuadrado(){
    float L,a;
    cout<<"Ingrese cuanto mide el lado\n";
    cin>>L;
    a=L*L;
    cout<<"El área de su cuadrado es: "<<a<<endl;
}
```

```
void rectangulo(){
    float b,h,a;
    cout<<"Ingrese cuando mide la base\n";
    cin>>b;
    cout<<"Ahora, ingrese la altura\n";
    cin>>h;
    a=b*h;
    cout<<"El área de su rectángulo es "<<a<<endl;
}
void trec(){
    float b,h,a;
    cout<<"Ingrese cuando mide la base\n";
    cin>>b;
    cout<<"Ahora, ingrese la altura\n";
    cin>>h;
    a=b*h/2;
    cout<<"El área de su triángulo rectángulo es: "<<a<<endl;
}
void tesc(){
    float b,h,a;
    cout<<"Ingrese cuando mide la base\n";
    cin>>b;
    cout<<"Ahora, ingrese la altura\n";
    cin>>h;
    a=b*h/2;
    cout<<"El área de su triángulo escaleno es: "<<a<<endl;
}
void trapecio(){
    float b1,b2,h,a;
    cout<<"Ingrese cuanto mide la base 1\n";
    cin>>b1;
    cout<<"Ingrese cuanto mide la base 2\n";
    cin>>b2;
    cout<<"Ingrese la altura del trapecio\n";
    cin>>h;
    a=(b1+b2)*h;
    cout<<"El área de su trapecio es "<<a<<endl;
}
void poligono(){
    float b1,h1,b2,h2,a;
    cout<<"Ingrese la base del triángulo 1\n";
    cin>>b1;
```

```
    cout<<"Ingrese la altura del triángulo 1\n";
    cin>>h1;
    cout<<"Ingrese la base del triángulo 2\n";
    cin>>b2;
    cout<<"Ingrese la altura del triángulo 2\n";
    cin>>h2;
    a=(b1*h1/2)+(b2*h2/2);
    cout<<"El área del poligono irregular es "<<a<<endl;
}
void circulo(){
    float r,a;
    const float pi=3.1416;
    cout<<"Ingrese el radio del círculo\n";
    cin>>r;
    a=r*r*pi;
    cout<<"El área de su círculo es "<<a<<endl;
}
void elipse(){
    float r1,r2,a;
    const float pi=3.1416;
    cout<<"Ingrese el radio 1 del elipse, es el menor, tambien conocido como b\n";
    cin>>r1;
    cout<<"Ahora, ingrese el radio 2 del elispse, el mayor, conocido como a\n";
    cin>>r2;
    a=pi*r1*r2;
    cout<<"El área de su elipse es "<<a<<endl;
}
      //fin de funciones
```

5.4 Diseñe un algoritmo que llene una matriz de 10 * 10 y determine:

A) El número mayor almacenado en la matriz
B) El número mayor almacenado en cada renglón
C) La columna que tuvo la máxima suma
D) El renglón que tuvo la máxima suma

Diseña una función para cada inciso.

```
//5.4 una matriz como parámetro en una función
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
int IA(int a[10][10]);//declaracion de funciones
void IB(int a[10][10]);
int IC(int a[10][10]);
int ID(int a[10][10]);
void main() {
    srand(time(NULL));//llenado aleatorio
    int a[10][10];
    for (int i=0;i<=9;i++){
    for (int j=0;j<=9;j++){
          a[i][j]=rand()%100+1;
          cout<< a[i][j]<<"\t";
          }
    cout<<endl;
    }
    cout<<"El numero mayor almacenado en la matriz es "<<IA(a)<<endl;
    cout<<"Los numeros mayores para cada renglon son:\n";
    IB(a);//llamada de funciones
    cout<<"La columna que tuvo la máxima suma es:"<<IC(a)<<endl;
    cout<<"El renglon que tuvo la maxima suma es:\n"<<ID(a)<<endl;
}//fin del main
//inicio de funciones
int IA(int a[10][10]){//Número mayor
    int mayor=0;
    for(int i=0;i<=9;i++)
          for(int j=0;j<=9;j++)
                if(a[i][j]>mayor)
                      mayor=a[i][j];
    return mayor;
}
```

```
void IB(int a[10][10]){//Numero mayor de cada renglon
    int mayor=0;
    for(int i=0;i<=9;i++){
          for(int j=0;j<=9;j++)
                  if(a[i][j]>mayor)
                          mayor=a[i][j];
      cout<<"Mayor renglon"<<i<<":"<< mayor<<endl;
      mayor=0;
   }
}

int IC(int a[10][10]){//columna con máxima suma
int suma=0,mayor=0,col,val;
for(int i=0;i<=9;i++){
    for(int j=0;j<=9;j++){
          suma=suma+a[j][i];
          }
    if (suma>mayor){
          mayor=suma;
          col=i;
    }
    suma=0;
 }
return col;
}

int ID(int a[10][10]){//renglón con maxima suma
    int suma=0,mayor=0,ren;
    for(int i=0;i<=9;i++){
        for(int j=0;j<=9;j++){
          suma=suma+a[i][j];
          }
    if (suma>mayor){
          mayor=suma;
          ren=i;
    }
    suma=0;
}
return ren;//regresa el valor
}
//fin funciones
```

5.5 Crear una función que reciba un número real y devuelva un número entero con el valor: -1 si el número es negativo, 1 si el número es positivo o 0 si es neutro. Función con parámetros y con valor de retorno.

```
//Ejercicio 5.5 regresar un numero, dependiendo de otro.
#include <iostream>
using namespace std;
int sig(int);//declaramos la función
void main() {
    int num;//Declaramos variables
    cout<<"Ingrese un número.\n Se le regresará un -1 si este es negativo,\n 1 si es
    positivo,\n o 0 si es neutro\n";
    cin>>num;
    cout<<"El número regresado es: "<<sig(num)<<endl;
}//fin del main
int sig(int a){//Se hace la verificación en una función aparte
    int c;
    if (a==0)
        c=0;//se regresa un numero, dependiendo si es positivo, negativo o neutro.
    else
     if (a<0)
        c=-1;
    else
        c=1;
   return c;
}
```

5.6 Diseñe una función que devuelva el porcentaje descontado en una compra, el precio (valor) de la compra y el precio (valor) pagado. Función con parámetros y con valor de retorno.

```
//Ejercicio 5.6 mostrar el porcentaje de descuento de la compra
#include <iostream>
using namespace std;
float desc(float, float);//Se declara la función
void main() {
    float comprao,comprap;//se declaran las varibles
    cout<<"Ingrese el precio original de la compra\n";
    cin>>comprao;
    cout<<"Ingrese el precio que usted pagó\n";
    cin>>comprap;//se lee la variable
```

```
    cout<<"El descuento que se le dio en su compra fue de:
    "<<desc(comprao,comprap)<<"%"<<endl;
}//fin del main

float desc(float original, float pagado){//Se hace el proceso en la función
    float desc;
    desc=100-(pagado*100/original);
    return desc;//Se retorna el valor del descuento.
}
```

5.7 Escribir un programa, que con funciones, verifique si un carácter introducido es una vocal o no. Función con parámetros y con valor de retorno.

```
//Ejercicio 5.7 Mostrar si la letra ingresada es vocal o no
#include <iostream>
using namespace std;
char letra(char);//Se declara la función
int main() {
    char x;//Se declara variable
    int centinela;
    cout<<"Ingrese una letra y se le regresará si es vocal o no\n";
    cin>>x;//Se lee la letra ingresada
    centinela=letra(x);
    if (centinela==0){//Se utiliza un valor centinela el cual indica si es vocal o no
        cout<<"La letra es consonante, no vocal\n";
    }
    else{
        cout<<"La letra es vocal\n";
        }
}//fin del main
char letra(char x){
    int c;
    if (x=='a'||x=='e'||x=='i'||x=='o'||x=='u')//si es vocal la centinela toma valor de 1
        c=1;
    else//sino es 0
        c=0;
   return c;//Regresa el valor para mostrarlo en el programa
}
```

Ejercicios Propuestos Para Solucionar por el Alumno Capitulo 5

1) Escribir una función que reciba por valor los catetos de un triángulo rectángulo y devuelva la hipotenusa del mismo.

2) Hacer una array unidimensional que acepte ocho números enteros. Luego le pregunte al usuario que ingrese un número a buscar, implementar una función que busque ese número, si lo encuentra, debe retornar un valor de verdaderos, en caso contrario, retornar falso.

3) Hacer un programa en C++ que lea dos palabras en dos arrays unidimensionales de caracteres, y luego verifique con una función si ambas palabras tienen la misma cantidad de letras.

6. PUNTEROS

6.1 Concepto de Puntero (Apuntador)

Cada vez que se declara una variable C++, el compilador establece un área de memoria para almacenar el contenido de la variable. Cuando se declara una variable int (entera), por ejemplo, el compilador asigna dos bytes de memoria. El espacio para esa variable se sitúa en una posición específica de la memoria, conocida como *dirección de memoria*. Cuando se referencia (se hace uso) al valor de la variable, el compilador de C++ accede automáticamente a la dirección de memoria donde se almacena el entero. Se puede ganar en eficacia en el acceso a esta dirección de memoria utilizando un *puntero*.

Cada variable que se declara en C++ tiene una dirección asociada con ella. Un *puntero* es una dirección de memoria. El concepto de punteros tiene correspondencia en la vida diaria. Cuando se envía una carta por correo, su información se entrega basada en un puntero que es la dirección de esa carta. Cuando se telefonea a una persona, se utiliza un puntero (el número de teléfono que se marca). Así pues, una dirección de correos y un número de teléfono tienen en común que ambos indican dónde encontrar algo. Son punteros a edificios y teléfonos, respectivamente. Un puntero en C o en C++, también indica dónde encontrar algo, ¿dónde encontrar los datos que están asociados con una variable? *Un puntero C++ es la dirección de una variable*. Los punteros se rigen por estas reglas básicas:

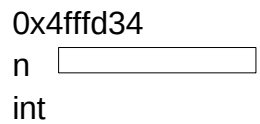
• un *puntero* es una *variable* como cualquier otra;
• una variable puntero contiene una *dirección* que apunta a otra posición en memoria;
• en esa posición se almacenan los datos a los que apunta el puntero;
• un puntero apunta a una variable de memoria.

6.2. Direcciones y Referencias

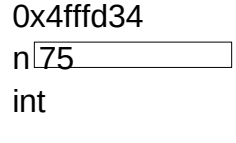
Cuando una variable se declara, se asocian tres atributos fundamentales con la misma: su *nombre*, su *tipo*y su *dirección* en memoria.

Ejemplo

```
int n;   // asocia al nombre n, el tipo int y la dirección de
         // alguna posición de memoria donde se almacena el
         // valor de n
```



Esta caja representa la posición de almacenamiento en memoria. El nombre de la variable está a la izquierda de la caja, la dirección de variable está encima de la caja y el tipo de variable está debajo en la caja. Si el valor de la variable se conoce, se representa en el interior de la caja.



Al valor de una variable se accede por medio de su nombre. Por ejemplo, se puede imprimir el valor de n con la sentencia.

```
cout << n;
```

A la dirección de la variable se accede por medio del *operador de dirección* &. Por ejemplo, se puede imprimir la dirección de n con la sentencia

```
cout << &n;
```

El operador de dirección «&» «opera» (se aplica) sobre el nombre de la variable para obtener sus direcciones. Tiene precedencia de nivel 15 con el mismo nivel que el operador lógico NOT (!) y el operador de preincremento ++.

Ejemplo 6.1

Obtener el valor y la dirección de una variable.

```
main()
{
        int n = 75;
        cout << "n = " << n << endl; // visualiza el valor de n
        cout << "&n = " << &n << endl; // visualiza dirección de n
}
```

Ejecución

```
n = 45
&n = 0x4fffd34
```

Nota

0x4fffd34 es una dirección en código hexadecimal.
“0x” es el prefijo correspondiente al código hexadecimal

6.2.1 Referencias

Una *referencia* es un alias de otra variable. Se declara utilizando el operador de referencia (&) que se añade al tipo de la referencia.

Ejemplo 6.2

Utilización de referencias.

```
void main()
{
        int n = 75;
        int& r = n; // r es una referencia para n
        cout << "n = " << n << ", r = " << r << endl;
}
```

Ejecución

```
n = 75, r = 75
```

Nota

Los dos identificadores n y r son nombres diferentes para la misma variable.

Ejemplo 6.3

Las variables n *y* r *tienen la misma dirección de memoria*

```
void main()
{
        int n = 75;
        int& r = n;
        cout << "&n = " << &n << ", &r =" << &r << endl;
}
```

Ejecución

```
&n = 0x4fffd34, &r = 0x4fffd34
```

Regla

El carácter **&** tiene diferentes usos en C++: 1) cuando se utiliza como prefijo de un nombre de una variable, devuelve la dirección de esa variable; 2) cuando se utiliza como un sufijo de un tipo en una declaración de una variable, declara la variable como sinónimo de la variable que se ha inicializado; y 3) cuando se utiliza como sufijo de un tipo en una declaración de parámetros de una función, declara el parámetro referencia de la variable que se pasa a la función.
& se refiere a la dirección en que se almacena el valor.

6.3 Declaración de Apuntadores

Para declarar un apuntador se especifica el tipo de dato al que apunta, el operador '*', y el nombre del apuntador. La sintaxis es la siguiente:
<tipo de dato apuntado> *<indentificador del apuntador>
Ejemplos:

```
int *ptr1;       // Apuntador a un dato de tipo entero ( int )
char *cad1, *cad2; // Dos apuntadores a datos de tipo carácter ( char )
float *ptr2;     // Apuntador a un dato de tipo punto-flotante ( float )
```

Siempre que aparezca un asterisco (*) en una definición de una variable, ésta es una variable puntero.

6.4 Inicialización (iniciación) de Punteros

Al igual que otras variables, C++ no inicializa los punteros cuando se declaran y es preciso inicializarlos antes de su uso. La *inicialización* de un puntero proporciona a ese puntero la dirección del dato correspondiente. Después de la inicialización, se puede utilizar el puntero para referenciar los datos direccionados. Para asignar una dirección de memoria a un puntero se utiliza el operador de referencia &. Así, por ejemplo,

```
&valor
```

significa «la dirección de valor». Por consiguiente, el método de inicialización (iniciación), también denominado *estático*, requiere:

• Asignar memoria (estáticamente) definiendo una variable y, a continuación, hacer que el puntero apunte al valor de la variable.

```
int i; //declara una variable i
int *p; //declara un puntero a un entero p
p = &i; //asigna la dirección de i a p
```

• Asignar un valor a la dirección de memoria.

```
*p = 50;
```

Cuando ya se ha definido un puntero, el asterisco delante de la variable puntero indica *«el contenido de»* de la memoria apuntada por el puntero y será del tipo dado.
Este tipo de inicialización es **estática**, ya que la asignación de memoria utilizada para almacenar el valor es fijo y no puede desaparecer. Una vez que la variable se define, el compilador establece suficiente memoria para almacenar un valor del tipo de dato dado. La memoria permanece reservada para esta variable y no se puede utilizar para otra cosa durante la ejecución del programa. En otras palabras, no se puede liberar la memoria reservada para una variable. El puntero a esa variable puede ser cambiado, pero la cantidad de memoria reservada permanecerá.

Otro ejemplo de inicialización estática es:

```
int edad = 50; //declara una variable edad de valor 50
int *p_edad = &edad; //declara un puntero de enteros
                     //inicializándolo con la dirección de edad
```

6.5 Asignación de Apuntadores

Se pueden asignar a un apuntador direcciones de variables a través del operador de referenciación (‘&’) o direcciones almacenadas en otros apuntadores. Ejemplos:

```
int i = 5 ;
int *p, *q;
p = &i;// Se le asigna a ’p’ la dirección de ’i’
q = p;  // Se le asigna a ’q’ la dirección almacenada en ’p’ (la misma de ’i’)
```

6.6 Desreferenciación de Apuntadores

La desreferenciación es la obtención del valor almacenado en el espacio de memoria donde apunta un apuntador. En C y C++ esto se hace a través del operador ‘*’, aplicado al apuntador que contiene la dirección del valor.

```
int x = 17, y;
int *p;
p = &x;
cout << "El valor de x es: " << *p << endl; // Imprime 17
y = *p + 3; // A ’y’ se le asigna 20
```

6.7 Apuntadores para Paso de Parámetros por Referencia

El lenguaje *C* no provee una manera de pasar parámetros por referencia. Sin embargo, es posible hacerlo a través del uso de apuntadores. A continuación se muestra un ejemplo del paso de un parámetro por referencia en *C*++, y luego un código equivalente en *C* o *C*++ utilizando un apuntador:

```
void suma(int a, int b, int& r)
{
        r = a + b;
}
void main()
{
        int x;
        suma(7, 5, x);
        cout << "7 + 5 = " << x;
}
```

Ejercicios Resueltos Capitulo 6

6.1 Obtener el valor y la dirección de una variable

```
#include <iostream>
using namespace std;
void main()
{
	int n=75;
	cout<<"n= "<<n<<endl;
	cout<<"&n= "<<&n<<endl;
	system("pause");
}
```

6.2 Obtener el valor p y q, donde a p se le asigna el valor de la dir i y a q se le asigna el valor de p.

```
#include <iostream>
using namespace std;
void main()
{
   int i = 5 ;
   int *p, *q;
   p = &i;        // Se le asigna a ’p’ la dirección de ’i’
   q = p;         // Se le asigna a ’q’ la dirección almacenada en ’p’ (la misma de ’i’)
   cout<<"p="<<p<<endl;
   cout<<"q="<<q<<endl;
   system("pause");
}
```

6.3 Obtener el valor y la dirección de una variable

```
#include <iostream>
using namespace std;
void main()
{
   int x = 17, y;
   int *p;
   p = &x;
   cout << "El valor de x es: " << *p << endl; // Imprime 17
   y = *p + 3; // A ’y’ se le asigna 20
   cout<<"y="<<y<<endl;
   system("pause");
}
```

6.4 Cuál es el resultado?

```
#include <iostream>
using namespace std;
void main()
{
   int aux = 4 ;
   int *puntero = &aux ;
   cout<<&puntero<<endl;
   cout<<*puntero<<endl;;
   system("pause");
}
```

6.5 Qué imprime el programa?

```
#include <iostream>
using namespace std;
void main()
{
        int a, b, c, *p1, *p2;
        p1 = &a;
        a = 1;
        p2 = &b;
        b=0;
        *p1= *p1+5;
        *p2 = 2;
        cout<<b<<" "<<a<<" "<<*p1<<" "<<p2<<endl;
        p1 = p2;
        *p1 = 0;
        p2 = &c;
        cout<<b<<" "<<a<<" "<<*p1<<" "<<p2<<" "<<c<<endl;
        system("pause");
}
```

6.6 Qué imprime el programa?

```
#include <iostream>
using namespace std;
void main()
{
        int *a;
        int b = 3;
        a = &b;
        cout<<*a<<" "<<b<<" "<<a<<endl;
        system("pause");
}
```

6.7 ¿Son iguales o diferentes las direcciones de memoria de r y n? ¿Por qué?

```
void main()
{
        int n = 33;
        int &r = n;
        cout << "&n = “ << &n << ", &r =" << &r << endl;
}
```

6.8 ¿Cuál es la salida del siguiente programa?

```
void main()
{
      int n = 35;
      int *p = &n;
      int &r = *p;
      cout << "r = " << r << endl;
}
```

6.9 ¿Cuál es la salida del siguiente programa?

```
void main ()
{
      int n = 35;
      int *p = fn;
      cout << "*p = " << *p << endl;
}
```

6.10 ¿Qué se imprime al ejecutar el siguiente programa?

```
#include <iostream>
using namespace std;
void main()
{
          int i=100, *p1, *p2;
          p1 = &i;
          p2 = p1;
          if (p1==p2)
          cout<<"p1 apunta a la misma dirección de memoria que p2";
          *p1 = *p1 + 2;
          cout<<" El valor de *p2 es "<< *p2<<endl;
          (*p2)++;
          cout<<" El valor de *p1 es,"
          <<*p1<<endl;
          i--;
          cout<<" El valor de i es "<< i<<endl;
          system("pause");
}
```

Ejercicios Propuestos Para Solucinar por el Alumno Capitulo 6

1) Confeccionar un programa que defina dos punteros a tipos de datos int y float. Acceder mediante estos punteros a otras variables de tipo int y float.

2) Definir tres variables enteras e inicializarlas con los valores 5,10 y 15. Luego definir una variable puntero a entero. Hacer que dicha variable apunte sucesivamente a las distintas variables definidas previamente e imprimir su contenido.

3) Definir dos variables float y un puntero a un tipo de dato float. Hacer que el puntero guarde sucesivamente las direcciones de la primera y segunda variable y cambiar el contenido de las mismas por asignación. Imprimir las dos variables de tipo float.

4) Escribir un programa que ejecute con una sola sentencia lo indicado en cada literal. Suponiendo que han sido declaradas las variables enteras largas valor1 y valor2, y que valor1 se inicialice a 200000.
a) Declarar la variable Ptr que sea un puntero a una variable de tipo long.
b) Asignar la dirección de la variable valor1 a la variable puntero Ptr.
c) Imprimir el valor de la variable apuntada por Ptr.
d) Asignar el valor de la variable apuntada por Ptr a la variable valor2.
e) Imprimir el valor de valor2.
f) Imprimir la dirección de valor1.
g) Imprimir la dirección almacenada en Ptr.

7. ARCHIVOS

Introducción

Toda persona que haya interactuado con los ordenadores (computadoras) tiene alguna noción de lo que es un fichero o archivo.

Se crean para almacenar, por ejemplo, el texto escrito mediante un editor de textos, como el Bloc de Notas, el código que creamos en un entorno de desarrollo o los datos que introducimos en una hoja de cálculo, como Excel. También sabemos que las aplicaciones, como las que creamos en las prácticas, son preservadas en ficheros ejecutables. Por otro lado, día tras día, copiamos o movemos ficheros de una carpeta o directorio del disco duro a otra, o hacia una memoria USB, ¡o los borramos definitivamente!

Hasta ahora, en nuestros programas, los datos que se generan, como los correspondientes a un vector, se guardan durante la ejecución en la memoria volátil de la computadora, por lo que su almacenamiento desaparece tras la ejecución del programa.

En general, la mayor parte de las aplicaciones necesitan leer datos de ficheros o escribir datos en ficheros para un uso posterior por el mismo u otro programa.

7.1 Conceptos Básicos

7.1.1 La Jerarquía de Datos

Podemos establecer una jerarquía para los distintos tipos de datos, organizándolos en diferentes niveles:

Bit: Su valor es 0 o 1. Es la unidad mínima de información.

Byte: 8 bits: Pueden usarse para almacenar caracteres o números pequeños

Campo: Grupo de bytes con un significado, por ejemplo, un nombre de persona, un entero, un real

Registro: Un grupo de campos relacionados. Por ejemplo, los campos nombre, apellidos, edad y DNI pueden ser un registro para describir a una persona en un contexto determinado.

Fichero: Un grupo de campos o registros relacionados.

Base de datos: Un grupo de ficheros relacionados.

7.2 Definición de Ficheros

Un fichero o archivo es una colección ordenada de datos que tienen entre sí una relación y que se almacenan de forma permanente en un dispositivo de memoria no volátil.

7.3 Utilidad de los Ficheros

Los ficheros son enormemente importantes en programación. No sólo debido al hecho evidente de que brindan el soporte que permite la mera existencia de las aplicaciones en el disco duro. Si no existieran los ficheros, por ejemplo, toda la interacción del usuario con una aplicación concreta, como la introducción y la obtención de resultados parciales, tendría que ser laboriosamente recreada cada vez que se ejecutara el programa. La existencia del fichero hace posible que, al terminar una sesión de trabajo, se almacenen los datos y resultados en el disco, para poder retomar la tarea en una sesión posterior.

Por otra parte, los ficheros son útiles como vehículos para el intercambio de información entre diferentes aplicaciones: los datos resultantes escritos en un fichero por el programa A podrán servir como entrada de un programa B.

7.4 El Formato

Un fichero tiene un formato predefinido, conocido de antemano, lo que permite su lectura y/o escritura siguiendo unas pautas preestablecidas.

Para manejar ficheros, siempre deberemos realizar tres operaciones básicas:

- Abrir el fichero.
- Leer datos de él o escribir datos en él.
- Cerrar el fichero.

Además tendremos que comprobar los posibles errores. Por ejemplo, puede ocurrir que intentemos abrir un fichero que realmente no exista, o que queramos escribir en un dispositivo que sea sólo de lectura.
Los sistemas operativos utilizan la extensión de los ficheros (pdf, docx, ppt, txt, cpp, etc.) para asociar al proceso de lectura del fichero la aplicación informática adecuada. Y, por supuesto, la aplicación abrirá correctamente el fichero si el formato es el idóneo.

7.5 Tipos de Ficheros

Como hemos comentado, el contenido y la estructura de un fichero responden a un criterio de diseño libre, elegido por el desarrollador de una aplicación.

En cualquier caso, en relación a la forma en que los datos son almacenados, los ficheros podríamos clasificarlos como:

Ficheros binarios: Contienen una representación exacta del contenido binario de los datos, ceros y unos. Por tanto, no son directamente editables.

Ficheros de texto: Los datos están representados con los caracteres alfanuméricos que los representan. Pueden ser leídos y modificados a través de un editor de texto.

7.6 Programación con Ficheros

Recalquemos que cuando en C++ usamos cout, estamos escribiendo datos en el fichero por defecto o estándar, la pantalla, y que cuando empleamos cin, estamos leyendo datos del fichero por defecto o estándar, el teclado.

Cuando no usamos los ficheros estándar, tanto en C++ como en cualquier otro lenguaje de programación, debemos realizar algunas tareas adicionales:

Abrir el fichero: hay que asociar el fichero (definido a nivel del S.O.) con un objeto que provea la fuente de datos y definir si se va utilizar para entrada o para salida de datos, es decir, para leer o para escribir.

Cerrar el fichero: una vez finalizada la interacción con el objeto que representa el fichero, este hecho debe ser informado al S.O. mediante los métodos apropiados. Así, el S.O. podrá realizar las acciones requeridas para garantizar que el fichero queda en un estado consistente y seguro.

7.6.1 Ficheros en C++

Para trabajar con ficheros en C++ debe incluirse la cabecera fstream.

Dependiendo de lo que deseemos hacer con el fichero, usaremos objetos de las clases:

ifstream (input file stream), clase orientada para la lectura

ofstream (output file stream). clase orientada para la escritura

fstream (file stream), cuando deseemos alternativamente leer o escribir del mismo fichero en el mismo programa.

A diferencia de cin y cout, que son objetos predefinidos de acceso global, listos para su uso, los objetos que representan flujos de datos a/desde ficheros deben ser definidos por el programador.

La apertura del fichero consiste en definir un objeto de la clase deseada (ifstream, ofstream ó fstream).

Para escribir o leer, se usan los operadores de inserción << y extracción >> como si del teclado o consola se tratase.

El fichero se cierra implícitamente cuando el objeto sale del ámbito en el que se ha definido o explícitamente llamando a la función miembro close().

Para el manejo de fichero se va a usar la biblioteca <fstream.h> que contiene define las clases base ifstream (lectura), ofstream(escritura) y fstream (lectura/escritura) en ficheros. Se crean objetos de estas clases de acuerdo a la operación que se requiera realizar.

fopen() Abre un fichero.
fclose() Cierra un fichero.
fgets() Lee una cadena de un fichero.
fputs() Escribe una cadena en un fichero
fseek() Busca un byte especifico de un fichero.
fprintf() Escribe una salida con formato en el fichero.
fscanf() Lee una entrada con formato desde el fichero.
feof() Devuelve cierto si se llega al final del fichero.
ferror() Devuelve cierto si se produce un error.
rewind() Coloca el localizador de posición del fichero al principio del mismo.
remove() Borra un fichero.
fflush() Vacía un fichero.

7.6.2 Escritura de un Fichero Tipo Texto

Vamos a ver un ejemplo, que cree un fichero de texto y escriba algo en él:

Escritura:

- Declaramos un objeto ofstream,
- Por medio de la función open abrimos el fichero
- Escribimos en el fichero los datos necesarios
- Cerramos el fichero con la función Close

Ejercicios Resueltos Capitulo 7

7.1 Escritura de tres líneas de texto en un fichero

```
// 7.1 Escritura en un fichero de texto
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
   ofstream fichero("prueba.txt");
   fichero << "Esto es una línea" << endl;
   fichero << "Esto es otra";
   fichero << " y esto es continuación de la anterior" << endl;
   fichero.close();
    return 0;
}
```

Hay varias cosas que comentar sobre este programa:

Deberemos incluir "fstream" para poder usar los "flujos de datos de ficheros" (cuyo manejo será muy parecido al de la consola, en la que escribíamos con "cout" y leíamos con "cin".
Un fichero de escritura será un "ofstream", abreviatura de "output file stream". En su uso más básico, bastará con indicar el nombre que tendrá el fichero dentro de nuestro programa (por ejemplo "fichero") y, entre paréntesis y comillas, el nombre físico que tendrá en nuestro disco o dispositivo de almacenamiento que estemos usando (por ejemplo, "prueba.txt").
Para escribir datos en el fichero usando "<<", de forma muy similar a como usábamos "cout" para enviar texto a pantalla.
Finalmente, debemos cerrar el fichero con "close()".

7.2 Escritura en un fichero de los 10 primeros enteros

```
// 72 escritura en un fichero de los 10 primeros enteros
#include <fstream>  // Para ofstream
#include <iostream> // Para cout
using namespace std;

int main()
{
  ofstream fich("ejemplo.txt");
  if (!fich)
  {
    cout << "Error al abrir ejemplo.txt\n";
    exit(EXIT_FAILURE);
  }

  for (int i = 0; i < 10; ++i)
    fich << i << endl;
fich.close();
return 0;
}
```

Edita, compila, ejecuta el código y analicemos las líneas más significativas:

ofstream fich("ejemplo.txt");

ofstream es una clase que permite crear la instancia fich para escribir en un fichero. En este caso, se crea un fichero para escritura llamado "ejemplo.txt".

Es conveniente saber dónde el IDE va a crear o leer por defecto el fichero.

if (!fich){...}

Verificar si se ha creado el objeto fich. Si hay error, el operador ! devuelve true.

exit(EXIT_FAILURE);

En general, y en el marco de la asignatura, en presencia de un error al intentar abrir el fichero, se optará por dar por finalizado el programa. Otra opción podría ser volver a solicitar el nombre del fichero al usuario del programa.

fich << i << endl;

A la hora de trabajar con el fichero, la operativa es idéntica a la que se usa con cout. En este caso, el papel del objeto estándar para la salida cout se sustituye por fich, y los datos, en lugar de dirigirse hacia la consola, se almacenan en el fichero "ejemplo.dat". Nótese que basta sustituir fich por cout para visualizar el aspecto que tendrán en el fichero los datos.

Por ultimo procedemos a cerrar el fichero mediante una sentencia del tipo fich.close().

En este ejemplo no es necesario, aunque cerrarlo explícitamente es totalmente legítimo. Lo que ocurre es que cuando cualquier objeto en C++ sale del ámbito en el que se encuentra declarado, una función interna llamada destructor se encarga automáticamente de liberar los recursos que ese objeto haya podido acaparar. En este caso, el destructor de la clase ofstream se encarga de cerrar el fichero, aunque nosotros no lo hayamos hecho.

7.6.3 Lectura de un Fichero Tipo Texto

Si queremos leer de un fichero, los pasos son muy parecidos, excepto que el fichero será un "ifstream" (input file stream) y que leeremos con ">>", de forma parecida a como leíamos desde teclado:

7.3 Lectura de un fichero de texto

```
//lectura de un fichero tipo texto
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
   ifstream fichero("prueba.txt");
   string linea;
   fichero >> linea;
   cout << "Se ha leido: " << endl;
   cout << linea << endl;
   fichero.close();
   return 0;
}
```

Leer toda una línea, incluyendo espacios.
Hasta ahora, hemos leído datos con "cin", pero cuando se trataba de texto, la lectura terminaba en el primer espacio. La alternativa es leer con "getline" y muestre su primera línea completa sería así:

7.4 Leer toda una línea, incluyendo espacios.

```
//7.4 lee toda una línea incluyendo espacios.
#include <iostream>
#include <fstream>
#include<string.h>
using namespace std;
int main()
{
   ifstream fichero("prueba.txt");
   string linea;
   getline(fichero,linea);
   cout << "Se ha leido: " << endl;
   cout << linea << endl;
   fichero.close();
   return 0;
}
```

Lectura hasta el final del fichero.

Normalmente no querremos leer sólo una frase del fichero, sino procesar todo su contenido. Para ayudarnos, tenemos una forma de saber si ya hemos llegado al final del fichero. Es ".eof()", que será verdadero si el fichero ha terminado (EOF es la abreviatura de End Of File, fin de fichero).

De igual modo, podemos comprobar si ha habido algún error (por ejemplo, hemos intentado abrir un fichero que no existe) usando ".fail()".

Por tanto, nuestro programa que avise si el fichero no existe y que muestre todo su contenido, deberá repetirse mientras que no se acabe el fichero, así:

7.5 Lectura hasta el final del fichero.

```
//7.5 lee todo un fichero hasta el final
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
int main()
{
    ifstream fichero("prueba.txt");
    string linea;
   while (! fichero.eof())
    {
        getline(fichero,linea);
        if (! fichero.eof())
            cout << linea << endl;
    }
    fichero.close();
    return 0;
}
```

Esa será la estructura básica de casi cualquier programa que deba leer un fichero completo, de principio a fin: abrir, comprobar que se ha podido acceder correctamente, leer con “while (!fichero.eof())” y cerrar.

Ejercicios Propuestos para Solucionar por el Alumno Capitulo 7

1) Cree un fichero de texto, y escriba algo en él: "Esto es una línea", "Esto es otra"," y esto es continuación de la anterior".

2) Cree un fichero de texto, y escriba algo en él: "Hola", "Adios".

3) Cree un fichero de texto, que genera un alfabeto en el fichero llamado Letras.txt.

4) Cree un fichero de texto, que genere y escriba los 10 primeros enteros.

Fuentes de Información Consultadas

Byron, G., (2005) Programacion en C++ (serie schaum). España: McGraw-Hill.

Lozano, L. (1986). Diagramación y programación estructurada y libre. España: Mc Graw Hill

Joyanes, L. (2008). Fundamentos de programación: algoritmos, estructura de datos y objetos. España: McGraw Hill.

Joyanes, L. Fernández, M. y Rodríguez L. (2003) Fundamentos de programación libro de problemas algoritmos estructura de datos y objetos. México: McGraw Hill.

Joyanes, L. (2012) Fundamentos generales de programación. España: McGraw Hill.

Joyanes, L. y Sánchez, L.(2006). Programación en C++ (serie schaum). España: McGraw-Hill.

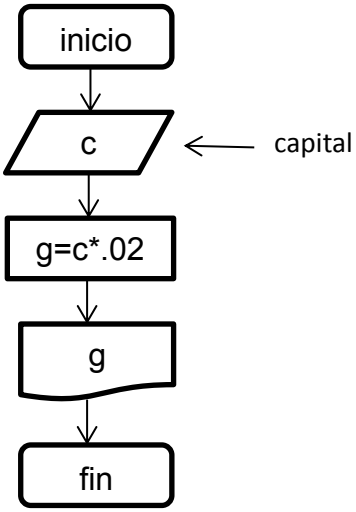
Sitio web del libro, www.mhe.es/joyanes

ANEXO 1

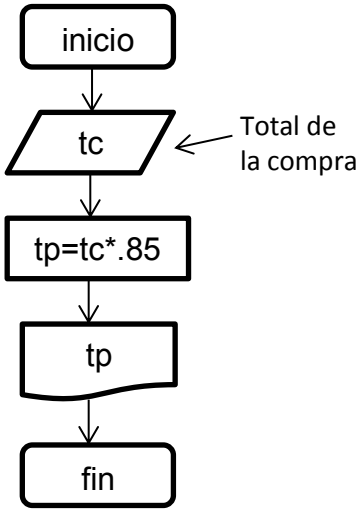
Reporte del Alumno Ejercicios Propuestos

Resultados Ejercicios Propuestos para Solucionar por el Alumno
Capitulo 1

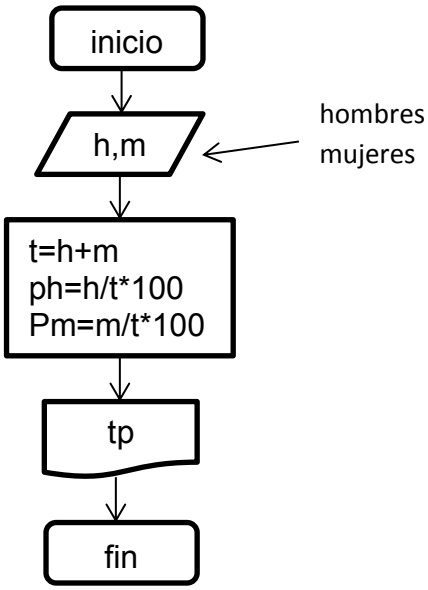
1)



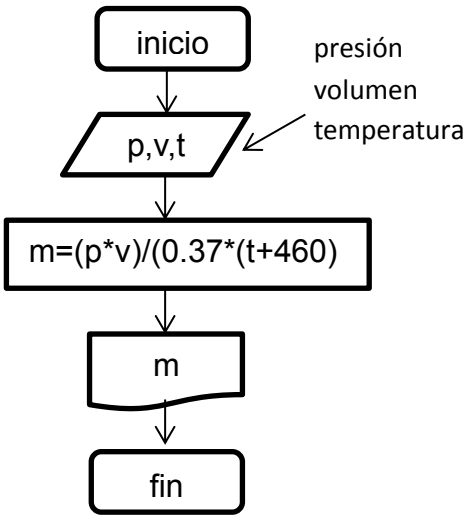
2)



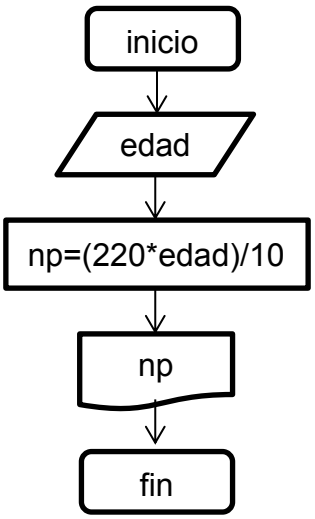
3)



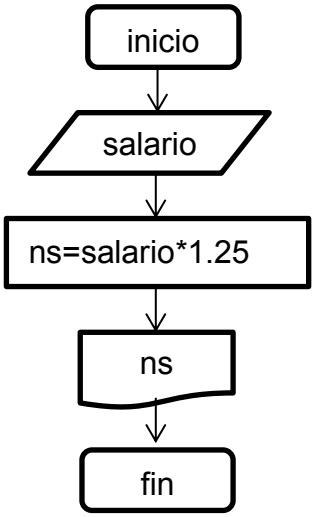
4)



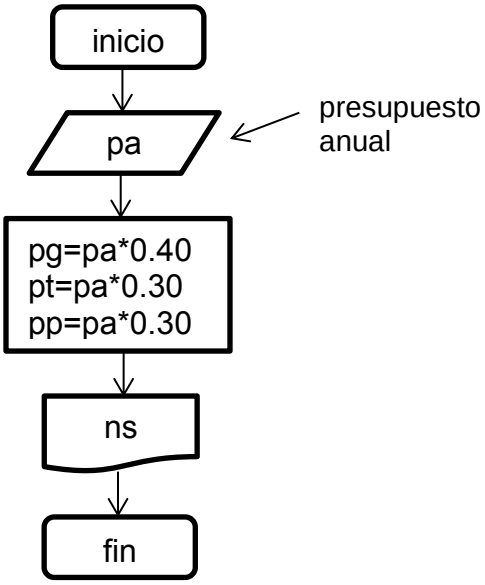
5)


inicio
edad
np=(220*edad)/10
np
fin

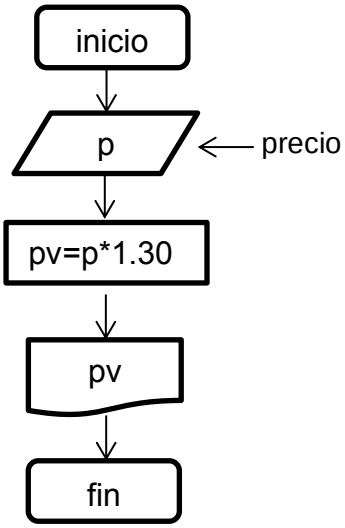
6)


inicio
salario
ns=salario*1.25
ns
fin

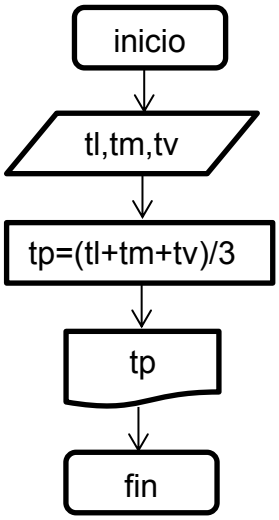
7)


inicio
pa
presupuesto anual
pg=pa*0.40
pt=pa*0.30
pp=pa*0.30
ns
fin

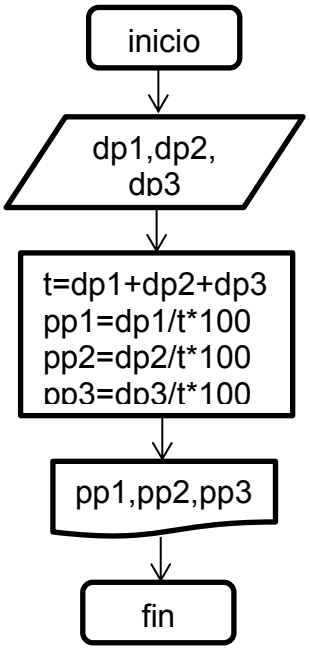
8)


inicio
p
precio
pv=p*1.30
pv
fin

9)


inicio
tl,tm,tv
tp=(tl+tm+tv)/3
tp
fin

10)


inicio
dp1,dp2,
dp3
t=dp1+dp2+dp3
pp1=dp1/t*100
pp2=dp2/t*100
pp3=dp3/t*100
pp1,pp2,pp3
fin

Resultados Ejercicios Propuestos para Solucionar por el Alumno
Capitulo 2

```
//ejercicio 1
#include<iostream>
using namespace std;
void main()
{
	float c, g;
	cout << "teclee el capital:\n ";
	cin >> c;
	g = c * 0.02;
	cout << "ganancia= " << g << endl;
	system("pause");
}//fin del main

// ejercicio 2
#include<iostream>
using namespace std;
void main()
{
	float tc,desc,tp;
	cout << "teclee el total de la compra: ";
	cin >> tc;
	desc = tc * .15;
	tp = tc - desc;
	cout << "total a pagar= " << tp << endl;
	system("pause"); // ya no se requiere en esta version
}//fin del main

//ejercicio 3
#include<iostream>
using namespace std;
void main()
{
	float h,m,ph,pm,t;
	cout << "teclee el total de hombres y mujeres \n";
	cin >>h>>m;
	t = h + m;
	ph = h / t * 100;
	pm = m / t * 100;
	cout << "Poncentaje de hombres= " << ph <<"%"<< endl;
	cout << "Poncentaje de mujeres= " << pm << "%" << endl;
	system("pause");
}//fin del main
```

```
//ejercicio 4
#include<iostream>
using namespace std;
void main()
{
	double p, v, t, m;
	cout << "teclee presion,volumen, y temperatura \n";
	cin >>p>>v>>t;
	m= (p * v)/(0.37*(t + 460));
	cout << "masa= " << m << endl;
	system("pause");
}//fin del main

//ejercicio 5
#include<iostream>
using namespace std;
void main()
{
	int edad,np;
	cout << "teclee la edad \n";
	cin >>edad;
	np= (220-edad)/10;
	cout << "numero de pulsaciones= " << np << endl;
	system("pause");
}//fin del main

//ejercicio 6
#include<iostream>
using namespace std;
void main()
{
	float salario,sn;
	cout << "teclee salario actual \n";
	cin >>salario;
	sn = salario + salario * 0.25;
	cout << "nuevo salario= " << sn << endl;
	system("pause");
}//fin del main
```

```
//Ejercicio 7
#include<iostream>
using namespace std;
void main()
{
	float pa, pg, pt, pp;
	cout<<"Ingrese el presupuesto anual del hospital: \n";
	cin>>pa;
	pg=pa*0.40;
	pt=pa*0.30;
	pp=pa*0.30;
	cout<<"Presupuesto para area de Ginecologia: $"<<pg<<endl;
	cout<<"Presupuesto para area de Traumatologia: $"<<pt<<endl;
	cout<<"Presupuesto para area de Pediatria: $"<<pp<<endl;
  system("pause");
}

//Ejercicio 8
#include<iostream>
using namespace std;
void main()
{
	float c, v;
	cout<<"Ingrese el costo del producto: \n";
	cin>>c;
	v=c+c*0.30;
	cout<<"El precio de venta es: $"<<v<<endl;
  system("pause");
}

//Ejercicio 9
#include<iostream>
using namespace std;
void main()
{
	float l,m,v,tp;
	cout<<"Ingrese el tiempo (minutos) del lunes: \n";
	cin>>l;
	cout<<"Ingrese el tiempo (minutos) del martes: \n";
	cin>>m;
	cout<<"Ingrese el tiempo (minutos) del viernes: \n";
	cin>>v;
	tp=(l+m+v)/3;
	cout<<"El tiempo promedio es de : "<<tp<<" minutos"<<endl;
  system("pause");
}
```

```
//Ejercicio 10
#include<iostream>
using namespace std;
void main()
{
	float p1,p2,p3,I,i1,i2,i3;
	cout<<"Ingrese la inversion de la persona 1: \n";
	cin>>p1;
	cout<<"Ingrese la inversion de la persona 2: \n";
	cin>>p2;
	cout<<"Ingrese la inversion de la persona 3: \n";
	cin>>p3;
	I=(p1+p2+p3);
	i1=(p1/I)*100;
	cout<<"El % de inversion de la persona 1 es de : "<<i1<<"%"<<endl;
	i2=(p2/I)*100;
	cout<<"El % de inversion de la persona 2 es de : "<<i2<<"%"<<endl;
	i3=(p3/I)*100;
	cout<<"El % de inversion de la persona 3 es de : "<<i3<<"%"<<endl;
  system("pause");
}
```

Resultados Ejercicios Propuestos para Solucionar por el Alumno
Capitulo 3

```
//ejercicio 1
#include<iostream>
using namespace std;
void main()
{
	int a,b;
	cout<<"Teclee valor de a y b \n";
	cin>>a>>b;
	if(a==b)
		cout<<a<<"*"<<b<<"="<<(a*b)<<endl;
	else
		if(a>b)
		cout<<a<<"-"<<b<<"="<<(a-b)<<endl;
		else
		cout<<a<<"+"<<b<<"="<<(a+b)<<endl;
system("pause");
}//fin main

//ejercicio 2
#include<iostream>
using namespace std;
void main()
{
	int a,b,c;
	cout<<"Teclee valor de a,b,c \n";
	cin>>a>>b>>c;
	if(a>b && a>c)
		cout<<"a es mayor="<<a<<endl;
	else
	 if(b>a && b>c)
		cout<<"b es mayor="<<b<<endl;
	 else
		cout<<"c es mayor="<<c<<endl;
	system("pause");
}//fin main
```

```
//ejercicio 3
 #include<iostream>
 using namespace std;
 void main()
 {
          int ht,s,ph,he,het;
          cout<<"Teclee las horas trabajadas y el costo \n";
          cin>>ht>>ph;
          if(ht<=40)
                    s=ht*ph;
          else
          {
            he=ht-40;
            if(he<=8)
               s=40*ph+he*ph*2;
            else
              {
                    het=he-8;
                    s=(40*ph)+(8*ph*2)+(het*ph*3);
              }//fin else
       }//fin else
  cout<<"salario=$ "<<s<<endl;
  system("pause");
 }//fin main

//ejercicio 4
#include<iostream>
using namespace std;
void main()
{
          double n1,n2;
          char op;
          cout<<"Teclee valor de n1 y n2 \n";
          cin>>n1>>n2;
          cout<<"Teclee opcion para operacion \n";
          cout<<"(+) suma \n";
          cout<<"(-) resta \n";
          cout<<"(*) multiplicacion \n";
          cout<<"(/) division \n";
          cin>>op;
switch (op)
{
          case '+':
                    cout<<n1<<"+"<<n2<<"="<<(n1+n2)<<endl;
                    break;
          case '-':
                    cout<<n1<<"-"<<n2<<"="<<(n1-n2)<<endl;
                    break;
          case '*':
```

```
                cout<<n1<<"*"<<n2<<"="<<(n1*n2)<<endl;
                break;
        case '/':
                cout<<n1<<"/"<<n2<<"="<<(n1/n2)<<endl;
                break;
        default:
                        cout<<"ERROR";
        }//fin de switch
system("pause");
}//fin main

//ejercicio 5
#include<iostream>
using namespace std;
void main()
{
float n1,n2;
int op;
cout<<"Ingrese el valor numerico de n1 y n2 ";
cin>>n1>>n2;
cout<<"Ingrese la operacion a realizar \n";
cout<<" [1] -> Suma \n";
cout<<" [2] -> Resta \n";
cout<<" [3] -> Multiplicacion \n";
cout<<" [4] -> Division \n";
cin>>op;
switch(op)
{
    case 1:
    cout<<n1<<"+"<<n2<<"="<<n1+n2<<endl;
    break;
    case 2
    cout<<n1<<"-"<<n2<<"="<<n1-n2<<endl;
    break;
    case 3:
    cout<<n1<<"*"<<n2<<"="<<n1*n2<<endl;
    break;
    case 4:
    cout<<n1<<"/"<<n2<<"="<<n1/n2<<endl;
    break;
    default:
    cout<<"ERROR"<<endl;
}
system("pause");
}
```

```
//11.6 promedio general gpo 10 alumnos
 #include<iostream>
 using namespace std;
 void main()
 {

     float  sum=0, cal, prom;
     for( int i=1;i<=10;i++)
      {
           cout<<"Ingrese calificacion: "<<i<<endl;
           cin>>cal;
           sum=sum+cal;
      } // fin del for
  prom=sum/10;
  cout<<"El Prom General Del Grupo Es: "<<prom<<endl;
  system("pause");
 } // fin main

//ejercicio 7
#include<iostream>
using namespace std;
void main()
{
    int cp=0,cn=0,cneg=0,num;
    for(int i=1;i<=20;i++)
    {
        cout<<"Ingrese el numero: ";
        cin>>num;
        if(num==0)
          cn++;
        else
          if(num>0)
            cp++;
        else
            cneg++;
    }
    cout<<"El total de numeros positivos es: "<<cp<<endl;
    cout<<"El total de numeros negativos es: "<<cneg<<endl;
    cout<<"El total de numeros neutros es: "<<cn<<endl;
    system("pause");
}
```

```
//ejercicio 8
  #include<iostream>
  using namespace std;
  void main()
  {
      int num,i;
      cout<<"ingrese num. De tabla \n";
      cin>>num;
      for(i=1; i<=10; i++)
            cout<<num<<"*"<<i<<"="<<num*i<<endl;
  system("pause");
  } //fin del main

//ejercicio 9
  #include<iostream>
  using namespace std;
  void main()
  {
    for (int i=1; i<=10; i++)
    {
        for (int j=1; j<=10; j++)
              cout<<i<<"*"<<j<<"="<<i*j<<endl;
     cout<<"--------------------"<<endl;
    } //fin for i
  system("pause");
 } //fin main

//ejercicio 10
#include<iostream>
using namespace std;
void main()
{
    float cal, sum=0, ca=0, prom;
    cout<<"Ingrese la calificacion, para finalizar teclee un valor fuera del
    rango\n";
    cin>>cal;
    while(cal>=0 && cal<=100)
    {
          sum=sum+cal;
          ca=ca+1;
          cout<<"Ingrese la calificacion \n";
          cin>>cal;
    }
    prom=sum/ca;
    cout<<"El promedio grupal es: "<<prom<<endl;
    system("pause");
}
```

```
//ejercicio 11
#include<iostream>
using namespace std;
void main()
{
    float num, sum=0, c=0, prom;
    cout<<"Ingrese el numero, para salir teclee -999 \n";
    cin>>num;
    while(num!=-999)
    {
          sum=sum+num;
          c=c+1;
          cout<<"Ingrese el numero \n";
          cin>>num;
    }
    prom=sum/c;
    cout<<"El promedio de los "<<c<<" numeros ingresados es: "<<prom<<endl;
    system("pause");
}

//ejercicio 12
#include<iostream>
using namespace std;
void main()
{
   int  ch=0, cm=0;
   char s;
   cout<<" leer sexo m-mujer o h-hombres  \n";
   cin>>s;
   while (s=='m' || s=='h' || s=='M' || s=='H')
   {
    if  (s=='m' || s=='M')
          cm++;
    else
          ch=ch+1;
    cout<<"sexo m o h,fin cualquier caracter \n";
    cin>>s;
   } //fin while
  cout<<"Total Mujeres:  "<<cm<<endl;
  cout<<"Total Hombres:  "<<ch<<endl;
 system("pause");
} //fin main
```

```
//ejercicio 13
#include <iostream>
using namespace std;
void main()
{
   int ca=0,cr=0,cro=0,cv=0,caz=0,d;
   cout<<"Teclee un digito \n";
   cout<<"1-2(amar),3-4(ros),5-6(roj),7-8(ver),0-9(az) \n";
   cin>>d;
   while(d>=0 && d<=9)
   {
         if(d==1 || d==2)
                  ca++;//ca=ca+1;
         else
                  if(d==3||d==4)
                   cr=cr+1;
                  else
                    if(d==5||d==6)
                             cro=cro+1;
                    else
                             if(d==7||d==8)
                               cv++;
                             else
                               caz=caz+1;
         cout<<"digito? fin cualquier num \n";
         cin>>d;
   }//fin while
   cout<<"amarillo="<<ca<<endl;
   cout<<"rosa="<<cr<<endl;
   cout<<"rojo="<<cro<<endl;
   cout<<"verde="<<cv<<endl;
   cout<<"azul="<<caz<<endl;
   system("pause");
}//fin main
```

```
//ejercicio 14
#include<iostream>
using namespace std;
void main()
{
  float ca=0,tp=0,cant,p,sub;
  char op,art[10];
  do
   {
     cout<<"nombre articulo \n";
     cin>>art;
     cout<<"cantidad? y precio$$ \n";
     cin>>cant>>p;
     sub=cant*p;
     tp=tp+sub;
     ca=ca+cant;
     cout<<"otro art s(si),fin cualquier otro caracter \n";
     cin>>op;
   }while(op=='s'||op=='S');
  cout<<"Total a pagar=$"<<tp<<endl;
  cout<<"cantidad de art="<<ca<<endl;
 system("pause");
} //fin main
```

```
//ejercicio 15
#include<iostream>
using namespace std;
void main()
{
    float mv,iva,caja=0,cli=0,efec,cambio,tp;
    char op;
    do
    {
        cout<<"Teclee el monto de ventas: ";
        cin>>mv;
        iva=mv*0.16;
        tp=mv+iva;
        caja=caja+tp;//se acumula en caja
        cli=cli+1;
        cout<<"Subtotal: $"<<mv<<endl;
        cout<<"IVA: $"<<iva<<endl;
        cout<<"Total a pagar: $"<<tp<<endl;
        cout<<"Efectivo recibido: $ ";
        cin>>efec;
        cambio=efec-tp;
        cout<<"Cambio: $"<<cambio<<endl;
        cout<<"Agregar otro cliente presione s(si). Finalice con cualquier otro caracter\n";
        cin>>op;
    }while(op=='s' || op=='S');
    cout<<"Total en caja: $"<<caja<<endl;
    cout<<"Cantidad de clientes: "<<cli<<endl;
    system("pause");
}
```

Resultados Ejercicios Propuestos para Solucionar por el Alumno
Capitulo 4

```
//ejercicio 1
#include<iostream>
using namespace std;
void main()
{
        Int a[11];
        Int n=4;
        For(int i=0;i<11;i++)
        {
            a[i]=n;
            cout<<a[i]<<endl;
            n++;
        }
}

//ejercicio 2
#include<iostream>
using namespace std;
void main()
{
        Int a[100];
        Int n=1;
        Cout<<"Los numeros pares menores que cien: "<<endl;
        For(int i=0;i<100;i++)
        {
            a[i]=n++;
            if(a[i]%2==0)
                cout<<a[i]<<endl;
        }
}

//ejercicio 3
#include <iostream>
#include <conio.h>
using namespace std;
void main ( ) {
   int total, num, j=0, z=0;
   cout << "Ingrese el total de numeros a ingresar: ";
   cin  >> total;
    int numPar[total], numImpar[total];
    // Ingresando numeros y almacenandolos en el vector correspondiente
```

```
    for (int i=0 ; i<total ; i++) {
        cout << "Ingrese numero: ";
        cin  >> num;
        if (num%2 == 0) {
            numPar[j] = num;
            j++;
        } else {
            numImpar[z] = num;
            z++;
        }
    }
    // Mostrando números pares
    cout << "\n\nNumeros pares:\n";
    for (int i=0 ; i<j ; i++)
        cout << numPar[i] << "\n";
    // Mostrando números impares
    cout << "\n\nNumeros impares:\n";
    for (int i=0 ; i<z ; i++)
        cout << numImpar[i] << "\n";
}

// ejercicio 4
// Encontrar el mayor valor del array
#include <iostream>
using namespace std;
#define LONG 15
void main()
{
        int lista[LONG] = { 5, 2, 4, 9, 14,
                            4, 2, 85, 11, 45,
                            25, 12, 45, 6, 99 };
        int valor_mayor, i;
        valor_mayor = lista[0];
        for (i = 1; i < LONG; i++)
        {
          // Almacenar valor actual si es mayor que
          // el último valor mayor encontrado
          if (lista[i] > valor_mayor)
               valor_mayor = lista[i];
        }
   cout << "El número mayor de la lista es: "<< valor_mayor << "\n";
}
```

```
//ejercicio 5
 #include<iostream>
 #include<time.h>
 using namespace std;
 void main()
 {
        srand(time(NULL));
        int a[10],i,j,aux;
        for(i=0;i<=9;i++)
        {
                a[i]=rand()%100;
                cout<<a[i]<<endl;
        }
        //aqui inicia el ordenamiento
        for(i=0;i<=9;i++)
          for(j=0;j<=8;j++)
                if (a[j]>a[j+1])
                        {
                                aux=a[j];
                                a[j]=a[j+1];
                                a[j+1]=aux;
                        }
        //aqui se imprime el arreglo ordenado
        cout<<"\n Arreglo A ordenado \n\n";
        for(i=0;i<=9;i++)
                cout<<a[i]<<endl;
        system("pause");
  }

//ejercicio 6
#include <iostream>
using namespace std;
void main() {
    int a[7],mayor=0,dia;
    for(int i=0;i<=6;i++){
        cout<<"Ingrese los ingresos del día "<<i<<" siendo el 0 el día domingo"<<endl;
        cin>>a[i];
    }
    cout<<endl<<endl;
    cout<<"Arreglo:\n";
        for(int j=0;j<=6;j++){
                cout<<j<<"\t"<<a[j]<<endl;
                if(a[j]>mayor){
                   mayor=a[j];
                   dia=j;
                }
        }
    cout<<endl;
```

```
    switch (dia){
        case 0:
        cout<<"El día que más hubo ingresos fue el domingo\n";
        cout<<"Los ingresos de ese dia son de: $"<<mayor<<endl;
        break;
        case 1:
        cout<<"El día que más hubo ingresos fue el lunes\n";
        cout<<"Los ingresos de ese dia son de: $"<<mayor<<endl;
        break;
        case 2:
        cout<<"El día que más hubo ingresos fue el martes\n";
        cout<<"Los ingresos de ese dia son de: $"<<mayor<<endl;
        break;
        case 3:
        cout<<"El día que más hubo ingresos fue el miércoles\n";
        cout<<"Los ingresos de ese dia son de: $"<<mayor<<endl;
        break;
        case 4:
        cout<<"El día que más hubo ingresos fue el jueves\n";
        cout<<"Los ingresos de ese dia son de: $"<<mayor<<endl;
        break;
        case 5:
        cout<<"El día que más hubo ingresos fue el viernes\n";
        cout<<"Los ingresos de ese dia son de: $"<<mayor<<endl;
        break;
        case 6:
        cout<<"El día que más hubo ingresos fue el sabado\n";
        cout<<"Los ingresos de ese dia son de: $"<<mayor<<endl;
        break;
        default:
        cout<<"Error";
        break;
    }//fin switch
}//fin del main
```

Resultados Ejercicios Propuestos para Solucionar por el Alumno
Capitulo 5

```
//ejercicio 1
#include <iostream>
#include <math.h>
using namespace std;

float pitagoras(float,float); //Prototipo de función

int main(){
 float cat1,cat2,hipo=0;
 cout << "Ingrese cateto 1: ";
 cin >> cat1;
 cout << "Ingrese cateto 2: ";
 cin >> cat2;
 hipo = pitagoras(cat1,cat2);
 cout<<"La hipotenusa del triangulo es: ";
 cout << hipo;
 return 0;
}

float pitagoras(float x, float y){
 float result = 0;
 //sqrt = raíz de un número. pow = potencia de
 //un número elevado a otro. Ambas funciones
 //Se encuentran en la biblioteca cmath.
 result = sqrt(pow(x,2)+pow(y,2));
 return result;
}
```

```
//ejercicio 2
#include <iostream>
using namespace std;
const int dim = 8;

bool BuscaVal(int v[], int);//prototipo de función
void main()
{
	int vector[dim],x,valor;

	cout<<"Ingresar 8 N enteros: ";
	for(int i = 0; i < dim; i++){
		cin >> vector[i];
	}
     cout << "Ingrese un N a buscar en el vector: ";
    cin >> x;
    valor = BuscaVal(vector,x);
   if(valor){ cout << "Valor encontrado!!!"; }
   else{ cout<<"No se encontro valor!!!"; }
}

bool BuscaVal(int v[],int val){
  for(int i = 0; i < dim; i++){
	if(val == v[i]){ return true; }
  }
 return false;
}
```

```
//ejercicio 3
#include <iostream>
#include <cstring.h>
using namespace std;
const int dim = 10;

void CargaVec();
int Compara(char vec1[], char vec2[], int, int);
char v1[dim],v2[dim];
int l1,l2; //longitudes

void main(){
      int x;
     CargaVec();
      x = Compara(v1,v2,l1,l2);//Se pasa v1, v2 y las longitudes de ambos
      if (x==0){ cout<<"Las palabras no son iguales"; }
      else{ cout<<"Palabras iguales"; }
}

void CargaVec(){
       cout << "Ingrese una palabra (maximo 10 caracteres c/u): ";
       cin >> v1;
       cout << "\nIngrese otra palabra: ";
       cin >> v2;
       //strlen, funcion que calcula la longitud de un array.
       //se encuentra en la biblioteca cstring.
       l1 = strlen(v1);
       l2 = strlen(v2);
}

int Compara(char vec1[], char vec2[], int len1, int len2){
      int cuenta=0;
      if (len1 == len2){ // Se comparan para ver si la longitud es la misma
          for(int i = 0; i <= len1; i++){
              if(vec1[i]==vec2[i])
                        cuenta++;
          }
      }else
          return 0;
  return cuenta;
}
```

Resultados Ejercicios Propuestos para Solucionar por el Alumno
Capitulo 6

1)

```
#include <iostream>
using namespace std;
int main()
{
   int x = 10;
   int *pe;
   pe = &x;
   cout << x;         // 10
   cout << "\n";
   cout << *pe;       // imprime lo apuntado por pe: 10
   cout << "\n";
   *pe = 5;           //asignamos 5 a lo apuntado por pe
   cout << x;         // 5
   cout << "\n";
   float valor = 10.9;
   float *pf;
   pf = &valor;
   cout << *pf;      //imprimime lo apuntado por pf: 10.9
   return 0;
}
```

2)

```
#include<iostream>
using namespace std;
int main()
{
   int x1 = 5;
   int x2 = 10;
   int x3 = 15;
   int *pe;
   pe = &x1;
   cout << *pe;
   cout << "\n";
   pe = &x2;
   cout << *pe;
   cout << "\n";
   pe = &x3;
   cout << *pe;
   return 0;
}
```

3)

```
#include<iostream>
using namespace std;
int main()
{
   float f1, f2;
   float *pf;
   pf = &f1;
   *pf = 10.5;
   pf = &f2;
   *pf = 20.2;
   cout << f1;
   cout << "\n";
   cout << f2;
   return 0;
}
```

4)

```
#include <iostream>
using namespace std;
int main ()
{
        long int valor1 = 200000, valor2;
        long int* ptr = &valor1;
        cout << "\nptr->valor1: " << *ptr << endl;
        valor2 = *ptr;
        cout << "\nvalor2: " << valor2 << endl;
        cout << "\n&valor1: " << &valor1 << endl;
        cout << "\nptr: " << ptr << endl;
        return 0;
}
```

Resultados Ejercicios Propuestos para Solucionar por el Alumno
Capitulo 7

1)

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
   ofstream fichero("prueba.txt");
   fichero << "Esto es una línea" << endl;
   fichero << "Esto es otra";
   fichero << " y esto es continuación de la anterior" << endl;
   fichero.close();
   return 0;
}
```

2)

```
#include <iostream>
#include <fstream>
using namespace std;
void main()
{
  ofstream fichero("ejemplo.txt");
  fichero << "Hola" << endl;
  fichero << "Adios" << endl;
  fichero.close();
  system("pause");
}
```

3)

```
#include <iostream>
#include <fstream>
using namespace std;
void main()
{
   ofstream archivo;
   archivo.open("Letras.txt");
            for(char letra='A'; letra <='Z'; letra++)
                  archivo << letra;
   archivo.close();
   system("pause");
}
```

4)

```
#include <fstream>  // Para ofstream
#include <iostream> // Para cout
using namespace std;

int main()
{
  ofstream fich("ejemplo.txt");
  if (!fich)
  {
    cout << "Error al abrir ejemplo.txt\n";
    exit(EXIT_FAILURE);
  }

  for (int i = 0; i < 10; ++i)
    fich << i << endl;
fich.close();
return 0;
}
```

ANEXO 2

Recursos de Evaluación

Instituto Tecnológico de Cd. Juárez

Examen Ordinario Correpondiente Unidad 1

Fundamentos de Programación

Nombre alumno: _____ Gpo. _____

1) Lee cuidadosamente cada uno de los siguientes incisos y selecciona la respuesta correcta.

( )	Es un dispositivo electrónico utilizado para procesar información y obtener resultados.	A. Dispositivo de entrada
( )	Es el conjunto de instrucciones escritas de algún lenguaje de programación y que ejecutadas secuencialmente resuelven un problema especifico.	B. ROM
( )	Sirven para introducir datos (información) en la computadora para su proceso.	C. Lenguaje de programación
( )	Regresan los datos procesados que sirven de información al usuario.	D. Ejecutables
( )	Recibe el nombre de memoria principal o memoria del usuario, en ella se almacena información solo mientras la computadora está encendida.	E. Computadora
( )	Es una memoria estática que no puede cambiar, la computadora puede leer los datos almacenados en ella.	F. Dispositivo de salida
( )	Es una serie de símbolos que sirven para transmitir uno o más mensajes (ideas) entre dos entidades diferentes.	G. RAM
( )	Debe forzosamente existir 4 elementos: Emisor, Receptor, Medio de Comunicación y Mensaje	H. algoritmo
( )	Es un conjunto de símbolos, caracteres y reglas (programas) que le permiten a las personas comunicarse con la computadora.	I. Un compilador
( )	Son aquellos cuyas instrucciones son directamente entendibles por la computadora y no necesitan traducción posterior para que la CPU pueda comprender y ejecutar el programa.	J. Editor de texto
( )	En este lenguaje las instrucciones se escriben en códigos alfabéticos conocidos como nemotécnicos para las operaciones y direcciones simbólicas.	K. Pseudocodigo

( )	Es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.	L. Algoritmos cuantitativos
( )	Son aquellos algoritmos en los que se describen los pasos utilizando palabras.	M. Diagrama de flujo
( )	Son aquellos algoritmos en los que se utilizan cálculos numéricos para definir los pasos del proceso.	N. Algoritmos cualitativos
( )	Es la representación gráfica de las operaciones que realiza un algoritmo.	O. Lenguaje de bajo nivel
( )	Representa en forma descriptiva las operaciones que debe realizar un algoritmo.	P. Lenguaje maquina
( )	Un editor de texto es un programa que permite crear y modificar archivos digitales compuestos únicamente por texto sin formato, conocidos comúnmente como archivos de texto o texto plano.	Q. Interpretes
( )	Es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar.	R. Programa
( )	Traduce frase a frase, realiza el proceso de traducción línea a línea, ejecutando el programa por líneas.	S. La comunicación
( )	Es un archivo binario cuyo contenido se interpreta por el ordenador como un programa.	T. Lenguaje

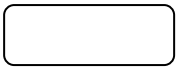
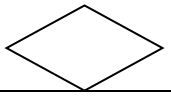
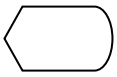
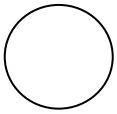
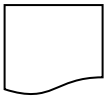
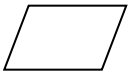
2) Lee cuidadosamente y coloque el número de orden (del 1 al 7) según la metodología para la solución de problemas por medio de computadora.

	Diseño del algoritmos (grafico o no grafico)
	documentación
	codificación
	Definición del problema
	Prueba y depuración
	Análisis del problema
	Mantenimiento

3) Lee cuidadosamente y traza una línea con la respuesta la respuesta correcta.

1- En esta etapa se definen los datos de entrada, salida y el proceso para obtener los resultados (formulas).	Análisis del problema
2- Es la operación de escribir la solución del problema, en una serie de instrucciones detalladas, en un código reconocible por la computadora, la serie de instrucciones detalladas se le conoce como código fuente, el cual se escribe en un lenguaje de programación o lenguaje de alto nivel.	Codificación
3- Requiere una definición clara y precisa.	Definición del problema
4- En la siguiente etapa se realizan la serie de pasos organizados que describe el proceso que se debe seguir, para dar solución al problema.	Diseño del algoritmos
5- Es la guía o comunicación escrita es sus variadas formas, ya sea en enunciados, procedimientos, dibujos o diagramas.	Documentación
6- Para poder realizar este trabajo se requiere que el programa este correctamente documentado.	Mantenimiento
7- Es el proceso de identificar y eliminar errores, para dar paso a la ejecución del programa para poder ver los resultados.	Prueba y depuración.

4) Relaciona las siguientes columnas.

1 	**( )**	Actividad (Representa la ejecución de una o más actividades o procedimientos).
2 	**( )**	Conector (Representa el enlace de actividades con otra dentro de un procedimiento).
3	**( )**	Representa la salida por impresora.
4 	**( )**	Representa la entrada de datos por medio del teclado.
5	**( )**	Decisión (Formula una pregunta o cuestión).
6 	**( )**	Representa la continuidad en otra pagina.
7 	**( )**	Representa salida a monitor.
8 	**( )**	Inicio y término (Abre y/o cierra el diagrama).

5) Diseñe los siguientes diagramas de flujo:

1.- Una persona desea invertir su dinero en un banco, el cual le otorga un 2% de interés. Cuál será la cantidad de dinero que esta persona tendrá al cabo de un año si la ganancia de cada mes es reinvertida?

2.- Al cerrar un expendio de naranjas, 15 clientes que aún no han pagado recibirán un 15% de descuento si compran más de 10 kilos. Determinar cuánto pagara cada cliente y cuanto percibirá la tienda por esas compras.

3.- Una persona que va de compras a la tienda “Enano, S.A.”, decide llevar un control sobre lo que va comprando, para saber la cantidad de dinero que tendrá que pagar al llegar a la caja. La tienda tiene una promoción del 20% de descuento sobre aquellos artículos cuya etiqueta sea roja. Determinar la cantidad de dinero que esta persona deberá pagar.

4.- En una fábrica de computadoras se planea ofrecer a los n clientes un descuento que dependerá del número de computadoras que compre. Si las computadoras son menos de cinco se les dará un 10% de descuento sobre el total de la compra; si el número de computadoras es mayor o igual a cinco pero menos de diez se le otorga un 20% de descuento; y si son 10 o más se les da un 40% de descuento. El precio de cada computadora es de $11,000.

Instituto Tecnológico de Cd. Juárez

Examen Ordinario Correpondiente Unidad 2

Fundamentos de Programación

Nombre alumno: _____ Gpo. _____

INSTRUCCIONES: Lee cuidadosamente cada una de las siguientes preguntas y conteste correctamente.

1) Sintaxis para comentarios.
De línea:_____
De párrafo:_____

2) Sintaxis para declarar variables.
Sintaxis: _____

3) Sintaxis para el cout y cin:
cout: _____
cin: _____

4) ¿Cuál de los siguientes identificadores son válidos?

N	Nombre
MiProblema	AAAAAAAAAA
Mi Juego	Nombre_Apellidos
MiJuego	Saldo_Actual
Write	92
m&m	Universidad
registro	Set 15
* 143Edad	

5) Describa el tipo de datos que almacena:

int	
float	
double	
char	

6) Describa el nombre del operador.

--	
!=	
%	
&&	
*	
/	
||	
++	
<	
=	
==	
>	
>=	

7) Evalue las siguientes expresiones:

a) ¿Cuál es el resultado de la expresión: 50 – 4 * 3 + 2?

b) ¿Cuál es el resultado de la expresión: 5 * (10 – 2 * 4 + 2) - 2?

c) ¿Cuál es el resultado de la expresión: 7 * 5 – 6 % 4 * 4 + 9?

d) ¿Cuál es el resultado de la expresión: 15 * 14 - 3 * 7?

e)¿Cuál es el resultado de la expresión: 3 + 4 * (8 * (4 - (9 + 2) / 6)))?.

7) Encuentre los errores, subraye y explique el error encontrado, en el siguiente programa:

```
#include<ioestream>
using namespace std;
void main()
{
        int n1,n2;
        cout<<”ingrese el valor de n1 y n2 \n”;
        cin>>n1,n2;
        s=n1+n2;
        cout<<n1<<”+”<<n2<<”=”<<S<<endl;
        return 0;
}
```

Describa el error encontrado

1.-

2.-

3.-

4.-

8) Escribir un programa que visualice la letra B con asteriscos.

```
*****
*    *
*    *
*    *
******
*    *
*    *
*    *
*****
```

9) Escribir un programa que lea una variable entera y dos reales y lo visualice.

10). Escribir un programa que lea el largo y el ancho de un rectángulo.

Instituto Tecnológico de Cd. Juárez

Examen Ordinario Correpondiente Unidad 3

Fundamentos de Programación

Nombre alumno: _____ Gpo. _____

INSTRUCCIONES: Lee cuidadosamente cada una de las siguientes preguntas y conteste correctamente.

1) ¿Cuál es la salida del siguiente segmento de programa?

```
for ( int cuenta = 1; cuenta < 5;cuenta++)
        cout << (2 * cuenta) << " ";
```

2) ¿Cuál es la salida de los siguientes bucles?

a)
```
for (int n = 10; n > 0; n = n – 2)
        {
        cout << "Hola ";
        cout << n << endl ;
        }
```
b)
```
for ( double n = 2; n > 0; n = n- 0.5)
        cout << m << " ";
```

3) Considerar el siguiente código de programa.

```
using namespace std;
void main()
{
        int i = 1 , n ;
        cin >> n;
        while (i <= n)
          if ((i % n) == 0)
                ++i;
        cout << i << endl;
}
```

a) ¿Cuál es la salida si se introduce como valor de n, 0?
b) ¿Cuál es la salida si se introduce como valor de n, 1?
c) ¿Cuál es la salida si se introduce como valor de n, 3?

4) Suponiendo que m = 3 y n = 5 ¿Cuál es la salida de los siguientes segmentos de programa?

```
a) for (int i = 0; i < n; i++)
 {
        for (int j = 0; j < i; j++)
                cout << " *";
        cout << endl;
 }
b) for (int i = n; i > 0; i--)
 {
        for (int j = m; j > 0; j--)
                cout << " * ";
        cout << endl;
 }
```

5) Diseñar e implementar un programa que lea un total de 10 números y cuente el número de sus entradas que son positivos, negativos y cero.

6) En una tienda de descuento se efectúa una promoción en la cual se hace un descuento sobre el valor de la compra total según el color de la bolita que el cliente saque al pagar en caja. Si la bolita es de color blanco no se le hará descuento alguno, si es verde se le hará un 10% de descuento, si es amarilla un 25%, si es azul un 50% y si es roja un 100%. Diseñe un programa para determinar la cantidad final que el cliente deberá pagar por su compra. se sabe que solo hay bolitas de los colores mencionados.

7) Al cerrar un expendio de naranjas, 15 clientes que aun no han pagado recibirán un 15% de descuento si compran más de 10 kilos. Diseñe un programa para determinar cuanto pagara cada cliente y cuanto percibirá la tienda por esas compras.

8) Escribir un programa que calcule y visualice el más grande, el más pequeño y la media de n números (n > 0).

9) Diseñar e implementar un programa que solicite a su usuario un valor no negativo n y visualice la siguiente salida (n = 6):

```
1 2 3 4 5 6
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

10) Codificar un programa que escriba la calificación correspondiente a una nota, de acuerdo con el siguiente criterio:

0 a < 5.0	Suspenso
5 a < 6.5	Aprobado
6.5 a 8.5	Notable
8.5 a 10	Sobresaliente
10	Matrícula de honor

Instituto Tecnológico de Cd. Juárez

Examen Ordinario Correpondiente Unidad 4

Fundamentos de Programación

Nombre alumno: _____ Gpo. _____

INSTRUCCIONES: Lee cuidadosamente cada una de las siguientes preguntas y conteste correctamente.

Considerar el siguiente arreglo X de ocho elementos

X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	X[6]	X[7]
2	4	6	8	10	12	14	16

1) Operaciones básicas con arreglos, ponga el resultado según la acción:

Accion	Resultado
cout<<x[1]	
x[4]=11	
suma=x[1]+x[3]	
suma=suma+x[4]	
X[5]=x[5]+3	
X[6]=x[0]+x[1]	

2) Declare e inicializar los siguientes arreglos:
Declarar e inicializar el arreglo X del ejemplo anterior:

Declara un arreglo de 25 elementos doublé y llenarlos por medio del teclado.

3) Según la ejecución del siguiente código, escriba el resultado generado:

programa	Escriba el resultado generado. (tomar el arreglo X)

for(i=0;i<=7;i++)
{
x[i]=x[i]*x[i];
cout<<x[i];
}	

4) Según el proceso cual es el arreglo final:

int tabla[9];
for (i=0;i<=8;i++)
{
tabla[i]=i+1;
cout<<tabla[i];
}
	Imprima el arreglo tabla (arreglo resultante)

5) ¿Cuál es la salida del siguiente código?

```
char simbolo[3] = {'a', 'b', 'c'} ;
for ( int indice=0 ; indice<3 ; indice++)
        cout << simbolo[indice] ;
```

6) ¿Cuál es la salida del siguiente código?

```
int i, aux[10] ;
for (i=0 ; i<10 ; i++)
        aux[i] = 2*i ;
for (i=0 ; i<10 ; i++)
        cout << aux[i] << " " ;
cout << endl ;
for (i=0 ; i<10 ; i = i + 2)
        cout << aux[i] << " " ;
```

7) Analice el siguiente codigo:

```
int K[2]={1,2}, L[3]={3,4,5}, i, j;
for(i=0; i<2; i++)
        for(j=0; j<3; j++)
                cout<<K[i]<<L[j];
```

¿Del código anterior cual será la salida que represente la correcta de las presentadas a continuación?

A) 13 24
B) 13 23 14 24 15 25
C) 11 22 33 44 55
D) 13 14 15 23 24 25

8). Diseñar un programa que determine e imprima el dia de la semana de mayor venta registradas en un array.

9) Se tiene una lista de números enteros almacenados en un array. Se pide ordenarlos: (1) en orden creciente; (2) en orden decreciente.

Instituto Tecnológico de Cd. Juárez

Examen Ordinario Correpondiente Unidad 5

Fundamentos de Programación

Nombre alumno: _____ Gpo. _____

INSTRUCCIONES: Lee cuidadosamente cada una de las siguientes preguntas y conteste correctamente.

1) Instrucción correcta para declarar una función llamada calcula con valor de retorno y 2 parámetros numéricos enteros.

a)void calcula(int,int);
b)int calcula(int,int);
c)int calcula(void);

2) Sintaxis correcta para el llamado a la función grados.

a)grados(f);
b)grados(int f);
c)int grados(f);

3) En la siguiente función indique, que función tiene valor de retorno, pero no tiene parámetros.

a)int calcula(int, float);
b)int area();
c)int calcula(int);

5) En una fábrica de electrodomésticos que opera los 30 días del mes diariamente se registra el porcentaje de fallas. Al final se genera un reporte con el día de mayor porcentaje de fallas y el promedio mensual. ¿Cuál sería el arreglo apropiado para almacenar el porcentaje de fallas?

a) int fallas[30] ;
b) float fallas[29];
c) float fallas[30] ;
d) int fallas[29];

6) ¿Cuál es el valor que regresa la función operación cuando
arreglo = {12,14,17,31,23,120,28,44,78,98}, n=10 y x=31?

```
int operacion (int arreglo[], int x, int n)
{
        for (i=0; i<=n; i++)
                if (arreglo[i]==x)
                        return i;
        return -1;
}
```

a) 3 b) -1 c) 2 d) 31

7) Escribir una función que tenga un argumento de tipo entero y que devuelva la letra P si el número es positivo, y la letra N si es cero o negativo.

8) Escribir una función lógica de dos argumentos enteros, que devuelva true si uno divide al otro y false en caso contrario.

9) Escribir una función que convierta una temperatura dada en grados Celsius a grados Fahrenheit.
La fórmula de conversión es:
F =9/5*C + 32

9) Escribir una función lógica Vocal que determine si un carácter es una vocal.

Instituto Tecnológico de Cd. Juárez

Examen Ordinario Correpondiente Unidad 6

Fundamentos de Programación

Nombre alumno: _____ Gpo. _____

INSTRUCCIONES: Lee cuidadosamente cada una de las siguientes preguntas y conteste correctamente.

1.- Explique la diferencia entre Referenciación y Desreferenciación de apuntadores.

2.- Escriba la sintaxis para declarar un apuntador.

3.- Capture, compile y ejecute el siguiente código.

```
 #include <iostream>
using namespace std;
void main()
{
int i = 5 ;
int *p, *q;
p = &i;
q = p;
cout<<"p="<<p<<endl;
cout<<"q="<<q<<endl;
cout<<"p="<<*p<<endl;
cout<<"q="<<*q<<endl;
system("pause");
}
```

4.-Capture la pantalla de resultados.

4.-Explique el resultado.

Instituto Tecnológico de Cd. Juárez

Examen Ordinario Correpondiente Unidad 7

Fundamentos de Programación

Nombre alumno: _____ Gpo. _____

INSTRUCCIONES: Lee cuidadosamente cada una de las siguientes preguntas y conteste correctamente.

1. Definición de archivos?
2. Explique la función de la biblioteca <fstream.h>.
3. ¿Cuáles son los operadores de inserción y extracción? ¿Cuál es su misión?
4. Explicar las diferencias de los formatos de cin.get() y cin.getline().
5. Diseñe programa que escriba un entero y un valor en coma flotante y una cadena en un archivo llamado DEMO.
6. Edita, compila, ejecuta el siguiente codigo y explique las lineas más significativas.

```
#include <fstream>  // Para ofstream
#include <iostream> // Para cout
using namespace std;

int main()
{
  ofstream fich("ejemplo.txt");
  if (!fich)
  {
    cout << "Error al abrir ejemplo.txt\n";
    exit(EXIT_FAILURE);
  }

  for (int i = 0; i < 10; ++i)
    fich << i << endl;
fich.close();
return 0;
}
```

ANEXO 3

Instrumentacion Didáctica

Asignatura: Programación Básica

1. Datos Generales de la asignatura

Nombre de la asignatura:	**Programación Básica**
Clave de la asignatura:	**MTD-1024**
SATCA[1]:	**2 – 3 – 5**
Carrera:	**Ingeniería Mecatrónica**

2. Presentación

Caracterización de la asignatura

Esta asignatura aporta, al perfil del ingeniero mecatrónico, la capacidad para desarrollar un pensamiento lógico, identificar el proceso de creación de un programa y desarrollo de algoritmos para resolver problemas.

La asignatura proporciona al estudiante de mecatrónica una herramienta para resolver problemas de aplicaciones de la vida ordinaria y de aplicaciones de la ingeniería

La Ingeniería Mecatrónica, está conformada por la integración de la Ingeniería Mecánica, Electrónica, Eléctrica y en los Sistemas Computacionales. Esta última está incluida dentro de las otras áreas mediante el uso de software de diseño y simulación, pero debe existir de manera explícita y separada para la creación de software de interface, automatización y control.

Está diseñada para el logro de competencias específicas dirigidas al aprendizaje de los diferentes dominios: manejo de consola y diseño de algoritmos. Comprende los conceptos básicos de la programación y escribe expresiones aritméticas y lógicas en un lenguaje de programación. Así como el uso y funcionamiento de las estructuras secuenciales, selectivas, arreglos unidimensionales y multidimensionales, punteros y archivos en el desarrollo de aplicaciones. Es capaz de aplicarlos al construir y desarrollar aplicaciones de software que requieran dichas estructuras.

La materia de programación básica pretende formar una lógica de pensamiento estructurada, para la comprensión del funcionamiento de los diferentes lenguajes de programación de alto nivel, indispensables para el desarrollo de tecnología informática.

Programación Básica es el soporte directo de la materia de programación avanzada en la que se abordan temas como POO y visual.

Intención didáctica

La asignatura proporciona al alumno los conceptos esenciales del diseño algorítmico, el temario se organiza en siete temas.
En el primer tema se estudian los conceptos básicos para introducir al estudiante en la programación con la finalidad de obtener las bases conceptuales para abordar las siguientes temáticas. También, se desarrolla el análisis y desarrollo de algoritmos, con la finalidad de dar solución a problemas reales utilizando el razonamiento lógico.

[1] Sistema de Asignación y Transferencia de Créditos Académicos

El segundo tema, tiene la finalidad de obtener y aplicar herramientas necesarias para diseñar e implementar soluciones en un lenguaje de programación, utilizando los conceptos adquiridos.

El tercer tema tiene como objetivo que el alumno identifique, comprenda, seleccione e implemente la estructura de control más adecuada a un problema específico, así como el diseño de bloques de códigos reutilizables, dado que es común encontrar en la práctica problemas cuyas operaciones están condicionadas o deban ejecutarse un número repetido de veces.

El cuarto tema tiene la finalidad de implementar arreglos para una gran variedad de propósitos que proporcionan un medio conveniente de agrupar variables relacionadas y organizar datos de una manera que puedan ser fácilmente procesados. Debe extender el concepto a conjuntos de datos relacionados de diferente tipo de dato como lo son las estructuras y las uniones.

El quinto tema, módulos, se trata como un bloque que ayuda al re uso del software, que incluso es adaptable al establecer módulos que pueden ser llamados con diferentes parámetros.

En el tema 6, punteros, se define el concepto, declaración y uso de punteros, llamadas por valor y referencia.

El Tema 7 es Archivos que pueden ser secuenciales o directos, el manejo de operaciones con archivos y la aplicación de estos junto con el resto de las otras estructuras de programación.

3. Participantes en el diseño y seguimiento curricular del programa

Lugar y fecha de elaboración o revisión	Participantes	Evento
Instituto Tecnológico Superior de Irapuato, del 24 al 28 de agosto de 2009.	Representantes de los Institutos Tecnológicos de: Apizaco, Celaya, Ciudad Cuauhtémoc, Cuautla, Durango, Guanajuato, Hermosillo, Huichapan, Irapuato, Jilotepec, Jocotitlán, La Laguna, Oriente del Estado de Hidalgo, Pabellón de Arteaga, Parral, Reynosa, Saltillo, San Luis Potosí, Tlalnepantla, Toluca y Zacapoaxtla.	Reunión Nacional de Diseño e Innovación Curricular para el Desarrollo y Formación de Competencias Profesionales de las Carreras de Ingeniería Eléctrica, Ingeniería Electromecánica, Ingeniería Electrónica e Ingeniería Mecatrónica.
Instituto Tecnológico de Mexicali, del 25 al 29 de enero del 2010.	Representantes de los Institutos Tecnológicos de: Apizaco, Celaya, Ciudad Cuauhtémoc, Cuautla, Durango, Guanajuato, Hermosillo, Huichapan, Irapuato, Jilotepec, Jocotitlán, La Laguna, Mexicali, Oriente del Estado de Hidalgo,	Reunión Nacional de Consolidación de los Programas en Competencias Profesionales de las Carreras de Ingeniería Eléctrica, Ingeniería Electromecánica, Ingeniería Electrónica e Ingeniería Mecatrónica.

	Pabellón de Arteaga, Reynosa, Saltillo, San Luis Potosí, Toluca y Zacapoaxtla.	
Instituto Tecnológico de la Laguna, del 26 al 29 de noviembre de 2012.	Representantes de los Institutos Tecnológicos de: Apizaco, Celaya, Chapala, Cd. Cuauhtémoc, Colima, Culiacán, Huixquilucan, La Laguna, León, Nuevo Laredo, Nuevo León, Oriente del Estado de Hidalgo, Querétaro, Tlalnepantla, Uruapan, Veracruz y Zacapoaxtla.	Reunión Nacional de Seguimiento Curricular de los Programas en Competencias Profesionales de las Carreras de Ingeniería Eléctrica, Ingeniería Electromecánica, Ingeniería Electrónica, Ingeniería Mecánica e Ingeniería Mecatrónica.
Instituto Tecnológico de Toluca, del 10 al 13 de febrero de 2014.	Representantes de los Institutos Tecnológicos de: Aguascalientes, Boca del Río, Celaya, Mérida, Orizaba, Puerto Vallarta y Veracruz.	Reunión de Seguimiento Curricular de los Programas Educativos de Ingenierías, Licenciaturas y Asignaturas Comunes del SNIT.
Tecnológico Nacional de México, del 25 al 26 de agosto de 2014.	Representantes de los Institutos Tecnológicos de: Aguascalientes, Apizaco, Boca del Río, Celaya, Cerro Azul, Cd. Juárez, Cd. Madero, Chihuahua, Coacalco, Coatzacoalcos, Durango, Ecatepec, La Laguna, Lerdo, Matamoros, Mérida, Mexicali, Motúl, Nuevo Laredo, Orizaba, Pachuca, Poza Rica, Progreso, Reynosa, Saltillo, Santiago Papasquiaro, Tantoyuca, Tlalnepantla, Toluca, Veracruz, Villahermosa, Zacatecas y Zacatepec. Representantes de Petróleos Mexicanos (PEMEX).	Reunión de trabajo para la actualización de los planes de estudio del sector energético, con la participación de PEMEX.

4. Competencia(s) a desarrollar

Competencia(s) específica(s) de la asignatura
Plantea metodológicamente la solución de problemas susceptibles de ser computarizados a través del manejo de técnicas estructuradas de diseño y formulación de algoritmos.

5. Competencias previas

- Utiliza las herramientas básicas de la computación para crear archivos de texto y buscar información en Internet.

6. Temario

No.	Temas	Subtemas
1	Conceptos Básicos y Algoritmos	1.1 Clasificación del software de: sistemas y aplicación. 1.2 Definición de: Algoritmo, Lenguaje de Programación, Programa, Programación, Paradigmas de programación, Editores de texto, Compiladores e intérpretes, Ejecutables, Consola de línea de comandos. 1.3 Análisis de problemas. 1.4 Representación de algoritmos: gráfica y pseudocódigo. 1.5 Diseño de algoritmos aplicados a problemas. 1.6 Diseño algorítmico de funciones
2	Introducción a la Programación	2.1 Características del lenguaje de programación 2.2 Estructura básica de un programa. 2.3 Traducción de un programa: compilación, enlace de un programa, errores en tiempo de compilación. 2.4 Ejecución de un programa. 2.5 Elementos del lenguaje: datos, literales y constantes, identificadores, variables, parámetros, operadores, entrada y salida de datos. 2.6 Errores en tiempo de ejecución.
3	Control de flujo	3.1 Estructuras secuenciales. 3.2 Estructuras selectivas: simple, doble y múltiple. 3.3 Estructuras iterativas: repetir mientras, hasta, desde 3.4 Diseño e implementación de funciones
4	Arreglos, Estructuras y Uniones	4.1 Unidimensionales: conceptos básicos, operaciones y aplicaciones. 4.2 Multidimensionales: conceptos básicos, operaciones y aplicaciones. 4.3 Aplicaciones en programas
5	Módulos	5.1 Diferencia entre Funciones y Procedimientos. 5.2 Activación / Invocación de Funciones y Procedimientos 5.3 Uso de Funciones y Procedimientos con Parámetros.

6	Punteros	6.1 Declaración e inicialización 6.2 Operadores de apuntador 6.3 Uso de apuntadores 6.4 Llamadas por valor y referencia
7	Archivos	7.1 Creación, lectura, escritura y cierre de archivos de acceso secuencial 7.2 Creación, lectura, escritura y cierre de archivos de acceso directo

7. Actividades de aprendizaje de los temas

1 Conceptos Básicos y Algoritmos	
Competencias	Actividades de aprendizaje
Específica(s): Domina los conceptos básicos de la programación para analizar problemas y representar su solución mediante algoritmos. Genéricas: • Capacidad de análisis y síntesis. • Capacidad de pensamiento lógico, algorítmico, heurístico, analítico y sintético. • Resolución de problemas. • Búsqueda y manejo de información. • Capacidad crítica y autocrítica	• Investigar la clasificación del software. • Reconocer los conceptos básicos: algoritmo, programa, programación, paradigmas de programación utilizando mapas conceptuales, mentales, cuadros sinópticos, etc. • Conocer el entorno de un lenguaje de programación. • Manejar la consola para la compilacion y ejecucion de programas. • Explicar los conceptos básicos para la formulación de algoritmos, así como sus ventajas y desventajas. • Generar un catálogo de problemas para su análisis y solución. • Resolver y analizar problemas cotidianos. • Investigar los diferentes métodos para la representacion de un algoritmo: diagrama de flujo, N-S (Nassi-Shneiderman), Pseudocódigo, Descripción Narrada.
2 Introducción a la Programación	
Competencias	Actividades de aprendizaje
Específica(s): Conoce las características principales de un lenguaje de programación que le permiten codificar algoritmos, compilar y ejecutar programas. Genéricas: • Capacidad de análisis y síntesis. • Capacidad de pensamiento lógico, algorítmico, heurístico, analítico y sintético. • Resolución de problemas. • Búsqueda y manejo de información.	• Realizar un mapa conceptual sobre los tipos de software y los conceptos básicos de programación. • Buscar y analizar información necesaria para Instalar y configurar el compilador del lenguaje de programación a utilizar. • Realizar cambios en expresiones lógicas y algebraicas de un programa modelo y analizar los resultados obtenidos. • Realizar programas completos de menor a mayor grado de dificultad y con base en cada una de las instrucciones que los componen

• Capacidad crítica y autocrítica	enseñar la sintaxis del lenguaje
3 Control de flujo	
Competencias	Actividades de aprendizaje
Específica(s): Construye programas utilizando estructuras condicionales y repetitivas para aumentar su funcionalidad Genéricas: • Capacidad de análisis y síntesis. • Capacidad de pensamiento lógico, algorítmico, heurístico, analítico y sintético. • Resolución de problemas. • Búsqueda y manejo de información. • Capacidad crítica y autocrítica. • Búsqueda y manejo de información. • Capacidad de aprender.	• Realizar una investigación sobre el funcionamiento y aplicación de las estructuras de selección y de repetición. • Diseñar programas donde se utilicen las estructuras de repetición y selección. • Construir programas que implementen métodos o funciones.
4 Arreglos, Estructuras y Uniones	
Competencias	Actividades de aprendizaje
Específica(s): Construye programas que utilicen arreglos unidimensionales y multidimensionales para solucionar problemas. Genéricas: • Capacidad de análisis y síntesis. • Capacidad de pensamiento lógico, algorítmico, heurístico, analítico y sintético. • Resolución de problemas. • Búsqueda y manejo de información. • Capacidad crítica y autocrítica. • Búsqueda y manejo de información. • Capacidad de aprender. • Capacidad de aplicar los conocimientos en la práctica.	• Diseñar algoritmos que utilicen arreglos unidimensionales y multidimensionales. • Desarrollar programas para implementar las operaciones básicas en arreglos. • Realizar programas que involucren estructuras y uniones.
5 Módulos	
Competencias	Actividades de aprendizaje
Específica(s): Aplica funciones y procedimientos en programas. Genéricas: • Capacidad de análisis y síntesis.	• Identificar la diferencia entre funciones y procedimientos participando en discusión grupal. • Aplicar la activación / invocación de funciones y procedimientos en ejercicios propuestos.

• Capacidad de pensamiento lógico, algorítmico, heurístico, analítico y sintético. • Resolución de problemas. • Búsqueda y manejo de información. • Capacidad crítica y autocrítica. • Búsqueda y manejo de información. • Capacidad de aprender. • Capacidad de aplicar los conocimientos en la práctica.	• Utilizar funciones y procedimientos en programas.
6 Punteros	
Competencias	Actividades de aprendizaje
Específica(s): Declara y aplica apuntadores como manera de manipular información en llamadas por valor y referencia. Genéricas: • Capacidad de análisis y síntesis. • Capacidad de pensamiento lógico, algorítmico, heurístico, analítico y sintético. • Resolución de problemas. • Búsqueda y manejo de información. • Capacidad crítica y autocrítica. • Búsqueda y manejo de información. • Capacidad de aprender. • Capacidad de aplicar los conocimientos en la práctica.	• Declarar apuntadores. • Utiliza operadores con apuntadores. • Hacer llamadas a módulos por valor y observa los resultados, saca conclusiones. • Hacer llamadas a módulos por referencia y observa los resultados, saca conclusiones. • Realizar aplicaciones que involucran el uso de apuntadores
7 Archivos	
Competencias	Actividades de aprendizaje
Específica(s): Utiliza las librerías de manejos de archivos para crearlos, leerlos, escribirlos, buscar registros y cerrarlos, ya sean secuenciales o directos. Genéricas: • Capacidad de análisis y síntesis. • Capacidad de pensamiento lógico, algorítmico, heurístico, analítico y sintético. • Resolución de problemas. • Búsqueda y manejo de información. • Capacidad crítica y autocrítica. • Búsqueda y manejo de información. • Capacidad de aprender. • Capacidad de aplicar los conocimientos en la práctica.	• Definir tipos de archivos secuenciales y directos • Crear archivos secuenciales y directos, con diferentes propiedades: W, R, RW, A, etc. • Leer, escribir datos de archivos ya sean tipos de datos simples o estructurados. • Hacer búsqueda de registros en archivos para luego modificar el dato.

8. Práctica(s)

- Investiga los tipos de software que se utilizan en una organización e identificarlos dentro de la clasificación del software.
- Elabora ejercicios que impliquen entrada y salida de datos, así como el uso de operadores, operandos.
- Soluciona problemas con algoritmos a partir de enunciados proporcionados por el profesor.
- Crea, compila y ejecuta programas.
- Declara variables y usa expresiones aritméticas, relacionales, lógicas y de igualdad.
- Resuelve problemas que utilicen entrada y salida de datos.
- Soluciona problemas utilizando sentencias de control.
- Implementa soluciones con arreglos.

9. Proyecto de asignatura

El objetivo del proyecto que planteé el docente que imparta esta asignatura, es demostrar el desarrollo y alcance de la(s) competencia(s) de la asignatura, considerando las siguientes fases:

- **Fundamentación:** marco referencial (teórico, conceptual, contextual, legal) en el cual se fundamenta el proyecto de acuerdo con un diagnóstico realizado, mismo que permite a los estudiantes lograr la comprensión de la realidad o situación objeto de estudio para definir un proceso de intervención o hacer el diseño de un modelo.
- **Planeación:** con base en el diagnóstico en esta fase se realiza el diseño del proyecto por parte de los estudiantes con asesoría del docente; implica planificar un proceso: de intervención empresarial, social o comunitario, el diseño de un modelo, entre otros, según el tipo de proyecto, las actividades a realizar los recursos requeridos y el cronograma de trabajo.
- **Ejecución:** consiste en el desarrollo de la planeación del proyecto realizada por parte de los estudiantes con asesoría del docente, es decir en la intervención (social, empresarial), o construcción del modelo propuesto según el tipo de proyecto, es la fase de mayor duración que implica el desempeño de las competencias genéricas y especificas a desarrollar.
- **Evaluación:** es la fase final que aplica un juicio de valor en el contexto laboral-profesión, social e investigativo, ésta se debe realizar a través del reconocimiento de logros y aspectos a mejorar se estará promoviendo el concepto de "evaluación para la mejora continua", la metacognición, el desarrollo del pensamiento crítico y reflexivo en los estudiantes.

10. Evaluación por competencias

Son las técnicas, instrumentos y herramientas sugeridas para constatar los desempeños académicos de las actividades de aprendizaje.

- Mapa conceptual
- Problemario
- Examen
- Esquemas
- Representaciones gráficas o esquemáticas
- Mapas mentales
- Ensayos
- Reportes de prácticas
- Resúmenes
- Rúbrica

- Exposiciones orales.
- Lista de cotejo
- Matriz de valoración
- Guía de observación

11. Fuentes de información

1. Luis Joyanes Aguilar, Fundamentos de Programación, Ed. Prentice Hall.
2. Jesús J. García Molina Introducción a la programación un Enfoque Algorítmico, Ed. Paraninfo.
3. Leobardo López Román, Metodología de la Programación Orientada a Objetos, Ed. Alfaomega.
4. Cairo Osvaldo, Metodología de la Programación, Ed. Alfaomega.
5. Deitel y Deitel. Como Programar en C++ quinta Edición. Prentice Hall.
6. Deitel y Deitel. Como Programar en C# quinta Edición. Prentice Hall.
7. Deitel y Deitel. Java como programar. Séptima edición. Prentice Hall.
8. Joyanes Aguilar, Luis Fernández, Azuela Matilde, Rodríguez Baena Luis, Fundamentos de Programación Libro de Problemas Algoritmos Estructura de Datos y Objetos. 2a. edición Ed. Mc. Graw Hill
9. Luis Joyanes Aguilar. Programación en JAVA 2 1ª Edición. Mc Graw Hill.
10. Martín Flowler Kendall Scott. UML Gota a Gota. Addison Wesley.
11. Ramírez Felipe, Introducción a la Programación, Algoritmos y su Implementación En Vb.Net C# Java y C++, 2a. edición, Alfa Omega.
12. Jean-Paul Tremblay, Richar B. Bunt. Introducción a la Ciencia de Las Computadoras. Enfoque Algorítmico. McGraw Hill.
13. Bjarne Storstrup. Lenguaje de Programación C/C++.
14. Cairo Battistutti Osvaldo, Metodología de la Programación, Algoritmos Diagramas de Flujo y Programas, 3a. edición, Alfa Omega.
15. Flores Cueto, Juan José, Método de las 6'D UML – Pseudocódigo – Java Enfoque Algorítmico, Serie Textos Universitarios Facultad de Ingeniería y Arquitectura, ed. Universidad de San Martín de Porres, (http://books.google.com/).

TECNOLÓGICO NACIONAL DE MÉXICO	**TECNOLÓGICO NACIONAL DE MÉXICO CAMPUS DE CIUDAD JUÁREZ** **INSTRUMENTACIÓN DIDÁCTICA** **FORMATO**	**Revision 1** ISO 9001:2015 ISO 14001:2015

Subdirección Académica

Periodo: Enero - Junio 2021

Nombre de la asignatura:	PROGRAMACION BASICA
Plan de estudios:	IMCT-2010-229
Clave de asignatura:	5GA
Horas teoría – horas prácticas – créditos:	2 - 3 - 5

1. CARACTERIZACIÓN DE LA ASIGNATURA

Esta asignatura aporta, al perfil del ingeniero mecatrónico, la capacidad para desarrollar un pensamiento lógico, identificar el proceso de creación de un programa y desarrollo de algoritmos para resolver problemas. La asignatura proporciona al estudiante de mecatrónica una herramienta para resolver problemas de aplicaciones de la vida ordinaria y de aplicaciones de la ingeniería La Ingeniería Mecatrónica, está conformada por la integración de la Ingeniería Mecánica, Electrónica, Eléctrica y en los Sistemas Computacionales. Esta última está incluida dentro de las otras áreas mediante el uso de software de diseño y simulación, pero debe existir de manera explícita y separada para la creación de software de interface, automatización y control. Está diseñada para el logro de competencias específicas dirigidas al aprendizaje de los diferentes dominios: manejo de consola y diseño de algoritmos. Comprende los conceptos básicos de la programación y escribe expresiones aritméticas y lógicas en un lenguaje de programación. Así como el uso y funcionamiento de las estructuras secuenciales, selectivas, arreglos unidimensionales y multidimensionales, punteros y archivos en el desarrollo de aplicaciones. Es capaz de aplicarlos al construir y desarrollar aplicaciones de software que requieran dichas estructuras. La materia de programación básica pretende formar una lógica de pensamiento estructurada, para la comprensión del funcionamiento de los diferentes lenguajes de programación de alto nivel, indispensables para el desarrollo de tecnología informática. Programación Básica es el soporte directo de la materia de programación avanzada en la que se abordan temas como POO y visual.

2. INTENCIÓN DIDÁCTICA

La asignatura proporciona al alumno los conceptos esenciales del diseño algorítmico, el temario se organiza en siete temas. En el primer tema se estudian los conceptos básicos para introducir al estudiante en la programación con la finalidad de obtener las bases conceptuales para abordar las siguientes temáticas. También, se desarrolla el análisis y desarrollo de algoritmos, con la finalidad de dar solución a problemas reales utilizando el razonamiento lógico. El segundo tema, tiene la finalidad de obtener y aplicar herramientas necesarias para diseñar e implementar soluciones en un lenguaje de programación, utilizando los conceptos adquiridos. El tercer tema tiene como objetivo que el alumno identifique, comprenda, seleccione e implemente la estructura de control más adecuada a un problema específico, así como el diseño de bloques de códigos reutilizables, dado que es común encontrar en la práctica problemas cuyas operaciones están condicionadas o deban ejecutarse un número repetido de veces. El cuarto tema tiene la finalidad de implementar arreglos para una gran variedad de propósitos que proporcionan un medio conveniente de agrupar variables relacionadas y organizar datos de una manera que puedan ser fácilmente procesados. Debe extender el concepto a conjuntos de datos relacionados de diferente tipo de dato como lo son las estructuras y las uniones. El quinto tema, módulos, se trata como un bloque que ayuda al re uso del software, que incluso es adaptable al establecer módulos que pueden ser llamados con diferentes parámetros. En el tema 6, punteros, se define el concepto, declaración y uso de punteros, llamadas por valor y referencia. El Tema 7 es Archivos que pueden ser secuenciales o directos, el manejo de operaciones con archivos y la aplicación de estos junto con el resto de las otras estructuras de programación.

3. COMPETENCIA DE LA ASIGNATURA

Plantea metodológicamente la solución de problemas susceptibles de ser computarizados a través del manejo de técnicas estructuradas de diseño y formulación de algoritmos.

4. ANÁLISIS POR COMPETENCIAS ESPECÍFICAS

Competencia No.: 1 Descripción: Específica(s): Domina los conceptos básicos de la programación para analizar problemas y representar su solución mediante algoritmos.

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje	Actividades de Enseñanza	Desarrollo de competencias genéricas	Horas teórico-práctica
Unidad # 1 Conceptos Básicos y Algoritmos 1.1 Clasificación del software de: sistemas y aplicación.	El alumno participará activamente en las actividades de la apertura del curso. El alumno tomará apuntes sobre la clasificación del software.	El docente realizará la apertura del curso. El docente explicará el tema de la clasificación del software.	Capacidad de análisis y síntesis. Habilidades de gestión de información. Habilidad para buscar y analizar información.	6 Horas teoría – 9 Horas prácticas 3 Semanas
1.2 Definición de: Algoritmo, Lenguaje de Programación, Programa, Programación, Paradigmas de programación, Editores de texto, Compiladores e intérpretes, Ejecutables, Consola de línea de comandos.	El alumno tomará apuntes sobre la Definición de: Algoritmo, Lenguaje de Programación, Programa, Programación, Paradigmas de programación, Editores de texto, Compiladores e intérpretes, Ejecutables, Consola de línea de comandos.	El docente por medio de Técnica Expositiva presentará a los alumnos Definición de: Algoritmo, Lenguaje de Programación, Programa, Programación, Paradigmas de programación, Editores de texto, Compiladores e intérpretes, Ejecutables, Consola de línea de comandos.	Capacidad de análisis y síntesis. Habilidades de gestión de información. Habilidad para buscar y analizar información.	
1.3 Análisis de problemas.	El alumno tomará apuntes sobre el tema de análisis de problemas. El alumno elaborará prácticas correspondientes de análisis de problemas y entregará reporte correspondiente.	El docente explicará el tema de Análisis de problemas . El docente mediante la técnica de Demostración – Ejecución explicará a los alumnos el desarrollo del análisis de problemas. El docente solicitará la realización de prácticas de Análisis de problemas.	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	
1.4 Representación de algoritmos: gráfica y pseudocódigo.	El alumno atenderá la demostración, sobre algoritmos gráficos y pseudocodigo, realizará y entregará reporte de prácticas.	El docente Mediante la técnica de Demostración – Ejecución explicará a los alumnos el desarrollo de Algoritmos gráficos y pseudocodigo. El docente solicitará al alumno la realización de prácticas correspondientes de diagramas de flujos con su análisis correspondiente.	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	

1.5 Diseño de algoritmos aplicados a problemas.	El alumno atenderá la demostración, sobre el Diseño de algoritmos aplicados a problemas, realizará y entregará reporte de prácticas.	El docente Mediante la técnica de Demostración – Ejecución explicará a los alumnos el diseño de Algoritmos aplicados a problemas. El docente solicitará al alumno la realización de prácticas correspondientes a diseño de algoritmos aplicados a problemas.	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	
1.6 Diseño algorítmico de funciones	El alumno atenderá la demostración, sobre el diseño de algorítmico de funciones, realizará y entregará reporte de prácticas. El alumno realizará examen de la competencia 1	El docente Mediante la técnica de Demostración – Ejecución explicará a los alumnos el diseño Algorítmico de funciones. El docente solicitará al alumno la realización de prácticas correspondientes de diseño algorítmico de funciones. El docente aplicará examen de la competencia 1	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	

CRITERIOS DE EVALUACIÓN

Indicadores de alcance	Valor del indicador
A. Se adapta a situaciones y contextos complejos: Trabajo en equipo.	Indica la ponderación de los criterios de valoración definidos en el punto anterior.
B. Hace aportaciones a las actividades académicas desarrolladas: Realiza la investigación de los conceptos señalados, utilizando bibliografía en un segundo idioma (inglés).	
C. Propone y/o explica soluciones o procedimientos no vistos en clase (Creatividad): Propone actividades para la generación de ideas de proyectos orientados al espíritu emprendedor.	
D. Introduce recursos y experiencias que promueven un pensamiento Crítico: Se apoya en las sugerencias y observaciones de sus compañeros de clase, después de su exposición de la idea emprendedora	
E. Incorpora conocimientos y actividades interdisciplinarias en su aprendizaje: Incorpora temas relacionados con su carrera para iniciar con su proyecto de la materia	
F. Realiza su trabajo de manera autónoma y autorregulada: Capacidad de organizar la planeación de los temas, utilizando las estrategias plasmadas en la instrumentación didáctica, anticipándose a las actividades de aprendizaje señaladas participando en trabajos colaborativos para fortalecer a la ciudad	

NIVELES DE DESEMPEÑO:

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple al menos cinco de los siguientes indicadores a) Se adapta a situaciones y contextos complejos. Puede trabajar en equipo, reflejar sus conocimientos en la interpretación de la realidad. Inferir	95-100

		comportamientos o consecuencias de los fenómenos o problemas en estudio. Incluir más variables en dichos casos de estudio. b) Hace aportaciones a las actividades académicas desarrolladas. Pregunta integrando conocimientos de otras asignaturas o de casos anteriores de la misma asignatura. Presenta otros puntos de vista que complementan al presentado en la clase. Presenta fuentes de información adicionales (Internet, documentales), usa más bibliografía, consulta fuentes en un segundo idioma, etc. c) Propone y/o explica soluciones o procedimientos no vistos en clase (creatividad). Ante problemas o casos de estudio propone perspectivas diferentes, para abordarlos y sustentarlos correctamente. Aplica procedimientos aprendidos en otra asignatura o contexto para el problema que se está resolviendo. d) Introduce recursos y experiencias que promueven un pensamiento crítico; (por ejemplo el uso de las tecnologías de la información estableciendo previamente un criterio). Ante temas de una asignatura, introduce cuestionamientos de tipo ético, ecológico, histórico, político, económico, etc.; que deben tomarse en cuenta para comprender mejor, o a futuro dicho tema. Se apoya en foros, autores, bibliografía, documentales, etc. para sustentar su punto de vista. e) Incorpora conocimientos y actividades interdisciplinarias en su aprendizaje. En el desarrollo de los temas de la asignatura, incorpora conocimientos y actividades desarrollados en otras asignaturas para lograr la competencia. f) Realiza su trabajo de manera autónoma y autorregulada. Es capaz de organizar su tiempo y trabajar sin necesidad de una supervisión estrecha y/o coercitiva. Aprovecha la planeación de la asignatura presentada por el (la) profesor(a) (instrumentación didáctica) para presentar propuestas de mejora de la temática vista durante el curso. Realiza actividades de investigación para participar activamente durante el curso.	
	Notable	Cumple cuatro de los indicadores diferidos en desempeño excelente	85-94
	Bueno	Cumple tres de los indicadores diferidos en desempeño excelente	75-84
	Suficiente	Cumple dos de los indicadores diferidos en desempeño excelente	70-74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores diferidos en el desempeño excelente.	NA (no alcanzada)

MATRIZ DE EVALUACIÓN

Evidencia de aprendizaje	%	Indicador de alcance						Método de Evaluación			
		A	B	C	D	E	F	Instrumento	P	C	A
Prácticas	50	X	X			X	X	Rubrica	X	X	X
Examen	50	X		X		X	X	Cuestionario		X	
	100 %										

Competencia No.: 2

Descripción: Específica(s): Conoce las características principales de un lenguaje de programación que le permiten codificar algoritmos, compilar y ejecutar programas.

Temas y subtemas para desarrollar la	Actividades de aprendizaje	Actividades de Enseñanza	Desarrollo de competencias genéricas	Horas teórico-

competencia específica				práctica
Unidad # 2 Introducción a la Programación 2.1 Características del lenguaje de programación	El alumno tomará apuntes sobre las Características del lenguaje de programación.	El docente explicará las Características del lenguaje de programación.	Capacidad de análisis y síntesis. Habilidades de gestión de información. Habilidad para buscar y analizar información.	4 Horas teoría – 6 Horas prácticas 2 Semanas
2.2 Estructura básica de un programa.	El alumno tomará apuntes sobre la Estructura básica de un programa.	El docente explicará la Estructura básica de un programa.	Capacidad de análisis y síntesis. Habilidades de gestión de información. Habilidad para buscar y analizar información.	
2.3 Traducción de un programa: compilación, enlace de un programa, errores en tiempo de compilación.	El alumno atenderá la demostración y tomará apuntes, sobre la compilación, enlace de un programa, errores en tiempo de compilación.	El docente Mediante la técnica de Demostración – Ejecución explicará a los alumnos la compilación, enlace de un programa, errores en tiempo de compilación.	Capacidad de análisis y síntesis. Habilidades de gestión de información. Habilidad para buscar y analizar información.	
2.4 Ejecución de un programa.	El alumno atenderá la demostración y tomará apuntes, sobre la ejecución de un programa, realizará y entregará reporte de prácticas.	El docente Mediante la técnica de Demostración – Ejecución explicará a los alumnos la Ejecución de un programa. El docente solicitará al alumno la realización de prácticas correspondientes.	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	
2.5 Elementos del lenguaje: datos, literales y constantes, identificadores, variables, parámetros, operadores, entrada y salida de datos.	El alumno atenderá la demostración y tomará apuntes, sobre los elementos del lenguaje, realizará y entregará reporte de prácticas.	El docente Mediante la técnica de Demostración – Ejecución explicará a los alumnos los elementos del lenguaje. El docente solicitará al alumno la realización de prácticas correspondientes.	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	
2.6 Errores en tiempo de ejecución.	El alumno atenderá la demostración y tomará apuntes, sobre los errores en tiempo de ejecución, realizará y entregará reporte de prácticas. El alumno realizará examen de la competencia 2	El docente Mediante la técnica de Demostración – Ejecución explicará a los alumnos los errores en tiempo de ejecución. El docente solicitará al alumno la realización de prácticas correspondientes. El docente aplicará examen de la competencia 2	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	

CRITERIOS DE EVALUACIÓN

Indicadores de alcance	Valor del indicador

A. Se adapta a situaciones y contextos complejos: Trabajo en equipo.	Indica la ponderación de los criterios de valoración definidos en el punto anterior.
B. Hace aportaciones a las actividades académicas desarrolladas: Realiza la investigación de los conceptos señalados, utilizando bibliografía en un segundo idioma (inglés).	
C. Propone y/o explica soluciones o procedimientos no vistos en clase (Creatividad): Propone actividades para la generación de ideas de proyectos orientados al espíritu emprendedor.	
D. Introduce recursos y experiencias que promueven un pensamiento Crítico: Se apoya en las sugerencias y observaciones de sus compañeros de clase, después de su exposición de la idea emprendedora	
E. Incorpora conocimientos y actividades interdisciplinarias en su aprendizaje: Incorpora temas relacionados con su carrera para iniciar con su proyecto de la materia	
F. Realiza su trabajo de manera autónoma y autorregulada: Capacidad de organizar la planeación de los temas, utilizando las estrategias plasmadas en la instrumentación didáctica, anticipándose a las actividades de aprendizaje señaladas participando en trabajos colaborativos para fortalecer a la ciudad	

NIVELES DE DESEMPEÑO:

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple al menos cinco de los siguientes indicadores a) Se adapta a situaciones y contextos complejos. Puede trabajar en equipo, reflejar sus conocimientos en la interpretación de la realidad. Inferir comportamientos o consecuencias de los fenómenos o problemas en estudio. Incluir más variables en dichos casos de estudio. b) Hace aportaciones a las actividades académicas desarrolladas. Pregunta integrando conocimientos de otras asignaturas o de casos anteriores de la misma asignatura. Presenta otros puntos de vista que complementan al presentado en la clase. Presenta fuentes de información adicionales (Internet, documentales), usa más bibliografía, consulta fuentes en un segundo idioma, etc. c) Propone y/o explica soluciones o procedimientos no vistos en clase (creatividad). Ante problemas o casos de estudio propone perspectivas diferentes, para abordarlos y sustentarlos correctamente. Aplica procedimientos aprendidos en otra asignatura o contexto para el problema que se está resolviendo. d) Introduce recursos y experiencias que promueven un pensamiento crítico; (por ejemplo el uso de las tecnologías de la información estableciendo previamente un criterio). Ante temas de una asignatura, introduce cuestionamientos de tipo ético, ecológico, histórico, político, económico, etc.; que deben tomarse en cuenta para comprender mejor, o a futuro dicho tema. Se apoya en foros, autores, bibliografía, documentales, etc. para sustentar su punto de vista. e) Incorpora conocimientos y actividades interdisciplinarias en su aprendizaje. En el desarrollo de los temas de la asignatura, incorpora conocimientos y actividades desarrollados en otras asignaturas para lograr la competencia. f) Realiza su trabajo de manera autónoma y autorregulada. Es capaz de organizar su tiempo y trabajar sin necesidad de una supervisión estrecha y/o coercitiva. Aprovecha la planeación de la asignatura presentada por el (la) profesor(a) (instrumentación didáctica) para presentar propuestas de mejora de la temática vista durante el curso. Realiza actividades de investigación para participar activamente durante el curso.	95-100
	Notable	Cumple cuatro de los indicadores diferidos en desempeño excelente	85-94
	Bueno	Cumple tres de los indicadores diferidos en desempeño excelente	75-84

	Suficiente	Cumple dos de los indicadores diferidos en desempeño excelente	70-74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores diferidos en el desempeño excelente.	NA (no alcanzada)

MATRIZ DE EVALUACIÓN

Evidencia de aprendizaje	%	Indicador de alcance						Método de Evaluación			
		A	B	C	D	E	F	Instrumento	P	C	A
Prácticas	50	X	X			X	X	Rubrica	X	X	X
Examen	50	X		X		X	X	Cuestionario		X	
	100 %										

Competencia No.: 3

Descripción: Específica(s): Construye programas utilizando estructuras condicionales y repetitivas para aumentar su funcionalidad

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje	Actividades de Enseñanza	Desarrollo de competencias genéricas	Horas teórico-práctica
Unidad # 3 Control de flujo 3.1 Estructuras secuenciales.	El alumno Participará activamente. El alumno tomará apuntes acerca de la presentación del tema sobre la estructura de secuenciales. El alumno elaborará las prácticas correspondientes. Cumpliendo con las reglas de sintaxis del lenguaje de programación y entregará su reporte de prácticas.	El docente por medio de Técnica Expositiva explicará los temas sobre estructuras secuenciales. El docente mediante la técnica de Demostración – Ejecución se explicará a los alumnos el desarrollo de programas utilizando estructuras selectivas. El docente solicitara la realización de prácticas.	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	6 Horas teoría – 9 Horas prácticas 3 Semanas
3.2 Estructuras selectivas: simple, doble y múltiple.	El alumno tomará apuntes acerca de la presentación del tema sobre la estructura selectivas. El alumno elaborará las prácticas correspondientes. Cumpliendo con las reglas de sintaxis del lenguaje de programación y entregará su reporte de prácticas. El docente por medio de Técnica Expositiva explicará los temas sobre estructuras selectivas.	El docente mediante la técnica de Demostración – Ejecución se explicará a los alumnos el desarrollo de programas utilizando estructuras selectivas. El docente solicitará la realización de prácticas.	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	
3.3 Estructuras iterativas: repetir	El alumno tomará apuntes acerca de la presentación del tema sobre la estructura Iterativas. El alumno	El docente por medio de Técnica Expositiva explicará los temas sobre estructuras iterativas. El	Capacidad de análisis y síntesis. Capacidad de	

mientras, hasta, desde	elaborará las prácticas correspondientes. Cumpliendo con las reglas de sintaxis del lenguaje de programación y entregará su reporte de prácticas.	docente mediante la técnica de Demostración – Ejecución se explicará a los alumnos el desarrollo de programas utilizando estructuras iterativas. El docente solicitará la realización de prácticas.	aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	
3.4 Diseño e implementación de funciones	El alumno tomará apuntes acerca de la presentación del tema diseño e implementación de funciones. El alumno elaborará las prácticas correspondientes. Cumpliendo con las reglas de sintaxis del lenguaje de programación y entregará su reporte de prácticas. El alumno realizará examen de la competencia 3	El docente por medio de Técnica Expositiva explicará los temas Diseño e implementación de funciones. El docente mediante la técnica de Demostración – Ejecución se explicará a los alumnos el desarrollo de programas utilizando funciones. El docente solicitara la realización de prácticas. El docente, aplicará examen de la competencia 3	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	

CRITERIOS DE EVALUACIÓN

Indicadores de alcance	Valor del indicador
A. Se adapta a situaciones y contextos complejos: Trabajo en equipo.	Indica la ponderación de los criterios de valoración definidos en el punto anterior.
B. Hace aportaciones a las actividades académicas desarrolladas: Realiza la investigación de los conceptos señalados, utilizando bibliografía en un segundo idioma (inglés).	
C. Propone y/o explica soluciones o procedimientos no vistos en clase (Creatividad): Propone actividades para la generación de ideas de proyectos orientados al espíritu emprendedor.	
D. Introduce recursos y experiencias que promueven un pensamiento Crítico: Se apoya en las sugerencias y observaciones de sus compañeros de clase, después de su exposición de la idea emprendedora	
E. Incorpora conocimientos y actividades interdisciplinarias en su aprendizaje: Incorpora temas relacionados con su carrera para iniciar con su proyecto de la materia	
F. Realiza su trabajo de manera autónoma y autorregulada: Capacidad de organizar la planeación de los temas, utilizando las estrategias plasmadas en la instrumentación didáctica, anticipándose a las actividades de aprendizaje señaladas participando en trabajos colaborativos para fortalecer a la ciudad	

NIVELES DE DESEMPEÑO:

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple al menos cinco de los siguientes indicadores a) Se adapta a situaciones y contextos complejos. Puede trabajar en equipo, reflejar sus conocimientos en la interpretación de la realidad. Inferir comportamientos o consecuencias de los fenómenos o problemas en estudio. Incluir más variables en dichos casos de estudio. b) Hace aportaciones a las actividades académicas desarrolladas. Pregunta	95-100

		integrando conocimientos de otras asignaturas o de casos anteriores de la misma asignatura. Presenta otros puntos de vista que complementan al presentado en la clase. Presenta fuentes de información adicionales (Internet, documentales), usa más bibliografía, consulta fuentes en un segundo idioma, etc. c) Propone y/o explica soluciones o procedimientos no vistos en clase (creatividad). Ante problemas o casos de estudio propone perspectivas diferentes, para abordarlos y sustentarlos correctamente. Aplica procedimientos aprendidos en otra asignatura o contexto para el problema que se está resolviendo. d) Introduce recursos y experiencias que promueven un pensamiento crítico; (por ejemplo el uso de las tecnologías de la información estableciendo previamente un criterio). Ante temas de una asignatura, introduce cuestionamientos de tipo ético, ecológico, histórico, político, económico, etc.; que deben tomarse en cuenta para comprender mejor, o a futuro dicho tema. Se apoya en foros, autores, bibliografía, documentales, etc. para sustentar su punto de vista. e) Incorpora conocimientos y actividades interdisciplinarias en su aprendizaje. En el desarrollo de los temas de la asignatura, incorpora conocimientos y actividades desarrollados en otras asignaturas para lograr la competencia. f) Realiza su trabajo de manera autónoma y autorregulada. Es capaz de organizar su tiempo y trabajar sin necesidad de una supervisión estrecha y/o coercitiva. Aprovecha la planeación de la asignatura presentada por el (la) profesor(a) (instrumentación didáctica) para presentar propuestas de mejora de la temática vista durante el curso. Realiza actividades de investigación para participar activamente durante el curso.	
	Notable	Cumple cuatro de los indicadores diferidos en desempeño excelente	85-94
	Bueno	Cumple tres de los indicadores diferidos en desempeño excelente	75-84
	Suficiente	Cumple dos de los indicadores diferidos en desempeño excelente	70-74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores diferidos en el desempeño excelente.	NA (no alcanzada)

MATRIZ DE EVALUACIÓN

Evidencia de aprendizaje	%	Indicador de alcance						Método de Evaluación			
		A	B	C	D	E	F	Instrumento	P	C	A
Prácticas	50	X	X			X	X	Rubrica	X	X	X
Examen	50	X		X		X	X	Cuestionario		X	
	100 %										

Competencia No.: 4

Descripción: Específica(s): Construye programas que utilicen arreglos unidimensionales y multidimensionales para solucionar problemas.

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje	Actividades de Enseñanza	Desarrollo de competencias genéricas	Horas teórico-práctica

Unidad # 4 Arreglos Estructuras y Uniones 4.1 Unidimensionales: conceptos básicos, operaciones y aplicaciones.	El alumno Participará activamente. El alumno tomará apuntes acerca de la presentación del tema sobre arreglos unidimensionales. El alumno elaborará las prácticas correspondientes. Cumpliendo con las reglas de sintaxis del lenguaje de programación y entregará su reporte de prácticas.	El docente por medio de Técnica Expositiva explicará los temas sobre arreglos unidimensionales. El docente mediante la técnica de Demostración – Ejecución se explicará a los alumnos el desarrollo de programas utilizando arreglos unidimensionales. El docente solicitará la realización de prácticas. Capacidad de análisis y síntesis.	Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	4 Horas teoría – 6 Horas prácticas 2 Semanas
4.2 Multidimensionales: conceptos básicos, operaciones y aplicaciones.	El alumno tomará apuntes acerca de la presentación del tema sobre arreglos multidimensionales. El alumno elaborará las prácticas correspondientes. Cumpliendo con las reglas de sintaxis del lenguaje de programación y entregará su reporte de prácticas. Capacidad de análisis y síntesis.	El docente por medio de Técnica Expositiva explicará los temas sobre arreglos Multidimensionales. El docente mediante la técnica de Demostración – Ejecución explicará a los alumnos el desarrollo de programas utilizando arreglos Multidimensionales. El docente solicitará la realización de prácticas.	Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	
4.3 Aplicaciones en programas	El alumno tomará apuntes acerca de la presentación del tema diseño de aplicaciones. El alumno elaborará las prácticas correspondientes. Cumpliendo con las reglas de sintaxis del lenguaje de programación y entregará su reporte de prácticas. El alumno realizará examen de la competencia 4	El docente por medio de Técnica Expositiva explicará el diseño de aplicaciones. El docente mediante la técnica de Demostración – Ejecución explicará a los alumnos el diseño de aplicaciones. El docente solicitará la realización de prácticas. El docente, aplicará examen de la competencia 4	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	

CRITERIOS DE EVALUACIÓN

Indicadores de alcance	Valor del indicador
A. Se adapta a situaciones y contextos complejos: Trabajo en equipo.	Indica la ponderación de los criterios de valoración definidos en el punto anterior.
B. Hace aportaciones a las actividades académicas desarrolladas: Realiza la investigación de los conceptos señalados, utilizando bibliografía en un segundo idioma (inglés).	
C. Propone y/o explica soluciones o procedimientos no vistos en clase (Creatividad): Propone actividades para la generación de ideas de proyectos orientados al espíritu emprendedor.	
D. Introduce recursos y experiencias que promueven un pensamiento Crítico: Se apoya en las sugerencias y observaciones de sus compañeros de clase, después de su exposición de la idea emprendedora	
E. Incorpora conocimientos y actividades interdisciplinarias en su aprendizaje: Incorpora temas relacionados con su carrera para iniciar con su proyecto de la materia	
F. Realiza su trabajo de manera autónoma y autorregulada: Capacidad de organizar la planeación de los temas, utilizando las estrategias plasmadas en la instrumentación didáctica, anticipándose a las actividades de aprendizaje señaladas participando en trabajos colaborativos para fortalecer a la ciudad	

NIVELES DE DESEMPEÑO:

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple al menos cinco de los siguientes indicadores a) Se adapta a situaciones y contextos complejos. Puede trabajar en equipo, reflejar sus conocimientos en la interpretación de la realidad. Inferir comportamientos o consecuencias de los fenómenos o problemas en estudio. Incluir más variables en dichos casos de estudio. b) Hace aportaciones a las actividades académicas desarrolladas. Pregunta integrando conocimientos de otras asignaturas o de casos anteriores de la misma asignatura. Presenta otros puntos de vista que complementan al presentado en la clase. Presenta fuentes de información adicionales (Internet, documentales), usa más bibliografía, consulta fuentes en un segundo idioma, etc. c) Propone y/o explica soluciones o procedimientos no vistos en clase (creatividad). Ante problemas o casos de estudio propone perspectivas diferentes, para abordarlos y sustentarlos correctamente. Aplica procedimientos aprendidos en otra asignatura o contexto para el problema que se está resolviendo. d) Introduce recursos y experiencias que promueven un pensamiento crítico; (por ejemplo el uso de las tecnologías de la información estableciendo previamente un criterio). Ante temas de una asignatura, introduce cuestionamientos de tipo ético, ecológico, histórico, político, económico, etc.; que deben tomarse en cuenta para comprender mejor, o a futuro dicho tema. Se apoya en foros, autores, bibliografía, documentales, etc. para sustentar su punto de vista. e) Incorpora conocimientos y actividades interdisciplinarias en su aprendizaje. En el desarrollo de los temas de la asignatura, incorpora conocimientos y actividades desarrollados en otras asignaturas para lograr la competencia. f) Realiza su trabajo de manera autónoma y autorregulada. Es capaz de organizar su tiempo y trabajar sin necesidad de una supervisión estrecha y/o coercitiva. Aprovecha la planeación de la asignatura presentada por el (la) profesor(a) (instrumentación didáctica) para presentar propuestas de mejora de la temática vista durante el curso. Realiza actividades de investigación para participar activamente durante el curso.	95-100
	Notable	Cumple cuatro de los indicadores diferidos en desempeño excelente	85-94
	Bueno	Cumple tres de los indicadores diferidos en desempeño excelente	75-84
	Suficiente	Cumple dos de los indicadores diferidos en desempeño excelente	70-74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores diferidos en el desempeño excelente.	NA (no alcanzada)

MATRIZ DE EVALUACIÓN

Evidencia de aprendizaje	%	Indicador de alcance						Método de Evaluación			
		A	B	C	D	E	F	Instrumento	P	C	A
Prácticas	50	X	X			X	X	Rubrica	X	X	X
Examen	50	X		X		X	X	Cuestionario		X	

	100 %										

Competencia No.: 5 Descripción: Específica(s): Aplica funciones y procedimientos en programas

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje	Actividades de Enseñanza	Desarrollo de competencias genéricas	Horas teórico-práctica
Unidad # 5 Módulos 5.1 Diferencia entre Funciones y Procedimientos.	El alumno Participará activamente. El alumno tomará apuntes acerca de la presentación del tema la Diferencia entre Funciones y Procedimientos.	El docente por medio de Técnica Expositiva explicará la diferencia entre Funciones y Procedimientos.	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	4 Horas teoría – 6 Horas prácticas 2 Semanas
5.2 Activación / Invocación de Funciones y Procedimientos	El alumno tomará apuntes acerca de la presentación del tema sobre la Activación / Invocación de Funciones y Procedimientos. El alumno elaborará las prácticas correspondientes. Cumpliendo con las reglas de sintaxis del lenguaje de programación y entregará su reporte de prácticas.	El docente por medio de Técnica Expositiva explicará los temas sobre la Activación / Invocación de Funciones y Procedimientos. El docente mediante la técnica de Demostración – Ejecución explicará a los alumnos el desarrollo de programas, activando e invocando funciones y procedimientos. El docente solicitará la realización de prácticas.	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	
5.3 Uso de Funciones y Procedimientos con Parámetros.	El alumno tomará apuntes acerca de la presentación del tema sobre el uso de funciones y procedimientos con parámetros. El alumno elaborará las prácticas correspondientes. Cumpliendo con las reglas de sintaxis del lenguaje de programación y entregará su reporte de prácticas. El alumno realizará examen de la competencia 5	El docente por medio de Técnica Expositiva explicará los temas sobre el uso de Funciones y Procedimientos con Parámetros. El docente mediante la técnica de Demostración – Ejecución explicará a los alumnos el desarrollo de programas, usando Funciones y Procedimientos con Parámetros. El docente solicitará la realización de prácticas. El docente, aplicará examen de la competencia 5	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	

CRITERIOS DE EVALUACIÓN

Indicadores de alcance	Valor del indicador
A. Se adapta a situaciones y contextos complejos: Trabajo en equipo.	Indica la ponderación de los criterios de
B. Hace aportaciones a las actividades académicas desarrolladas: Realiza la investigación de	

los conceptos señalados, utilizando bibliografía en un segundo idioma (inglés).	valoración definidos en el punto anterior.
C. Propone y/o explica soluciones o procedimientos no vistos en clase (Creatividad): Propone actividades para la generación de ideas de proyectos orientados al espíritu emprendedor.	
D. Introduce recursos y experiencias que promueven un pensamiento Crítico: Se apoya en las sugerencias y observaciones de sus compañeros de clase, después de su exposición de la idea emprendedora	
E. Incorpora conocimientos y actividades interdisciplinarias en su aprendizaje: Incorpora temas relacionados con su carrera para iniciar con su proyecto de la materia	
F. Realiza su trabajo de manera autónoma y autorregulada: Capacidad de organizar la planeación de los temas, utilizando las estrategias plasmadas en la instrumentación didáctica, anticipándose a las actividades de aprendizaje señaladas participando en trabajos colaborativos para fortalecer a la ciudad	

NIVELES DE DESEMPEÑO:

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple al menos cinco de los siguientes indicadores a) Se adapta a situaciones y contextos complejos. Puede trabajar en equipo, reflejar sus conocimientos en la interpretación de la realidad. Inferir comportamientos o consecuencias de los fenómenos o problemas en estudio. Incluir más variables en dichos casos de estudio. b) Hace aportaciones a las actividades académicas desarrolladas. Pregunta integrando conocimientos de otras asignaturas o de casos anteriores de la misma asignatura. Presenta otros puntos de vista que complementan al presentado en la clase. Presenta fuentes de información adicionales (Internet, documentales), usa más bibliografía, consulta fuentes en un segundo idioma, etc. c) Propone y/o explica soluciones o procedimientos no vistos en clase (creatividad). Ante problemas o casos de estudio propone perspectivas diferentes, para abordarlos y sustentarlos correctamente. Aplica procedimientos aprendidos en otra asignatura o contexto para el problema que se está resolviendo. d) Introduce recursos y experiencias que promueven un pensamiento crítico; (por ejemplo el uso de las tecnologías de la información estableciendo previamente un criterio). Ante temas de una asignatura, introduce cuestionamientos de tipo ético, ecológico, histórico, político, económico, etc.; que deben tomarse en cuenta para comprender mejor, o a futuro dicho tema. Se apoya en foros, autores, bibliografía, documentales, etc. para sustentar su punto de vista. e) Incorpora conocimientos y actividades interdisciplinarias en su aprendizaje. En el desarrollo de los temas de la asignatura, incorpora conocimientos y actividades desarrollados en otras asignaturas para lograr la competencia. f) Realiza su trabajo de manera autónoma y autorregulada. Es capaz de organizar su tiempo y trabajar sin necesidad de una supervisión estrecha y/o coercitiva. Aprovecha la planeación de la asignatura presentada por el (la) profesor(a) (instrumentación didáctica) para presentar propuestas de mejora de la temática vista durante el curso. Realiza actividades de investigación para participar activamente durante el curso.	95-100
	Notable	Cumple cuatro de los indicadores diferidos en desempeño excelente	85-94
	Bueno	Cumple tres de los indicadores diferidos en desempeño excelente	75-84
	Suficiente	Cumple dos de los indicadores diferidos en desempeño excelente	70-74

Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores diferidos en el desempeño excelente.	NA (no alcanzada)

MATRIZ DE EVALUACIÓN

Evidencia de aprendizaje	%	Indicador de alcance						Método de Evaluación			
		A	B	C	D	E	F	Instrumento	P	C	A
Prácticas	50	X	X			X	X	Rubrica	X	X	X
Examen	50	X		X		X	X	Cuestionario		X	
	100 %										

Competencia No.: 6 Descripción: Específica(s): Declara y aplica apuntadores como manera de manipular información en llamadas por valor y referencia.

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje	Actividades de Enseñanza	Desarrollo de competencias genéricas	Horas teórico-práctica
Unidad # 6 Punteros 6.1 Declaración e inicialización	El alumno Participará activamente. El alumno tomará apuntes acerca de la presentación del tema declaración e inicialización.	El docente por medio de Técnica Expositiva explicará el tema la declaración e inicialización .	Capacidad de análisis y síntesis. Habilidades de gestión de información. Habilidad para buscar y analizar información.	4 Horas teoría – 6 Horas prácticas 2 Semanas
6.2 Operadores de apuntador	El alumno tomará apuntes acerca de la presentación del tema operadores de apuntador.	El docente por medio de Técnica Expositiva explicará el tema operadores de apuntador.	Capacidad de análisis y síntesis. Habilidades de gestión de información. Habilidad para buscar y analizar información.	
6.3 Uso de apuntadores	El alumno tomará apuntes acerca de la presentación del tema uso de apuntadores. El alumno elaborará las prácticas correspondientes. Cumpliendo con las reglas de sintaxis del lenguaje de programación y entregará su reporte de prácticas. Capacidad de análisis y síntesis.	El docente por medio de Técnica Expositiva explicará los temas sobre Uso de apuntadores. El docente mediante la técnica de Demostración – Ejecución explicará a los alumnos el desarrollo de programas, usando apuntadores. El docente solicitará la realización de prácticas.	Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	

6.4 Llamadas por valor y referencia	El alumno tomará apuntes acerca de la presentación del tema llamadas por valor y referencia. El alumno elaborará las prácticas correspondientes. Cumpliendo con las reglas de sintaxis del lenguaje de programación y entregará su reporte de prácticas. El alumno realizará examen de la competencia 6	El docente por medio de Técnica Expositiva explicará los temas sobre llamadas por valor y referencia. El docente mediante la técnica de Demostración – Ejecución explicará a los alumnos el desarrollo de programas, usando llamadas por valor y referencia. El docente solicitará la realización de prácticas. El docente, aplicará examen de la competencia 6	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	

CRITERIOS DE EVALUACIÓN

Indicadores de alcance	Valor del indicador
A. Se adapta a situaciones y contextos complejos: Trabajo en equipo.	Indica la ponderación de los criterios de valoración definidos en el punto anterior.
B. Hace aportaciones a las actividades académicas desarrolladas: Realiza la investigación de los conceptos señalados, utilizando bibliografía en un segundo idioma (inglés).	
C. Propone y/o explica soluciones o procedimientos no vistos en clase (Creatividad): Propone actividades para la generación de ideas de proyectos orientados al espíritu emprendedor.	
D. Introduce recursos y experiencias que promueven un pensamiento Crítico: Se apoya en las sugerencias y observaciones de sus compañeros de clase, después de su exposición de la idea emprendedora	
E. Incorpora conocimientos y actividades interdisciplinarias en su aprendizaje: Incorpora temas relacionados con su carrera para iniciar con su proyecto de la materia	
F. Realiza su trabajo de manera autónoma y autorregulada: Capacidad de organizar la planeación de los temas, utilizando las estrategias plasmadas en la instrumentación didáctica, anticipándose a las actividades de aprendizaje señaladas participando en trabajos colaborativos para fortalecer a la ciudad	

NIVELES DE DESEMPEÑO:

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple al menos cinco de los siguientes indicadores a) Se adapta a situaciones y contextos complejos. Puede trabajar en equipo, reflejar sus conocimientos en la interpretación de la realidad. Inferir comportamientos o consecuencias de los fenómenos o problemas en estudio. Incluir más variables en dichos casos de estudio. b) Hace aportaciones a las actividades académicas desarrolladas. Pregunta integrando conocimientos de otras asignaturas o de casos anteriores de la misma asignatura. Presenta otros puntos de vista que complementan al presentado en la clase. Presenta fuentes de información adicionales (Internet, documentales), usa más bibliografía, consulta fuentes en un segundo idioma, etc. c) Propone y/o explica soluciones o procedimientos no vistos en clase (creatividad). Ante problemas o casos de estudio propone perspectivas diferentes, para abordarlos y sustentarlos correctamente. Aplica procedimientos aprendidos en otra asignatura o contexto para el problema que se está resolviendo.	95-100

		d) Introduce recursos y experiencias que promueven un pensamiento crítico; (por ejemplo el uso de las tecnologías de la información estableciendo previamente un criterio). Ante temas de una asignatura, introduce cuestionamientos de tipo ético, ecológico, histórico, político, económico, etc.; que deben tomarse en cuenta para comprender mejor, o a futuro dicho tema. Se apoya en foros, autores, bibliografía, documentales, etc. para sustentar su punto de vista. e) Incorpora conocimientos y actividades interdisciplinarias en su aprendizaje. En el desarrollo de los temas de la asignatura, incorpora conocimientos y actividades desarrollados en otras asignaturas para lograr la competencia. f) Realiza su trabajo de manera autónoma y autorregulada. Es capaz de organizar su tiempo y trabajar sin necesidad de una supervisión estrecha y/o coercitiva. Aprovecha la planeación de la asignatura presentada por el (la) profesor(a) (instrumentación didáctica) para presentar propuestas de mejora de la temática vista durante el curso. Realiza actividades de investigación para participar activamente durante el curso.	
	Notable	Cumple cuatro de los indicadores diferidos en desempeño excelente	85-94
	Bueno	Cumple tres de los indicadores diferidos en desempeño excelente	75-84
	Suficiente	Cumple dos de los indicadores diferidos en desempeño excelente	70-74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores diferidos en el desempeño excelente.	NA (no alcanzada)

MATRIZ DE EVALUACIÓN

Evidencia de aprendizaje	%	Indicador de alcance						Método de Evaluación			
		A	B	C	D	E	F	Instrumento	P	C	A
Prácticas	50	X	X			X	X	Rubrica	X	X	X
Examen	50	X		X		X	X	Cuestionario		X	
	100 %										

Competencia No.: 7

Descripción: Específica(s): Utiliza las librerías de manejos de archivos para crearlos, leerlos, escribirlos, buscar registros y cerrarlos, ya sean secuenciales o directos.

Temas y subtemas para desarrollar la competencia específica	Actividades de aprendizaje	Actividades de Enseñanza	Desarrollo de competencias genéricas	Horas teórico-práctica
Unidad # 7 Archivos 7.1 Creación,	El alumno tomará apuntes acerca de la presentación del tema la creación, lectura, escritura y cierre de archivos de acceso secuencial. El alumno	El docente por medio de Técnica Expositiva explicará los temas sobre la creación, lectura, escritura y cierre de archivos de acceso secuencial. El docente mediante la técnica de Demostración –	Capacidad de análisis y síntesis. Capacidad de aplicar los	4 Horas teoría – 6 Horas prácticas

lectura, escritura y cierre de archivos de acceso secuencial	elaborará las prácticas correspondientes. Cumpliendo con las reglas de sintaxis del lenguaje de programación y entregará su reporte de prácticas.	Ejecución explicará a los alumnos el desarrollo de programas usando archivos, desde la creación, lectura, escritura y cierre. Archivos de acceso secuencial. El docente solicitará la realización de prácticas.	conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	2 Semanas
7.2 Creación, lectura, escritura y cierre de archivos de acceso directo	El alumno tomará apuntes acerca de la presentación del tema la creación, lectura, escritura y cierre de archivos de acceso directo. El alumno elaborará las prácticas correspondientes. Cumpliendo con las reglas de sintaxis del lenguaje de programación y entregará su reporte de prácticas. El alumno realizará examen de la competencia 7	El docente por medio de Técnica Expositiva explicará los temas sobre la creación, lectura, escritura y cierre de archivos de acceso directo. El docente mediante la técnica de Demostración – Ejecución explicará a los alumnos el desarrollo de programas usando archivos, desde la creación, lectura, escritura y cierre. Archivos de acceso directo. El docente solicitará la realización de prácticas. El docente, aplicará examen de la competencia 7	Capacidad de análisis y síntesis. Capacidad de aplicar los conocimientos en la práctica. Habilidades en el uso de las tecnologías de la información y de la comunicación.	

CRITERIOS DE EVALUACIÓN

Indicadores de alcance	Valor del indicador
A. Se adapta a situaciones y contextos complejos: Trabajo en equipo.	Indica la ponderación de los criterios de valoración definidos en el punto anterior.
B. Hace aportaciones a las actividades académicas desarrolladas: Realiza la investigación de los conceptos señalados, utilizando bibliografía en un segundo idioma (inglés).	
C. Propone y/o explica soluciones o procedimientos no vistos en clase (Creatividad): Propone actividades para la generación de ideas de proyectos orientados al espíritu emprendedor.	
D. Introduce recursos y experiencias que promueven un pensamiento Crítico: Se apoya en las sugerencias y observaciones de sus compañeros de clase, después de su exposición de la idea emprendedora	
E. Incorpora conocimientos y actividades interdisciplinarias en su aprendizaje: Incorpora temas relacionados con su carrera para iniciar con su proyecto de la materia	
F. Realiza su trabajo de manera autónoma y autorregulada: Capacidad de organizar la planeación de los temas, utilizando las estrategias plasmadas en la instrumentación didáctica, anticipándose a las actividades de aprendizaje señaladas participando en trabajos colaborativos para fortalecer a la ciudad	

NIVELES DE DESEMPEÑO:

Desempeño	Nivel de desempeño	Indicadores de alcance	Valoración numérica
Competencia alcanzada	Excelente	Cumple al menos cinco de los siguientes indicadores a) Se adapta a situaciones y contextos complejos. Puede trabajar en equipo, reflejar sus conocimientos en la interpretación de la realidad. Inferir comportamientos o consecuencias de los fenómenos o problemas en estudio.	95-100

		Incluir más variables en dichos casos de estudio. b) Hace aportaciones a las actividades académicas desarrolladas. Pregunta integrando conocimientos de otras asignaturas o de casos anteriores de la misma asignatura. Presenta otros puntos de vista que complementan al presentado en la clase. Presenta fuentes de información adicionales (Internet, documentales), usa más bibliografía, consulta fuentes en un segundo idioma, etc. c) Propone y/o explica soluciones o procedimientos no vistos en clase (creatividad). Ante problemas o casos de estudio propone perspectivas diferentes, para abordarlos y sustentarlos correctamente. Aplica procedimientos aprendidos en otra asignatura o contexto para el problema que se está resolviendo. d) Introduce recursos y experiencias que promueven un pensamiento crítico; (por ejemplo el uso de las tecnologías de la información estableciendo previamente un criterio). Ante temas de una asignatura, introduce cuestionamientos de tipo ético, ecológico, histórico, político, económico, etc.; que deben tomarse en cuenta para comprender mejor, o a futuro dicho tema. Se apoya en foros, autores, bibliografía, documentales, etc. para sustentar su punto de vista. e) Incorpora conocimientos y actividades interdisciplinarias en su aprendizaje. En el desarrollo de los temas de la asignatura, incorpora conocimientos y actividades desarrollados en otras asignaturas para lograr la competencia. f) Realiza su trabajo de manera autónoma y autorregulada. Es capaz de organizar su tiempo y trabajar sin necesidad de una supervisión estrecha y/o coercitiva. Aprovecha la planeación de la asignatura presentada por el (la) profesor(a) (instrumentación didáctica) para presentar propuestas de mejora de la temática vista durante el curso. Realiza actividades de investigación para participar activamente durante el curso.	
	Notable	Cumple cuatro de los indicadores diferidos en desempeño excelente	85-94
	Bueno	Cumple tres de los indicadores diferidos en desempeño excelente	75-84
	Suficiente	Cumple dos de los indicadores diferidos en desempeño excelente	70-74
Competencia no alcanzada	Insuficiente	No se cumple con el 100% de evidencias conceptuales, procedimentales y actitudinales de los indicadores diferidos en el desempeño excelente.	NA (no alcanzada)

MATRIZ DE EVALUACIÓN

Evidencia de aprendizaje	%	Indicador de alcance						Método de Evaluación			
		A	B	C	D	E	F	Instrumento	P	C	A
Prácticas	50	X	X			X	X	Rubrica	X	X	X
Examen	50	X		X		X	X	Cuestionario		X	
	100 %										

5. FUENTES DE INFORMACIÓN Y APOYOS DIDÁCTICOS

Fuentes de Información	Apoyos didácticos

1. Luis Joyanes Aguilar, Fundamentos de Programación, Ed. Prentice Hall. 2. Jesús J. García Molina Introducción a la programación un Enfoque Algorítmico, Ed. Paraninfo. 3. Leobardo López Román, Metodología de la Programación Orientada a Objetos, Ed. Alfaomega. 4. Cairo Osvaldo, Metodología de la Programación, Ed. Alfaomega. 5. Deitel y Deitel. Como Programar en C++ quinta Edición. Prentice Hall. 6. Deitel y Deitel. Como Programar en C# quinta Edición. Prentice Hall. 7. Deitel y Deitel. Java como programar. Séptima edición. Prentice Hall. 8. Joyanes Aguilar, Luis Fernández, Azuela Matilde, Rodríguez Baena Luis, Fundamentos de Programación Libro de Problemas Algoritmos Estructura de Datos y Objetos. 2a. edición Ed. Mc. Graw Hill 9. Luis Joyanes Aguilar. Programación en JAVA 2 1ª Edición. Mc Graw Hill. 10. Martín Flowler Kendall Scott. UML Gota a Gota. Addison Wesley. 11. Ramírez Felipe, Introducción a la Programación, Algoritmos y su Implementación En Vb.Net C# Java y C++, 2a. edición, Alfa Omega. 12. Jean-Paul Tremblay, Richar B. Bunt. Introducción a la Ciencia de Las Computadoras. Enfoque Algorítmico. McGraw Hill. 13. Bjarne Storstrup. Lenguaje de Programación C/C++. 14. Cairo Battistutti Osvaldo, Metodología de la Programación, Algoritmos Diagramas de Flujo y Programas, 3a. edición, Alfa Omega. 15. Flores Cueto, Juan José, Método de las 6'D UML – Pseudocódigo – Java Enfoque Algorítmico, Serie Textos Universitarios Facultad de Ingeniería y Arquitectura, ed. Universidad de San Martín de Porres, (http://books.google.com/).	Pizarrón Proyector Computadora Plataforma Moodle

6. CALENDARIZACIÓN DE EVALUACIÓN EN SEMANAS

Semana	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
TP	ED		EF(1)		EF(2)			EF(3)		EF(4)		EF(5)		EF(6)		EF(7) ES
TR																
SD																

TP=tiempo planeado TR=tiempo real SD=seguimiento departamental

ED=evaluación diagnóstica EFn=evaluación formativa (competencia específica n) ES=evaluación sumativa

Fecha de elaboración: 8/Marzo/2021

LUDIVINA TORAL AGUILAR	ING. ELISA DANIA MOLINA PORTILLO
Nombre del profesor(a)	Nombre del Jefe(a) de Departamento Académico

ANEXO 4

1. **PROYECTO INTEGRADOR DE ALGORITMOS (DIAGRAMAS DE FLUJO)**
2. **PROYECTO INTEGRADOR DE INTRODUCCION A LA PROGRAMACION**
3. **PROYECTO INTEGRADOR DE CONTROL DE FLUJO**
4. **PROYECTO INTEGRADOR DE ARREGLOS, ESTRUCTURAS Y UNIONES**
5. **PROYECTO INTEGRADOR DE MODULOS**
6. **PROYECTO INTEGRADOR DE PUNTEROS**
7. **PROYECTO INTEGRADOR DE FICHEROS**

PROYECTO INTEGRADOR DE ALGORITMOS (DIAGRAMAS DE FLUJO)

TEMA 1

Domina los conceptos básicos de la programación para analizar problemas y representar su solución mediante algoritmos (diagrama de flujo).

CONTENIDO

1 Introducción 3

1.1 Planteamiento del Problema 3

1.2 Justificación 3

1.3 Objetivos 4

2 Marco Teórico 4

3 Metodología 6

4 Resultados 10

5 Conclusión 10

6 Rúbrica de Evaluación 11

1. INTRODUCCION

El desarrollo de algoritmos es un tema fundamental en el diseño de programas por lo cual el alumno debe tener buenas bases que le sirvan para poder desarrollar de manera fácil y rápida sus programas. Es muy importante dar a conocer los conceptos básicos para introducir al estudiante en la programación con la finalidad de obtener el fundamento conceptual para el análisis, diseño y desarrollo de algoritmos, así como los primeros pasos al mundo de la programación a través de ejercicios dirigidos. En este proyecto es fundamental propiciar el pensamiento lógico del estudiante.

1.1 Planteamiento del Problema

El diseño de soluciones mediante la computadora, requiere como en otras disciplinas una metodología que nos enseñe de manera gradual, la forma de llegar a estas soluciones. ¿Cuál es la metodología a seguir para la solución de problemas por medio de la computadora? ¿Qué conceptos básicos, simbología y herramientas se aplican para el diseño de algoritmos?. El mejor entendimiento y aplicación de estos temas en el diseño de algoritmos logran un mayor interés de aprendizaje en los estudiantes. La enseñanza- aprendizaje no debe ser solamente informativa si no también formativa, para que el estudiante desarrolle sus propias capacidades de aprendizaje.

1.2 Justificación

Este proyecto nos permite poner en práctica los conocimientos para diseñar soluciones a la medida, de problemas específicos que se nos presenten. Más aún, si estos involucran operaciones matemáticas complejas y/o repetitivas, o requieren del manejo de un volumen muy grande de datos. De igual forma brindar un aporte significativo para que los estudiantes gocen de una herramienta para el aprendizaje de diseño de algoritmos de la asignatura de Programación Básica.

1.3 Objetivos

1.3.1 Objetivo General

Aplica algoritmos para diseñar e implementar soluciones a problemáticas del entorno.

1.3.2 Objetivos Específicos

Comprende y aplica los conceptos básicos, nomenclatura y herramientas para el diseño de algoritmos orientado a la resolución de problemas.
Desarrolla diagrama de flujo.
Aplica la metodología para la solución de problemas mediante la computadora.

2. MARCO TEÓRICO

2.1 Conceptos Básicos

2.1.1 Definición de Algoritmo

La palabra algoritmo se deriva de la traducción al latín de la palabra árabe alkhowarizmi, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX. Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.

2.1.2 Diagrama de Flujo

Es una representación gráfica de un algoritmo que se caracteriza por usar símbolos gráficos, para expresar de forma sencilla y clara el orden lógico en el que se realizan las acciones de un algoritmo.

2.1.3 Simbología de Diagrama de Flujo

Los símbolos utilizados han sido normalizados por el instituto norteamericano de normalización (ANSI) figura 1.1.

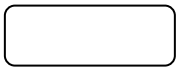
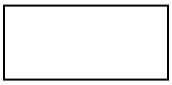
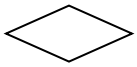
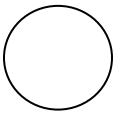
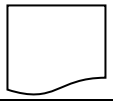
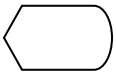
 ***Óvalo*****:** Inicio y término (Abre y/o cierra el diagrama).	 ***Rectángulo:*** Actividad (Representa la ejecución de una o más actividades o procedimientos).
 ***Rombo:*** Decisión (Formula una pregunta o cuestión).	 ***Círculo:*** Conector (Representa el enlace de actividades con otra dentro de un procedimiento).
Conector de pagina: Representa la continuidad en otra pagina	 ***salida:*** Representa la salida por impresora.
 ***Salida:*** representa salida a monitor	***Entrada***: reprenta la entrada de datos por medio del teclado.

Fig 1.1 Simbología de Diagramas de Flujo.

2.1.4 Características de un Buen Diseño de Algoritmos

- Un algoritmo debe ser preciso e indicar el orden de realización de cada paso.
- Un algoritmo debe estar definido. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- Un algoritmo debe ser finito.

2.1.5 Normas o Reglas para la Creación de Diagramas de Flujo

1. Los diagramas de flujo se escriben de arriba abajo y de izquierda a derecha.

2. Todo símbolo (excepto las líneas de flujo) llevará en su interior información que indique su función exacta y unívoca.

3. Un elemento del diagrama no puede tener más de una salida si no es un elemento de decisión. Existen excepciones que veremos en su debido momento.

4. Las líneas de flujo no pueden cruzarse.

3. METODOLOGIA

3.1 Diseño de Diagrama de Flujo

Ejercicio 1:

Definición del problema:

Desarrolle un algoritmo que le permita calcular el área de un triángulo en función de las longitudes de sus lados previamente leídos desde el teclado.

Análisis del problema:

Datos de Entrada: Aquí van todos los datos que necesito saber para resolver el problema y que desconozco porque el enunciado no me los da, y necesito ingresarlos por medio del dispositivo de entrada el teclado, el dato ingresado por teclado se le conoce como dato de entrada.
El dato ingresado por teclado necesitamos identificarlo mediante una variable.

Ejemplo:
Datos de Entrada:
b.- es la base del triángulo, numérica
h.- es la altura del triángulo, numérica

Proceso: Aquí van todos los cálculos matemáticos. Respetando la prioridad de los operadores aritmético.
Operadores del mismo nivel de prioridad que se encuentren en una misma expresión matemática se empieza a resolver de izquierda a derecha.

Proceso:
a=b*h/2

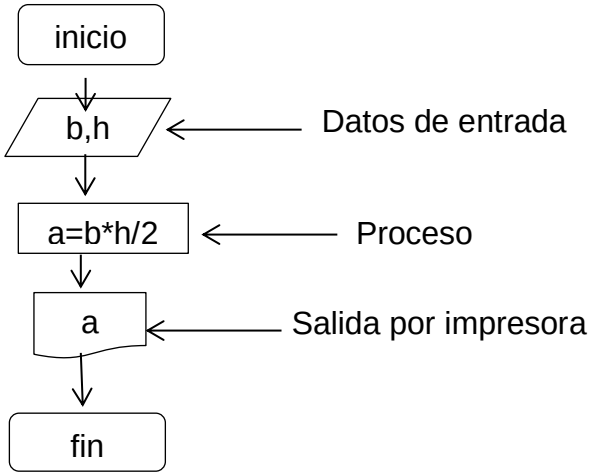
La multiplicación y la división, tienen el mismo nivel de prioridad, de izquierda a derecha primero multiplica y después divide.

La variable a, es una variable de trabajo, y recibe su valor por medio del resultado de una operación matemáticas.

Salida: Aquí va todo lo que queremos mandar a imprimir mediante un dispositivo de salida.

Salida:
a

SOLUCION:



Ejercicio 2:

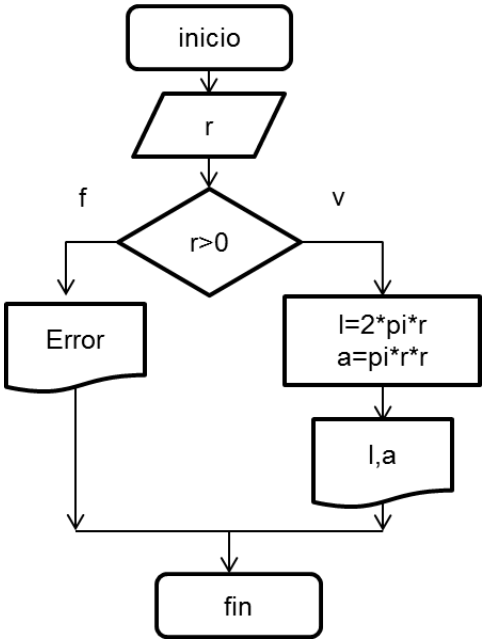
Escribir un algoritmo que lea tres números y si el primero es positivo calcule el producto de los tres números, y en otro caso calcule la suma.

El algoritmo en pseudocódigo es:

```
variables

entero: Numero1, Numero2, Numero3, Producto, Suma

inicio

        leer(Numero1, Numero2, Numero3)

        si (Numero1 > 0) entonces

                Producto=Numero1* Numero2 * Numero3

                imprime('El producto de los números es', Producto)

        sino

                Suma=Numero1+ Numero2 + Numero3

                imprime('La suma de los números es', Suma)

        fin si

fin
```

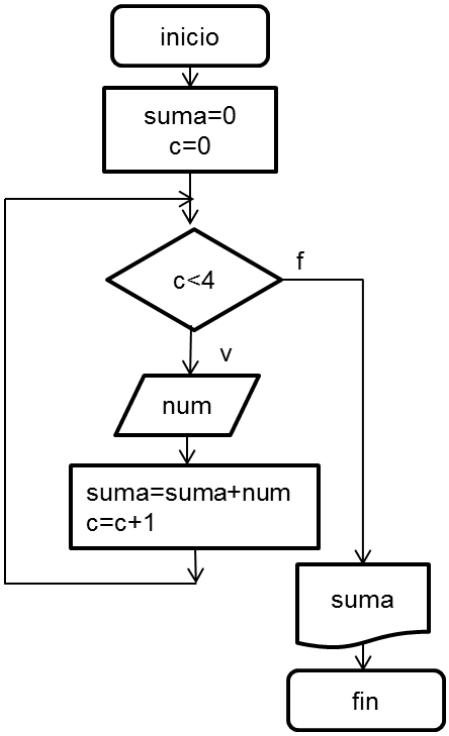
Ejercicio 3:

Escribir un algoritmo (Diagrama de Flujo) que lea el radio de un círculo, y calcule su perímetro y su área.
Se declaran las variables reales r, longitud y área, así como la constante pi=3.14 variables real r, longitud, área:



Ejercicio 4:

Escribir un algoritmo que lea cuatro números enteros del teclado y calcule su suma.



4. RESULTADOS

El alumno, comprendió y aplicó los conceptos básicos, nomenclatura y herramientas para el diseño de algoritmos orientado a la resolución de problemas. Desarrolla diagrama de flujo y Aplica la metodología para la solución de problemas mediante la computadora.

5. CONCLUSION

A finalizar el proyecto nos ayuda a tener un mejor entendimiento del diseño del algoritmo (diagrama de flujo), obteniendo el fundamento conceptual para el análisis, diseño y desarrollo de algoritmos.

6. RÚBRICA DE EVALUACIÓN

INDICADORES	CUMPLE	NO CUMPLE	PTOS	CALIFICACIÓN
Se realizó análisis del problema.			10	
Las variables tienen nombres significativos.			10	
El Diseño del Algoritmo (diagrama de flujo) cumple con las características de un buen algoritmo:			10	
Tiene un punto particular de inicio.			10	
Es definido, no permite dobles interpretaciones.			10	
Es general, es decir, soportar la mayoría de las variantes que se puedan presentar en la definición del problema.			10	
Es finito en tamaño y tiempo de ejecución			10	
El Diagramas de Flujo, cumple con las reglas de un buen diseño			10	
Usa solamente líneas de flujos horizontales y/o verticales.			10	
Evita el cruce de líneas utilizando los conectores.			10	
Usa conectores solo cuando sea necesario.			10	
No deja líneas de flujo sin conectar.			10	

PROYECTO INTEGRADOR DE INTRODUCCION A LA PROGRAMACION

TEMA 2

Conoce las características principales de un lenguaje de programación (C++) que le permiten codificar algoritmos, compilar y ejecutar programas.

CONTENIDO

1 Introducción 3

1.1 Planteamiento del Problema 3

1.2 Justificación 3

1.3 Objetivos 4

2 Marco Teórico 4

3 Metodología 7

4 Resultados 9

5 Conclusión 9

6 Rúbrica de Evaluación 10

1. INTRODUCCION

El propósito de este proyecto es dar a conocer los conceptos básicos del lenguaje de programación e introducir al estudiante en la programación con la finalidad de obtener el fundamento conceptual para el análisis, diseño e implementación de algoritmos. La implementación de los algoritmos se debe realizar en un lenguaje de programación para que una computadora pueda comprender las instrucciones que el algoritmo modela para así poder ejecutarlas y lograr el resultado esperado. Lo fundamental de este proyecto es conocer y aplicar el lenguaje de programación C++ para la resolución de problemas.

1.1 Planteamiento del Problema

La implementación de los algoritmos se debe realizar en un lenguaje de programación para que una computadora pueda comprender las instrucciones que el algoritmo modela para así poder ejecutarlas y lograr el resultado esperado.
¿Cuál es la estructura de un programa en c++?, ¿Cuáles son los conceptos básicos, del lenguaje de programación C++?, ¿Cuáles son los tipos de datos, constantes, variables, identificadores, operadores y salida de datos propios del lenguaje de programación C++?. Implementar algoritmos al lenguaje de programación C++, logrando así un mayor interés de aprendizaje en los estudiantes. La enseñanza-aprendizaje no debe ser solamente informativa si no también formativa, para que el estudiante desarrolle sus propias capacidades de aprendizaje.

1.2 Justificación

Este proyecto tiene la finalidad de propiciar que el estudiante adquiera los conocimientos necesarios para diseñar e implementar soluciones en el lenguaje de programación C++ y resolver problemas del contexto. De igual forma brindar un aporte significativo para que los estudiantes gocen de una herramienta para el aprendizaje del lenguaje de programación C++ de la asignatura de Programación Básica.

1.3 Objetivos

1.3.1 Objetivo General

Conoce y aplica un lenguaje de programación para la resolución de problemas.

1.3.2 Objetivos Específicos

Identificar las características del lenguaje de programación C++.
Identificar tipos de datos, literales, constantes, variables, identificadores, operadores y salida de datos propios del lenguaje de programación C++.
Aplicar las herramientas de programación, retomando un algoritmo planteado en el tema anterior, para su codificación, compilación, enlace, ejecución e identificación de errores.
Analiza un programa completo donde identifica los elementos básicos (estructura y sintaxis) del lenguaje C++-

2. MARCO TEÓRICO

2.1 Conceptos Básicos

2.1.1 Estructura Básica de un Programa

La estructura típica completa de un programa C++ se muestra en la figura 2.1. Un ejemplo de un programa sencillo en C++.

```
1) // El programa imprime "Bienvenido a la programación C++"
2) #include <iostream>
3) using namespace std;
4) int main()
5) {
6)    cout << "Bienvenido a la programación C++\n";
7)    return 0;
8) }
```

Fig 2.1 Estructura Básica de un Programa

La primera línea es un *comentario*, identificado por dobles barras inclinadas (//). Los comentarios se incluyen en programas que proporcionan explicaciones a los lectores de los mismos. Son ignorados por el compilador. Para comentarios de varias líneas Los comentarios comienzan con la secuencia /* y terminan con la secuencia */.

La directiva #include de la segunda línea es necesaria para que el programa tenga salida. Se refiere a un archivo externo denominado iostream en el que se proporciona la información relativa al objeto cout. Obsérvese que los ángulos < y > no son parte del nombre del archivo; se utilizan para indicar que el archivo es un archivo de la biblioteca estándar C++.

La tercera línea es la directiva using namespace std**;** indica que todas las sentencias del programa que vienen a continuación están dentro del espacio de nombres std. Diversos componentes de programa, tales como cout, están declarados dentro de este espacio de nombres.

La cuarta línea contiene la cabecera de la función main(), obligatoria en cada programa de C++ contiene una función main() que es la función principal y que es el primer punto de entrada al programa.

La quinta y octava línea contienen sólo las llaves { y } que encierran el cuerpo de la función main() y son necesarias en todos los programas C++.

La sexta línea contiene la sentencia cout << "Bienvenido a la programación C++\n"; que indica al sistema que envía el mensaje "Bienvenido a la programación en C++\n" al objeto cout. Este objeto es el *flujo estándar de salida* que normalmente representa la pantalla de presentación de la computadora (el nombre cout es una abreviatura de *«console output»* —salida a consolas).

La salida será Bienvenido a la programación C++.

El símbolo '\n' es el símbolo de *nueva línea*. Poniendo este símbolo al final de la cadena entre comillas, indica al sistema que comience una nueva línea después de imprimir los caracteres precedentes, terminando, por consiguiente, la línea actual.

La séptima línea contiene la sentencia return 0. Esta sentencia termina la ejecución del programa y devuelve el control al sistema operativo de la computadora. El número 0 se utiliza para señalar que el programa ha terminado correctamente (*con éxito*).

La sentencia de salida de la sexta línea incluye diversos símbolos típicos de C++. El símbolo << se denomina *operador de salida* u *operador de inserción*. Inserta el mensaje en el flujo de salida. El símbolo \n incluido al final del mensaje significa *carácter de nueva línea*. Siempre que aparece en un mensaje de salida, se termina la línea actual de salida y comienza una nueva línea.
Obsérvese el punto y coma (;) al final de la sexta y séptima línea. C++ requiere que cada sentencia termine con un punto y coma. No es necesario que esté al final de una línea. Se pueden poner varias sentencias en la misma línea y se puede hacer que una sentencia se extienda sobre varias líneas.

Entrada (cin)
El archivo iostream define cin como un objeto que representa el flujo. Para la entrada, cin utiliza el operador >> para extraer caracteres del flujo de entrada y por ello se denomina *operador de extracción*. Al igual que cout, es un objeto inteligente. Convierte la entrada, una serie de caracteres escritos desde el teclado, en un formato aceptable a la variable donde se almacena la información. Si no se redirige explícitamente cin, la entrada procede del teclado variable.

Declaración de una Variable
Una *declaración* de una variable es una sentencia que proporciona información de la variable al compilador C++. Su sintaxis es:

tipo <variable>;

tipo.- *es el nombre de un tipo de dato conocido por C++*
variable.- *es un identificador (nombre) válido en C++.*

3. METODOLOGIA

3.1 Diseño del Programa

Definición del problema:

Desarrolle un programa que le permita calcular e imprimir la suma de 2 números enteros.
Realizar el análisis, el algoritmo (diagrama de flujo), el programa en C++ y explicar cada línea del programa.

Análisis del problema:

Datos de Entrada: Aquí van todos los datos que necesito saber para resolver el problema y que desconozco porque el enunciado no me los da, y necesito ingresarlos por medio del dispositivo de entrada el teclado, el dato ingresado por teclado se le conoce como dato de entrada.
El dato ingresado por teclado necesitamos identificarlo mediante una variable.

Ejemplo:
Datos de Entrada:
a.- es la base del triángulo, numérica
b.- es la altura del triángulo, numérica

Proceso: Aquí van todos los cálculos matemáticos. Respetando la prioridad de los operadores aritmético.

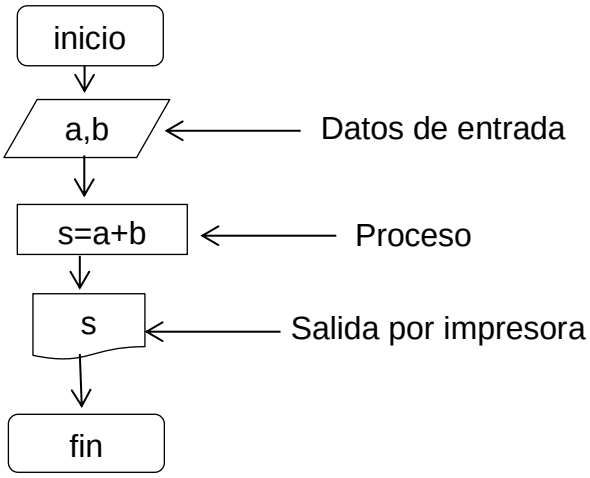
Proceso:
s=a+b

La variable s, es una variable de trabajo, y recibe su valor por medio del resultado de una operación matemáticas.

Salida: Aquí va todo lo que queremos mandar a imprimir mediante un dispositivo de salida.

Salida:
s

SOLUCION:




```
#include<iostream>
using namespace std;
int main()
{
int a,b,s;
cout<<"ingrese el valor de a: \n ";
cin>>a;
cout<<"ingrese el valor de b: "<<endl;
cin>>b;
s=a+b;
cout<<a<<"+"<<b<<"="<<s<<endl;
system("pause");
return 0;
}
```

<iostream> esta biblioteca permite el flujo de entrada y salida de datos.

Declaración de variables, primero va el tipo de dato y después el nombre de la variable.

Flujo de entrada, cin >>

El endl y \n, significan lo mismo, línea nueva, la diferencia es que la \n va siempre entre " ".

Flujo de salida, cout<<, imprime textos entre " ", imprime valor de la variable, va sin comillas.

System("pause"), mantiene en pausa la pantalla de resultados.

return 0, devuelve el valor 0 a la función main para finalizar correctamente.

Fin de main.

4. RESULTADOS

El alumno aprendió a Identificar las características del lenguaje de programación C++. Identifica tipos de datos, literales, constantes, variables, identificadores, operadores y salida de datos propios del lenguaje de programación C++. Aplica las herramientas de programación, retomando un algoritmo planteado en el tema anterior, para su codificación, compilación, enlace, ejecución e identificación de errores. Analiza un programa completo donde identifica los elementos básicos (estructura y sintaxis) del lenguaje C++-

5. CONCLUSION

A finalizar el proyecto nos ayuda a tener un mejor entendimiento de la implementación de los algoritmos al lenguaje de programación en C++, cumpliendo con la estructura y sintaxis correcta propia del lenguaje de programación. Aportando un aprendizaje significativo a los estudiantes para la resolución de problemas aplicando el lenguaje de programación C++ de la asignatura de Programación Básica.

6 RÚBRICA DE EVALUACION

INDICADORES	CUMPLE	NO CUMPLE	PTOS	CALIFICACIÓN
El diseño del programa cumple con la estructura y sintaxis del lenguaje de programación C++.			**10**	
Los nombres de variables son significativos			**10**	
Cumple con razonamiento lógico			**10**	
La Estructuración del código es correcta			**10**	
El programa funciona correctamente			**10**	

PROYECTO INTEGRADOR DE CONTROL DE FLUJO

TEMA 3

Construye programas utilizando estructuras condicionales y repetitivas para aumentar su funcionalidad

CONTENIDO

1 Introducción 3

1.1 Planteamiento del Problema 3

1.2 Justificación 3

1.3 Objetivos 4

2 Marco Teórico 4

3 Metodología 9

4 Resultados 11

5 Conclusión 11

6 Rúbrica de Evaluación 12

1. INTRODUCCION

Las **estructuras de control** controlan el flujo de ejecución de un programa o función. Las estructuras de control permiten combinar instrucciones o sentencias individuales en una simple unidad lógica con un punto de entrada y un punto de salida.
Las instrucciones o sentencias se organizan en tres tipos de estructuras de control que sirven para controlar el flujo de la ejecución: *secuencia*, *selección* (*decisión*) y *repetición*. El propósito de este proyecto es dar a conocer la sintaxis y uso de estas estructuras en el lenguaje de programación C++ y aplicar la combinación de ellas para el desarrollo de programas más complejos.

1.1 Planteamiento del Problema

La implementación de las estructuras de control, en la solución de problemas más complejos es muy importante saber seleccionar y aplicar la estructura correcta, para obtener el resultado esperado. ¿Cómo utilizamos las estructuras secuencial, selectiva e iterativa en el diseño de programas. En este proyecto el alumno conocerá las diferentes estructuras como componentes básicos de los programas y aplicará la combinación correcta de ellas, para un buen desarrollo de programas más complejos. Logrando así un mayor interés de aprendizaje en los estudiantes. La enseñanza-aprendizaje no debe ser solamente informativa si no también formativa, para que el estudiante desarrolle sus propias capacidades de aprendizaje.

1.2 Justificación

Este proyecto tiene la finalidad de propiciar que el estudiante adquiera los conocimientos necesarios para implementar las diferentes estructuras de control como componentes básicos de los programas y aplicará la combinación de ellas para el desarrollo de programas más complejos. Utilizando la sintaxis correcta propias del lenguaje de programación C++. De igual forma brindar un aporte significativo para que los estudiantes gocen de una herramienta para el desarrollo de programas en el lenguaje C++ de la asignatura de Programación Básica.

1.3 Objetivos

1.3.1 Objetivo General

Conoce y aplica las estructuras condicionales y repetitivas de un lenguaje de programación para resolver problemas reales.

1.3.2 Objetivos Específicos

Identifica la estructura correcta aplicar en el desarrollo del programa.
Aplica la sintaxis correcta del lenguaje de programación C++.
Conoce el funcionamiento y aplicación de las estructuras de selección y de repetición.
Identificar tipos de datos, literales, constantes, variables, identificadores, operadores y salida de datos propios del lenguaje de programación C++.
Realizar ejercicios, donde se utilicen las estructuras secuencial, selectiva e iterativa desarrollando el programa

2. MARCO TEÓRICO

2.1 Conceptos Básicos

Las **estructuras de control** controlan el flujo de ejecución de un programa o función. Las estructuras de control permiten combinar instrucciones o sentencias individuales en una simple unidad lógica con un punto de entrada y un punto de salida.
Las instrucciones o sentencias se organizan en tres tipos de estructuras de control que sirven para controlar el flujo de la ejecución: *secuencia*, *selección* (*decisión*) y *repetición*, el flujo secuencial. Una **sentencia compuesta** es un conjunto de sentencias encerradas entre llaves ({ y }) que se utiliza para especificar un flujo secuencial.

2.1.1 Estructuras Secuenciales

El control fluye de la *sentencia*1 a la *sentencia*2, y así sucesivamente. Sin embargo, existen problemas que requieren etapas con dos o más opciones o alternativas a elegir en función del valor de una condición o expresión.

```
{
   sentencia1;
   sentencia2;
   .
   .
   .
   sentencia n;
}
```

2.1.2 Estructuras Selectivas: if, if-else, switch

La Sentencia if

En C++, la estructura de control de selección principal es una sentencia if. La sentencia if tiene dos alternativas o formatos posibles. El formato más sencillo tiene la sintaxis siguiente:

Sintaxis

```
if (Expresión)
 Acción;
```

Expresión lógica que determina si la acción se ha de ejecutar

Acción se ejecuta si la expresión lógica es verdadera

La Sentencia: Condición Doble if-else

Un segundo formato de la sentencia if es la sentencia if-else. Este formato de la sentencia if tiene la siguiente sintaxis:

Sintaxis

```
If (Expresión)
        Acción1;
else
        Acción2;
```

Expresión lógica que determina la acción a ejecutar

Acción1 que se ejecuta si la expresión lógica es verdadera

Acción2 que se ejecuta si la expresión lógica es falsa

La Sentencia: Condiciones Múltiples switch

La sentencia switch es una sentencia C++ que se utiliza para seleccionar una de entre múltiples alternativas.

Sintaxis

```
switch (selector)
{
        case etiqueta1 : sentencias1;
                         break;
        case etiqueta2 : sentencias2;
                         break;
        .
        .
        case etiqueta n : sentenciasn;
                         break;
        default: sentenciasd; // opcional
}
```

2.1.3 Estructuras Iterativas: while, for, do-while

Un bucle es cualquier construcción de programa que repite una sentencia o secuencia de sentencias un número de veces.

Repetición: El Bucle while

Sintaxis

1 while (condición_bucle)
  sentencia; ← cuerpo

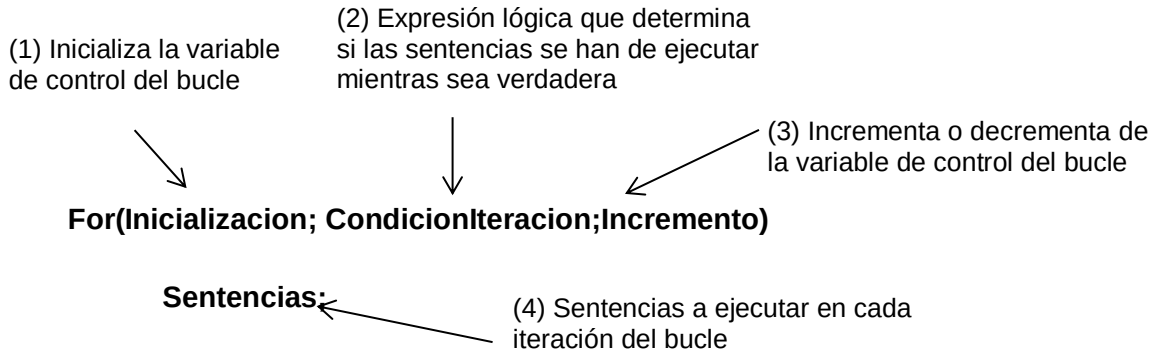
2 while (condición_bucle)
  {
    sentencia-1;
    sentencia-2;
    .
    .
    .
    sentencia-n;
  }

cuerpo

While.- palabra reservada C++
condición_bucle.- expresión lógica o booleana
sentencia.- sentencia simple o compuesta

Repetición: El Bucle for

Sintaxis

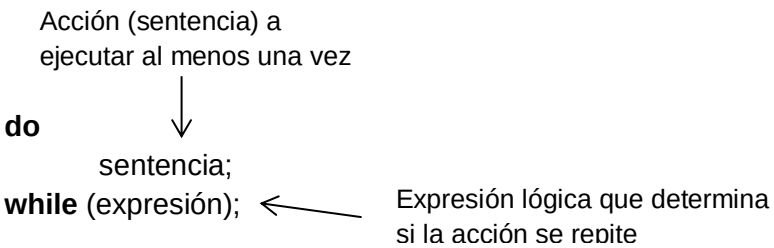


El bucle for contiene las cuatro partes siguientes:
• *Parte de inicialización*, que inicializa las variables de control del bucle. Se pueden utilizar variables de control del bucle simple o múltiple.
• *Parte de iteración,* que contiene una expresión lógica que hace que el bucle realice las iteraciones de las sentencias, mientras que la expresión sea verdadera.
• *Parte de incremento*, que incrementa o decrementa la variable o variables de control del bucle.
• *Sentencias*, acciones o sentencias que se ejecutarán por cada iteración del bucle.

Repetición: El Bucle do-while

La sentencia **do-while** se utiliza para especificar un bucle condicional que se ejecuta al menos una vez. Esta situación se suele dar en algunas circunstancias en las que se ha de tener la seguridad de que una determinada acción se ejecutará una o varias veces, pero al menos una vez.

Sintaxis



La construcción **do** comienza ejecutando sentencia. Se evalúa a continuación expresión. Si expresión es verdadera, entonces se repite la ejecución de sentencia. Este proceso continúa hasta que expresión es falsa.

3. METODOLOGIA

3.1 Diseño del Programa

Definición del problema:

Diseñar un programa que lea 20 números, contar los positivos, negativos y neutros.

Análisis del problema:

Datos de Entrada:
num.- numero

Proceso:
cp.- contador de positivos
cn.- contador de neutros
cneg.- contador de negativos
Se analiza el valor de numérico de num y se incrementa el contador.

```
si(num==0)
        cn++;
sino
        si(num>0)
          cp++;
        sino
          cneg++;
```

Salida:
Se imprime el valor de los contadores
cp, cn, cneg

SOLUCION:

```
//Leer 20 números, contar los positivos, negativos y neutros
#include<iostream>
using namespace std;
void main()
{
    int cp=0,cn=0,cneg=0,num;
    for(int i=1;i<=20;i++)
    {
        cout<<"Ingrese el numero: ";
        cin>>num;
        if(num==0)
          cn++;
        else
          if(num>0)
            cp++;
        else
            cneg++;
    }
    cout<<"El total de numeros positivos es: "<<cp<<endl;
    cout<<"El total de numeros negativos es: "<<cneg<<endl;
    cout<<"El total de numeros neutros es: "<<cn<<endl;
    system("pause");
}
```

4. RESULTADOS

El alumno aprendió a Identifica y aplicar la estructura correcta en el desarrollo del programa. Aplica la sintaxis correcta del lenguaje de programación C++. Conoce el funcionamiento y aplicación de las estructuras de selección y de repetición, Identificar tipos de datos, literales, constantes, variables, identificadores, operadores y salida de datos propios del lenguaje de programación C++.

5. CONCLUSION

A finalizar el proyecto nos ayuda a tener un mejor entendimiento de la implementación de las estructuras de control en el diseño de programas en el lenguaje de programación C++, cumpliendo con la estructura y sintaxis correcta propia del lenguaje de programación. Aportando un aprendizaje significativo a los estudiantes para la aplicación de las estructuras de un lenguaje de programación para resolver problemas reales.

6 RÚBRICA DE EVALUACION

INDICADORES	CUMPLE	NO CUMPLE	PTOS	CALIFICACIÓN
El diseño del programa cumple con la estructura y sintaxis del lenguaje C++.			**10**	
Los nombres de variables son significativos			**10**	
Cumple con razonamiento lógico			**10**	
Aplica la estructura de control correcta en el desarrollo del programa			**10**	
El programa funciona correctamente			**10**	

PROYECTO INTEGRADOR DE ARREGLOS, ESTRUCTURAS Y UNIONES

TEMA 4

Construye programas que utilicen arreglos unidimensionales y multidimensionales para solucionar problemas.

CONTENIDO

1 Introducción 3

1.1 Planteamiento del Problema 3

1.2 Justificación 3

1.3 Objetivos 4

2 Marco Teórico 4

3 Metodología 6

4 Resultados 8

5 Conclusión 8

6 Rúbrica de Evaluación 8

1. INTRODUCCION

Un array (arreglo) almacena muchos elementos del mismo tipo, tales como veinte enteros, cincuenta números de coma flotante o quince caracteres. El *array* es muy importante por diversas razones, una de ellas es almacenar secuencias o *cadenas* de texto.

• Existen arrays de una dimensión, de dos dimensiones y multidimensionales.

• En C++ los arrays se definen especificando el tipo de dato del elemento, el nombre del array y el tamaño de cada dimensión del array. Para acceder a los elementos del array se deben utilizar sentencias de asignación directas, sentencias de lectura/escritura o bucles.

En este proyecto se aprenderá el concepto y tratamiento de los arrays.

1.1 Planteamiento del Problema

Es posible almacenar, organizar y recuperar información de manera estructurada para representar programas del mundo real. ¿Qué son los arrays?, ¿Cómo aplicamos las herramientas para el desarrollo de programas, que implementen operaciones (almacenar y recuperar datos) sobre registros y arrays(unidimensionales y multidimensionales). La enseñanza-aprendizaje no debe ser solamente informativa si no también formativa, para que el estudiante desarrolle sus propias capacidades de aprendizaje.

1.2 Justificación

En este proyecto tiene como finalidad de propiciar al estudiante el conocimiento de los conceptos fundamentales de los arrays.

Se describen y analizan los siguientes conceptos:

• Un *array* (**arreglo**) es un tipo de dato estructurado que se utiliza para localizar y almacenar elementos de un tipo de dato dado.

• Existen arrays de una dimensión, de dos dimensiones y multidimensionales.

• En C++ los arrays se definen especificando el tipo de dato del elemento, el nombre del array y el tamaño de cada dimensión del array. Para acceder a los elementos del array se deben utilizar sentencias de asignación directas, sentencias de lectura/escritura.

• Los arrays de caracteres contienen cadenas de textos. En C++ se terminan las cadenas de caracteres situando un carácter nulo (‘\0’) como último byte de la cadena.

• C++ soporta arrays multidimensionales.

De igual forma brindar un aprendizaje significativo para que los estudiantes gocen de una herramienta para el desarrollo de programas de manipulación de datos en el lenguaje C++ para la asignatura de Programación Básica.

1.3 Objetivos

1.3.1 Objetivo General

Conoce y aplica estructuras de datos en un lenguaje de programación que permitan la organización de datos en la resolución de problemas reales.

1.3.2 Objetivos Específicos

Identifica las estructuras de datos: registros y arrays unidimensionales, bidimensionales.
Realiza ejercicios, donde se utilicen los arrays.
Aplica las herramientas para el desarrollo de programas, que implementen operaciones (almacenary recuperar datos) sobre registros y arrays (unidimensionales y multidimensionales),

2. MARCO TEÓRICO

2.1 Conceptos Básicos

Un *array* (**arreglo**) es un tipo de dato estructurado que se utiliza para localizar y almacenar elementos de un tipo de dato dado.
• Existen arrays de una dimensión, de dos dimensiones y multidimensionales.
• En C++ los arrays se definen especificando el tipo de dato del elemento, el nombre del array y el tamaño de cada dimensión del array. Para acceder a los elementos del array se deben utilizar sentencias de asignación directas, sentencias de lectura/escritura o bucles.

2.1.1 Arrays Unidimensionales

Un *array* o **arreglo** unidimensional es una secuencia de objetos del mismo tipo. Los objetos se llaman elementos del *array* y se numeran consecutivamente 0, 1, 2, 3... El tipo de elementos almacenados en el *array* puede ser cualquier tipo de dato de C++. Normalmente, el *array* se utiliza para almacenar tipos tales como char, int o float. Cada item del *array* se denomina *elemento*.
Estos arrays se conocen como *unidimensionales* (una sola dimensión) y se caracterizan por tener un solo subíndice. Estos arrays se conocen también por el término *listas*.

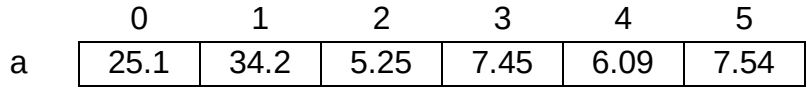


Fig 2.1 Representa Grafica de un *array* a con seis elementos

Podemos concluir que un array tiene:

Tamaño: cuantas cajas va a tener, el número de datos.
Tipo: cuál es el tipo de todos los datos del array.
Nombre: el único nombre bajo el cual vamos a dirigirnos al mismo.

2.1.2 Arrays Multidimensionales

Los *arrays multidimensionales* son aquellos que tienen más de una dimensión y, en consecuencia, más de un índice.
Un array de dos dimensiones equivale a una tabla con múltiples filas y múltiples columnas (figura 2.2).

	0	1	2	3
0	0,0	0,1	0,2	0,3
1	1,0	1,1	1,2	1,3
2	2,0	2,1	2,2	2,3
3	3,0	3,1	3,2	3,3

Figura 2.2 Estructura de Dos Dimensiones

3. METODOLOGIA

3.1 Diseño del Programa

Definición del problema:

Diseñar un programa que llene una matriz aleatoria una matriz de 4 filas por 4 columnas. Imprimir la diagonal principal.

x			
	x		
		x	
			X

Análisis del problema:

Datos de Entrada:
No hay datos ingresados por teclado, la matriz se llena aleatoriamente.
Proceso:
La diagonal principal tiene la característica que la fila y columna tienen mismo valor.

```
si(i==j)
        imprime el valor de a[i][j]
```

Salida:
Imprimir diagonal principal de la matriz

SOLUCION:

```
//4.12 imprimir la diagonal principal
#include<iostream>
 using namespace std;
 void main()
 {
  int a[4][4];
  for(int i=0;i<=3;i++)
  {
            for(int j=0;j<=3;j++)
             {
                  a[i][j]=rand()%100;//llenado aleatorio
                cout<<a[i][j]<<"\t";
             }
            cout<<endl;
  }
 cout<<"\n\n";
 for(int i=0;i<=3;i++)
  {
            for(int j=0;j<=3;j++)
                      if(i==j)
                       cout<<a[i][j]<<"\t";
                      else
                       cout<<"\t";
           cout<<endl;
  }
 system("pause");
 }//fin main
```

4. RESULTADOS

El alumno aprendió a Identificar las estructuras de datos: registros y arrays unidimensionales, bidimensionales. Y aplica las herramientas para el desarrollo de programas, que implementen operaciones (almacenary recuperar datos) sobre registros y arrays (unidimensionales y multidimensionales),

5. CONCLUSION

Al finalizar este proyecto acerca de arrays nos ayuda a tener mejor conocimiento de estos temas para aplicarlos en el desarrollo de programas, que implementen operaciones (almacenar y recuperar datos) sobre registros y arrays(unidimensionales y multidimensionales), este tema es muy importante para la asignatura de Programación Básica.

6 RÚBRICA DE EVALUACION

INDICADORES	CUMPLE	NO CUMPLE	PTOS	CALIFICACIÓN
El diseño del programa cumple con la estructura y sintaxis del lenguaje de programación C++.			**10**	
Los nombres de variables son significativos			**10**	
Cumple con razonamiento lógico			**10**	
La aplicación e implementación del array en el programa es correcta.			**10**	
El programa funciona correctamente			**10**	

PROYECTO INTEGRADOR DE MODULOS

TEMA 5

Conoce y aplica funciones en el desarrollo de programas para la optimización de los mismos y reutilización de código.

CONTENIDO

1 Introducción 3

1.1 Planteamiento del Problema 3

1.2 Justificación 3

1.3 Objetivos 4

2 Marco Teórico 4

3 Metodología 6

4 Resultados 11

5 Conclusión 11

6 Rúbrica de Evaluación 12

1. INTRODUCCION

Una función es un miniprograma dentro de un programa. Las funciones contienen varias sentencias bajo un solo nombre, que un programa puede utilizar una o más veces para ejecutar dichas sentencias. Las funciones ahorran espacio, reduciendo repeticiones y haciendo más fácil la programación, proporcionando un medio de dividir un proyecto grande en módulos pequeños más manejables.

1.1 Planteamiento del Problema

Es posible dividir un programa grande en módulos pequeños más manejables, con la aplicación de funciones. ¿Qué son las funciones?, ¿para qué nos sirven? ¿Cómo implementamos las funciones en desarrollo de programas del mundo real?, ¿Qué son los parámetros, cuál es su estructura? el conocimiento de estos temas logra así un mayor interés de aprendizaje en los estudiantes. La enseñanza-aprendizaje no debe ser solamente informativa si no también formativa, para que el estudiante desarrolle sus propias capacidades de aprendizaje.

1.2 Justificación

En este proyecto tiene como finalidad de propiciar que el estudiante implemente el diseño de bloques de códigos reutilizables, que optimizan la funcionalidad y reducen la complejidad de los programas desarrollados. Identificará la diferencia entre parámetros y argumentos, su estructura e importancia, realizara prácticas que involucren la implementación de métodos o funciones con pase de parámetros en la resolución de problemas del contexto. De igual forma brindar un aporte significativo para que los estudiantes gocen de una herramienta para el desarrollo de programas de manipulación de datos en el lenguaje C++ de la asignatura de Programación Básica

1.3 Objetivos

1.3.1 Objetivo General

Conoce y aplica la modularidad en el desarrollo de programas para la optimización de los mismos y reutilización de código.

1.3.2 Objetivos Específicos

Gestiona información sobre la declaración y el uso de métodos o funciones. Identifica la diferencia entre parámetros y argumentos, su estructura e importancia, mediante.
Realiza ejercicios que involucren la implementación de métodos o funciones con pase de parámetros en la resolución de problemas del contexto.

2. MARCO TEÓRICO

2.1 Conceptos Básicos

C++ se puede utilizar como *lenguaje de programación estructurada,* también conocida como *programación modular*. Por esta razón, para escribir un programa se divide éste en varios módulos, en lugar de uno solo largo. El programa se divide en muchos módulos (rutinas pequeñas denominadas *funciones*), que producen muchos beneficios: aislar mejor los problemas, escribir programas correctos más rápido y producir programas que son más fáciles de mantener.

2.1.1 Prototipos de las Funciones

C++ requiere que una función se declare o defina antes de su uso. La declaración de una función se denomina prototipo.

Sintaxis:
valor_de_retorno nombre_funcion (lista_tipos_de_parametros);

- El prototipo declara el valor de retorno de la función y el ***número***, ***tipos*** y ***orden*** de los parámetros (valores) que la función espera recibir.
- Normalmente, colocaremos los prototipos de funciones al principio del programa antes de la función main.

2.1.2 Diseño o Definición de Función

El diseño o definición de función, contiene las instrucciones con las que realizará su trabajo la función.

Sintaxis:

```
valor_de_retorno  nombre_funcion (lista_de_parametros)
 {
        proceso;
 }
```

2.1.3 Llamada a la Función

La llamada a una función, se realiza escribiendo el nombre de la función, seguido por la lista de parámetros entre paréntesis.

- Una función invocada puede regresar el control a su invocador de tres formas:
- Si la función no regresa valor, el control se devuelve al encontrar la llave final de la misma.
- ejecutando una instrucción *return*;
- Si la función regresa valor, se debe utilizar la siguiente instrucción para regresar el valor a su invocador: *return* valor;

Sintaxis:

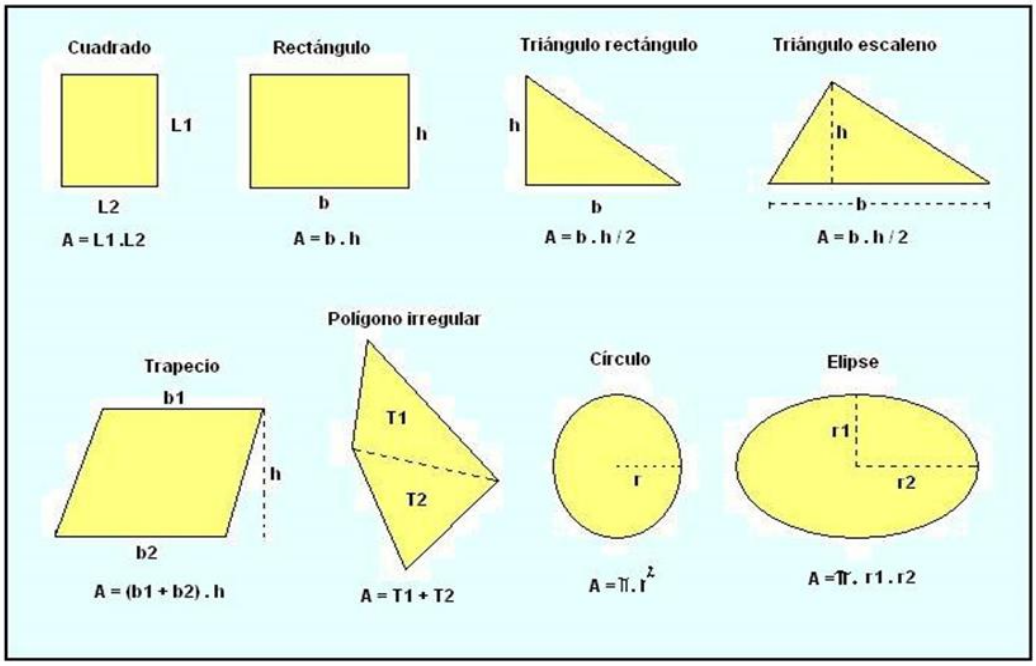
```
Nombre_de_funcion(parametros);
```

3. METODOLOGIA

3.1 Diseño del Programa

Definición del problema:

Diseñe el siguiente programa: mediante un menú presente las opciones de cálculo de área para cada figura y mediante una función calcule el área de cada figura geométrica:



Análisis del problema:

Datos de Entrada:
Se ingresará por teclado las longitudes de la figura
geométrica a calcular su área.
b.-base
h.-altura

Proceso:
Se calcula el área de la figura geométrica deseada.
a=b*h/2

Salida:
Imprimir el área de la figura geométrica
a

SOLUCION:

```
// area de diversas figuras
#include <iostream>
using namespace std;
void cuadrado();//declaración de funciones
void rectangulo();
void trec();
void tesc();
void trapecio();
void poligono();
void circulo();
void elipse();
void main() {
    int op;//declarar variables
    cout<<"Elija el numero dependiendo de que figura sacará el área \n";
    cout<<"(1)cuadrado\n";
    cout<<"(2)rectángulo\n";
    cout<<"(3)triángulo rectángulo\n";
    cout<<"(4)triángulo escaleno \n";
    cout<<"(5)trapecio\n";
    cout<<"(6)polígono irregular\n";
    cout<<"(7)círculo\n";
    cout<<"(8)elipse\n";
    cin>>op;//lectura de valor
```

```
    switch (op){//switch para escoger opción
    case 1:
        cuadrado();//Llamada de función
        break;
    case 2:
        rectangulo();
        break;
    case 3:
        trec();
        break;
    case 4:
        tesc();
        break;
    case 5:
        trapecio();
    break;
    case 6:
        poligono();
        break;
    case 7:
        circulo();
        break;
    case 8:
        elipse();
        break;
    default:
        cout<<"Error\n";
        break;
    }
}//fin del main

//inicio de funciones.
void cuadrado(){
    float L,a;
    cout<<"Ingrese cuanto mide el lado\n";
    cin>>L;
    a=L*L;
    cout<<"El área de su cuadrado es: "<<a<<endl;
}
```

```
void rectangulo(){
    float b,h,a;
    cout<<"Ingrese cuando mide la base\n";
    cin>>b;
    cout<<"Ahora, ingrese la altura\n";
    cin>>h;
    a=b*h;
    cout<<"El área de su rectángulo es "<<a<<endl;
}
void trec(){
    float b,h,a;
    cout<<"Ingrese cuando mide la base\n";
    cin>>b;
    cout<<"Ahora, ingrese la altura\n";
    cin>>h;
    a=b*h/2;
    cout<<"El área de su triángulo rectángulo es: "<<a<<endl;
}
void tesc(){
    float b,h,a;
    cout<<"Ingrese cuando mide la base\n";
    cin>>b;
    cout<<"Ahora, ingrese la altura\n";
    cin>>h;
    a=b*h/2;
    cout<<"El área de su triángulo escaleno es: "<<a<<endl;
}
void trapecio(){
    float b1,b2,h,a;
    cout<<"Ingrese cuanto mide la base 1\n";
    cin>>b1;
    cout<<"Ingrese cuanto mide la base 2\n";
    cin>>b2;
    cout<<"Ingrese la altura del trapecio\n";
    cin>>h;
    a=(b1+b2)*h;
    cout<<"El área de su trapecio es "<<a<<endl;
}
void poligono(){
    float b1,h1,b2,h2,a;
    cout<<"Ingrese la base del triángulo 1\n";
```

```
    cin>>b1;
    cout<<"Ingrese la altura del triángulo 1\n";
    cin>>h1;
    cout<<"Ingrese la base del triángulo 2\n";
    cin>>b2;
    cout<<"Ingrese la altura del triángulo 2\n";
    cin>>h2;
    a=(b1*h1/2)+(b2*h2/2);
    cout<<"El área del poligono irregular es "<<a<<endl;
}
void circulo(){
    float r,a;
    const float pi=3.1416;
    cout<<"Ingrese el radio del círculo\n";
    cin>>r;
    a=r*r*pi;
    cout<<"El área de su círculo es "<<a<<endl;
}
void elipse(){
    float r1,r2,a;
    const float pi=3.1416;
    cout<<"Ingrese el radio 1 del elipse, es el menor, tambien conocido como b\n";
    cin>>r1;
    cout<<"Ahora, ingrese el radio 2 del elispse, el mayor, conocido como a\n";
    cin>>r2;
    a=pi*r1*r2;
    cout<<"El área de su elipse es "<<a<<endl;
}
      //fin de funciones
```

4. RESULTADOS

El alumno aprendió a gestionar información sobre la declaración y el uso de métodos o funciones. Identifica la diferencia entre parámetros y argumentos, su estructura e importancia, mediante. La realización de este proyecto que involucra la implementación de métodos o funciones con pase de parámetros en la resolución de problemas del contexto.

5. CONCLUSION

Al finalizar este proyecto acerca de métodos o funciones nos ayuda a tener mejor conocimiento de estos temas para aplicarlos en el desarrollo de programas para la optimización de los mismos y reutilización de código, este tema es muy importante para la asignatura de Programación Básica.

6 RÚBRICA DE EVALUACION

INDICADORES	CUMPLE	NO CUMPLE	PTOS	CALIFICACIÓN
El diseño del programa cumple con la estructura y sintaxis del lenguaje de programación C++.			**10**	
Los nombres de variables son significativos			**10**	
Cumple con razonamiento lógico			**10**	
la implementación de métodos o funciones con pase de parámetros en la resolución del problema es correcta.			**10**	
El programa funciona correctamente			**10**	

PROYECTO INTEGRADOR DE PUNTEROS

TEMA 6

Declara y aplica apuntadores como manera de manipular información en llamadas por valor y referencia.

CONTENIDO

1 Introducción 3

1.1 Planteamiento del Problema 3

1.2 Justificación 3

1.3 Objetivos 4

2 Marco Teórico 4

3 Metodología 6

4 Resultados 7

5 Conclusión 7

6 Rúbrica de Evaluación 7

1. INTRODUCCION

Cada variable que se declara en C++ tiene una dirección asociada con ella. Un *puntero* es una dirección de memoria.

1.1 Planteamiento del Problema

Un puntero en C o en C++, también indica dónde encontrar algo, ¿dónde encontrar los datos que están asociados con una variable? *Un puntero C++ es la dirección de una variable*. La enseñanza-aprendizaje no debe ser solamente informativa si no también formativa, para que el estudiante desarrolle sus propias capacidades de aprendizaje.

1.2 Justificación

En este proyecto tiene como finalidad de propiciar que el estudiante implemente el diseño de programas que involucren la implementación de apuntadores en la resolución de problemas del contexto. De igual forma brindar un aporte significativo para que los estudiantes gocen de una herramienta para el desarrollo de programas de manipulación de direcciones de memoria en el lenguaje C++ de la asignatura de Programación Básica.

1.3 Objetivos

1.3.1 Objetivo General

Declara y aplica apuntadores como manera de manipular información en llamadas por valor y referencia.

1.3.2 Objetivos Específicos

Realiza ejercicios que involucren la declaración e implementación del uso de apuntadores en la resolución de problemas del contexto.

2. MARCO TEÓRICO

2.1 Conceptos Básicos

Los apuntadores en C y C++ son una herramienta muy potente de programación. Sin embargo, cuando los programadores cometen un error en su utilización, puede ser muy difícil encontrar el error, por lo cual es importante saber utilizarlos muy bien.

2.1.1 Definición de Apuntador

Cuando se declara una variable, el compilador reserva un espacio de memoria para ella y asocia el nombre de esta a la dirección de memoria desde donde comienzan los datos de esa variable. Las direcciones de memoria se suelen describir como números en hexadecimal.
Un apuntador es una variable cuyo valor es la dirección de memoria de otra variable. Se dice que un apuntador “apunta” a la variable cuyo valor se almacena a partir de la dirección de memoria que contiene el apuntador.
Por ejemplo, si un apuntador p almacena la dirección de una variable x, se dice que “p apunta a x”.

2.1.2 Referenciación

La referenciación es la obtención de la dirección de una variable. En C y C++ esto se hace a través del operador '&', aplicado a la variable a la cual se desea saber su dirección.
Ejemplo:
int x = 25 ;
cout << "La dirección de x es: " << &x << endl;
Este código imprime un valor del estilo "0x4fffd34". Este valor puede variar durante cada ejecución del programa.

2.1.3 Declaración de Apuntadores

Para declarar un apuntador se especifica el tipo de dato al que apunta, el operador '*', y el nombre del apuntador.
La sintaxis es la siguiente:
<tipo de dato apuntado> *<indentificador del apuntador>
Ejemplos:
int *ptr1; // Apuntador a un dato de tipo entero (int)
char *cad1, *cad2; // Dos apuntadores a datos de tipo carácter (char)
float *ptr2; // Apuntador a un dato de tipo punto-flotante (float)

3. METODOLOGIA

3.1 Diseño del Programa

Definición del problema:

Escribir un programa que ejecute con una sola sentencia lo indicado en cada literal. Suponiendo que han sido declaradas las variables enteras largas valor1 y valor2, y que valor1 se inicialice a 200000.

a) Declarar la variable Ptr que sea un puntero a una variable de tipo long.
b) Asignar la dirección de la variable valor1 a la variable puntero Ptr.
c) Imprimir el valor de la variable apuntada por Ptr.
d) Asignar el valor de la variable apuntada por Ptr a la variable valor2.
e) Imprimir el valor de valor2.
f) Imprimir la dirección de valor1.
g) Imprimir la dirección almacenada en Ptr.

SOLUCION:

```
#include <iostream>
using namespace std;
int main ()
{
	long int valor1 = 200000, valor2;
	long int* ptr = &valor1;
	cout << "\nptr->valor1: " << *ptr << endl;
	valor2 = *ptr;
	cout << "\nvalor2: " << valor2 << endl;
	cout << "\n&valor1: " << &valor1 << endl;
	cout << "\nptr: " << ptr << endl;
	return 0;
}
```

4. RESULTADOS

El alumno aprendió a declarar apuntadores, a realizar aplicaciones que involucran el uso de apuntadores.

5. CONCLUSION

Al finalizar este proyecto acerca de apuntadores nos ayuda a tener mejor conocimiento de estos temas para aplicarlos en la realización de aplicaciones que involucran el uso de apuntadores, este tema es muy importante para la asignatura de Programación Básica.

6 RÚBRICA DE EVALUACION

INDICADORES	CUMPLE	NO CUMPLE	PTOS	CALIFICACIÓN
Declara la variable Ptr como un puntero a una variable de tipo long.			**10**	
Asigna la dirección de la variable valor1 a la variable puntero Ptr.			**10**	
Imprime el valor de la variable apuntada por Ptr.			**10**	
Asigna el valor de la variable apuntada por Ptr a la variable valor2..			**10**	
Imprime el valor de valor2, Imprimir la dirección de valor1, Imprimir la dirección almacenada en Ptr.			**10**	

PROYECTO INTEGRADOR DE FICHEROS

TEMA 7

Utiliza las librerías de manejos de ficheross para crearlos, leerlos, escribirlos, buscar registros y cerrarlos, ya sean secuenciales o directos.

CONTENIDO

1 Introducción 3

1.1 Planteamiento del Problema 3

1.2 Justificación 3

1.3 Objetivos 4

2 Marco Teórico 4

3 Metodología 6

4 Resultados 7

5 Conclusión 7

6 Rúbrica de Evaluación 7

1. INTRODUCCION

Un fichero o archivo es una colección ordenada de datos que tienen entre sí una relación y que se almacenan de forma permanente en un dispositivo de memoria no volátil.

1.1 Planteamiento del Problema

En general, la mayor parte de las aplicaciones necesitan leer datos de ficheros o escribir datos en ficheros para un uso posterior por el mismo u otro programa.
La enseñanza-aprendizaje no debe ser solamente informativa si no también formativa, para que el estudiante desarrolle sus propias capacidades de aprendizaje.

1.2 Justificación

En este proyecto tiene como finalidad de propiciar que el estudiante implemente ficheros en la resolución de problemas del contexto. De igual forma brindar un aporte significativo para que los estudiantes gocen de una herramienta para el desarrollo de programas de manipulación de direcciones de memoria en el lenguaje C++ de la asignatura de Programación Básica.

1.3 Objetivos

1.3.1 Objetivo General

Utiliza las librerías de manejos de ficheros para crearlos, leerlos, escribirlos, buscar registros y cerrarlos, ya sean secuenciales o directos.

1.3.2 Objetivos Específicos

Realiza ejercicios que involucren la declaración e implementación del uso de ficheros. Leer, escribir datos de ficheros ya sean tipos de datos simples o estructurados.

2. MARCO TEÓRICO

2.1 Conceptos Básicos

La existencia del fichero hace posible que, al terminar una sesión de trabajo, se almacenen los datos y resultados en el disco, para poder retomar la tarea en una sesión posterior.

2.1.1 Definición de Fichero

Un fichero o archivo es una colección ordenada de datos que tienen entre sí una relación y que se almacenan de forma permanente en un dispositivo de memoria no volátil.

2.1.2 Ficheros en C++

Para trabajar con ficheros en C++ debe incluirse la cabecera fstream.

Dependiendo de lo que deseemos hacer con el fichero, usaremos objetos de las clases:

ifstream (input file stream), clase orientada para la lectura
ofstream (output file stream). clase orientada para la escritura
fstream (file stream), cuando deseemos alternativamente leer o escribir del mismo fichero en el mismo programa.

2.1.3 Escritura de un Fichero Tipo Texto

Para crear un fichero de texto y escribir algo en él: tenemos que seguir los siguientes pasos.

Escritura:

- Declaramos un objeto ofstream,
- Por medio de la función open abrimos el fichero
- Escribimos en el fichero los datos necesarios
- Cerramos el fichero con la función Close

Para escribir o leer, se usan los operadores de inserción << y extracción >> como si del teclado o consola se tratase.

3. METODOLOGIA

3.1 Diseño del Programa

Definición del problema:

Crear un fichero de texto, que genere y escriba los 10 primeros enteros.

SOLUCION:

```
#include <fstream>  // Para ofstream
#include <iostream> // Para cout
using namespace std;

int main()
{
  ofstream fich("ejemplo.txt");
  if (!fich)
  {
    cout << "Error al abrir ejemplo.txt\n";
    exit(EXIT_FAILURE);
  }

  for (int i = 0; i < 10; ++i)
    fich << i << endl;
fich.close();
return 0;
}
```

4. RESULTADOS

El alumno aprendió a leer, escribir datos de fichero ya sean tipos de datos simples o estructurados.

5. CONCLUSION

Al finalizar este proyecto acerca de ficheros nos ayuda a tener mejor conocimiento de estos temas para aplicarlos en la realización de aplicaciones que involucran el uso de ficheros, este tema es muy importante para la asignatura de Programación Básica.

6 RÚBRICA DE EVALUACION

INDICADORES	CUMPLE	NO CUMPLE	PTOS	CALIFICACIÓN
Se declara un objeto ofstream			**10**	
Por medio de la función open se abrió el fichero			**10**	
Escribe en el fichero los datos necesarios			**10**	
Se cerró el fichero con la función Close			**10**	
El programa cumple con las reglas de sintaxis propias del lenguaje c++.			**10**	