SEP SNEST DGEST

INSTITUTO TECNOLÓGICO DE TOLUCA

INFORME DE PERÍODO SABÁTICO

“ALGORITMOS Y LENGUAJES DE PROGRAMACIÓN”

PRESENTADO POR:

MARÍA DEL SOCORRO MORENO FIGUEROA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

METEPEC, MÉXICO A 12 DE AGOSTO DE 2014

INDICE

Contenido

1. INTRODUCCIÓN A LA COMPUTACIÓN Y HOJA DE CÁLCULO. 1

1.1 Introducción a la computación. 1

1.2 Sistemas operativos. 2

1.3 Elementos de Excel. 3

1.4 Fórmulas y funciones. 11

1.5 Tablas dinámicas. 22

1.6 Macros. 28

1.7 Aplicaciones. 31

2. DESARROLLO DE LÓGICA ALGORÍTMICA. 38

2.1 Metodología para la solución de problemas. 38

2.2 Metodología para el diseño de software: 39

Top down, bottom up, modular y programación estructurada. 39

Elementos y reglas de la representación gráfica y manuscrita de los algoritmos (diagrama de flujo, diagrama N-S, diagrama estructurado y pseudocódigo). 41

2.3 Implementación de algoritmos secuenciales. 47

2.4 Pruebas y depuración. 52

3. INTRODUCCIÓN A LA PROGRAMACIÓN DE UN LENGUAJE ESTRUCTURADO. 54

3.1 Introducción a la programación. 54

3.2 Estructura básica de un programa. 54

3.3 Tipos de datos. 56

3.4 Identificadores. 56

3.5 Almacenamiento, direccionamiento y representación en memoria. 57

3.6 Proposición de Asignación. 59

3.7 Operadores, operandos y expresiones. 59

3.8 Prioridad de operadores, evaluación de expresiones. 63

3.9 Elaboración de programas. 66

4. ESTRUCTURAS SELECTIVAS Y DE REPETICIÓN. 70

4.1 Selectiva simple. 70

4.2 Selectiva doble. 72

4.3 Selectiva anidada. 76
4.4 Selectiva múltiple. 77
4.5 Repetir mientras. 81
4.6 Repetir hasta. 87
4.7 Repetir desde. 91
4.8 Elaboración de programas. 95
5. ARREGLOS Y ARCHIVOS. 99
5.1 Arreglos unidimensionales. 99
5.2 Arreglos bidimensionales. 104
5.3 Archivos. 112
5.4 Elaboración de programas. 130
6. FUNCIONES. 133
6.1 Introducción. 133
6.2 Funciones estándar. 133
6.3 Entrada y salida de datos. 141
6.4 Funciones definidas por el usuario. 148
6.5 Pase por valor. 156
6.6 Pase por referencia. 161
6.7 Punteros. 165
6.8 Elaboración de programas. 182
Bibliografía 185

ÍNDICE DE TABLAS YFIGURAS

Tabla 1.1. Dispositivos de entrada, salida y mixtos. 2
Tabla 1.2. Operadores que se utilizan para crear una fórmula. 11
Tabla 1.3. Ejemplos de fórmulas. 12
Tabla 1.4. Errores en Excel. 12
Tabla 2.1. Símbolos de diagrama de flujo. 41
Tabla 3.1 Tipos de datos simples. 55
Tabla 3.2. Palabras reservadas del lenguaje C. 56
Tabla 3.3. Ejemplos de declaración de variables. 56
Tabla 3.4. Operadores aritméticos. 59
Tabla 3.5. Jerarquía de los operadores aritméticos. 59
Tabla 3.6 Ejemplos de operadores simplificados. 59
Tabla 3.7. Operadores de incremento y decremento. 60
Tabla 3.8. Operadores relacionales. 60
Tabla 3.9 Ejemplos del uso de operadores relacionales. 61
Tabla 3.10 Tabla de verdad de los operadores lógicos. 61
Tabla 3.11. Ejemplos del uso de los operadores lógicos. 61
Tabla 3.12 Prioridad de los operadores en lenguaje C. 62
Tabla 5.1. Modo para abrir un archivo. 114
Tabla 6.1. Funciones matemáticas (librería *math.h*). 134
Tabla 6.2. Operaciones con cadenas de caracteres (*string.h*). 137
Tabla 6.3. Pruebas y conversiones de caracteres (*ctype.h*). 138
Tabla 6.4. Funciones para la conversión de tipos (stdlib.h). 140
Tabla 6.5. Formato de escritura de las variables. 143
Tabla 6.6 Secuencias de escape. 144
Tabla 6.7 Funciones de entrada y salida sin formato. 147
Figura 1.1. Elementos de una hoja de cálculo. 4
Figura 1.2. **Personalizar barra de herramientas de acceso rápido**. 6
Figura 1.3. **Lanzador de cuadro de diálogo** del grupo formato. 6
Figura 1.4 Cuadro de diálogo del **Formato celdas**. 7
Figura 1.5. Selección de un rango en el **Cuadro de nombres**. 8
Figura 1.6. Nombrar a un grupo de celdas. 8
Figura 1.7. Botón cancelar, botón introducir, botón insertar función. 9
Figura 1.8. **Barra de desplazamiento.** 9
Figura 1.9. **Barra de estado**. 10
Figura 1.10. Cuadro de diálogo **Guardar como**. 11
Figura 1.11. División entre cero 13
Figura 1.12. **Auditoría de fórmulas**. 14
Figura 1.13. Referencia relativa. 15
Figura 1.14. Referencia absoluta. 15
Figura 1.15. Cuadro de diálogo **Insertar función**. 17
Figura 1.16. Función CONTAR.SI. 18
Figura 1.17. Función SUMAR.SI. 19
Figura 1.18. Función PROMEDIO.SI. 19

Figura 1.19. Función BUSCARH. 20
Figura 1.20. Cuadro de diálogo **Argumentos de función**. 21
Figura 1.21. Función BUSCARV. 21
Figura 1.22. Base de datos. 22
Figura 1.23. Cuadro de diálogo **Crear tabla dinámica**. 23
Figura 1.24. Pantalla de tabla dinámica. 23
Figura 1.25. Tabla dinámica. 24
Figura 1.26. Botones para ordenar ascendente y descendente. 24
Figura 1.27. Tabla dinámica ordenada ascendentemente. 24
Figura 1.28. Etiquetas de fila. 25
Figura 1.29. Submenú de Etiquetas de fila. 25
Figura 1.30. Lista de opciones. 26
Figura 1.31. Filtrar por valor. 26
Figura 1.32. Tabla dinámica filtrada. 26
Figura 1.33. Ficha Vista. 27
Figura 1.34. Cuadro de diálogo **Grabar macro**. 28
Figura 1.35. Ícono para **Detener Grabación.** 28
Figura 1.36. Cuadro de diálogo **Macro** para eliminar una macro. 29
Figura 2.1. Estructura condicional o selectiva: (a) diagrama de flujo: (b) diagrama N-S. 43
Figura 3.1. Representación gráfica de la memoria. 56
Figura 4.1. Estructura selección simple if. 71
Figura 4.2. La condición if se cumple. 74
Figura 4.3. La sentencia else se ejecuta. 74
Figura 4.4. La condición if se cumple y realiza bloque de instrucciones. 75
Figura 4.5. Ejecución del bloque de instrucciones de la sentencia else. 75
Figura 4.6. Ejecución del programa. 77
Figura 4.7. Caso 1. 79
Figura 4.8. Caso 2. 80
Figura 4.9 Caso 3 o 4. 80
Figura 4.10. Ejemplo del default. 80
Figura 4.11. Ciclo mientras controlado por un contador. 82
Figura 4.12.Ejecución del programa con repetición mientras. 83
Figura 4.13. Ciclo controlado por un centinela. 84
Figura 4.14. Programa que resuelve la función $f(x) = x * \log(x)$. 86
Figura 4.15. Ciclo controlado por una tarea. 87
Figura 4.16. Estructura **do-while** controlada por un contador numérico. 88
Figura 4.17. Estructura **do-while** controlada por un contador alfabético. 89
Figura 4.18. Programa controlado por un centinela. 90
Figura 4.19. Ciclo controlado por una tarea. 91
Figura 4.20. Programa con ciclo **for**. 93
Figura 4.21. Ejecución de programa con ciclo **for**. 94
Figura 4.22. Programa con ciclo for descendente. 95
Figura 5.1. Arreglo unidimensional. 99

Figura 5.2. Programa que almacena estaturas 102
Figura 5.3. Programa que imprime posiciones. 103
Figura 5.4. Busqueda de datos. 104
Figura 5.5. Uso de función rand(). 107
Figura 5.6. Dimensiones no válidas. 109
Figura 5.7. Suma de matrices. 110
Figura 5.8. Encuentra elemento mayor y menor en una matriz. 111
Figura 5.9. Escritura de un archivo de texto. 113
Figura 5.10. Lectura de caracteres de un archivo. 115
Figura 5.11. Archivo con cadena de caracteres. 116
Figura 5.12. Archivo y cadena de caracteres. 117
Figura 5.13 Ejecución de programa con archivo y acceso secuencial. 119
Figura 5.14. Lectura e impresión de un archivo secuencial. 121
Figura 5.15. Salida de ejemplo del programa. 123
Figura 5.16. Lectura de archivo con acceso directo. 126
Figura 5.17. Impresión de archivo con acceso directo. 127
Figura 5.18. Ejecución del programa que realiza modificaciones. 129
Figura 5.19. Archivo “alum.dat” modificado. 130
Figura 6.1. Librería math.h . 135
Figura 6.2. Librería string.h. 138
Figura 6.3. Librería ctype.h. 140
Figura 6.4. Librería stdlib.h. 141
Figura 6.5. Salida con formato y alineación a la derecha. 145
Figura 6.6. Salida con formato y relleno. 145
Figura 6.7. Salida con formato y alineación a la izquierda. 146
Figura 6.8. Salida con formato decimal. 147
Figura 6.9. Salida sin formato. 148
Figura 6.10. Ejecución de programa con funciones. 153
Figura 6.11. Función que realiza suma. 154
Figura 6.12. Ejecución de programa con funciones y prototipo. 155
Figura 6.13. Función que calcula el cuadrado. 156
Figura 6.14. Función con parámetros. 158
Figura 6.15. Función suma de cuatro números. 159
Figura 6.16. Programa con dos funciones. 161
Figura 6.17. Parámetros por referencia. 162
Figura 6.18. Cálculo del cubo. 163
Figura 6.19. Conversión de minúsculas a mayúsculas. 164
Figura 6.20. Cálculo de pares e impares. 165
Figura 6.21. Impresión del valor y dirección de una variable. 167
Figura 6.22. Asignaciones de punteros. 168
Figura 6.23. Uso de punteros, variables y valor. 169
Figura 6.24. Modificaciones al programa anterior. 169
Figura 6.25. Continuación de uso de punteros. 170

Figura 6.26. Incrementando el puntero. 170
Figura 6.27. Modificando el puntero. 171
Figura 6.28. Desplazamiento del puntero. 171
Figura 6.29. Vectores y funciones. 173
Figura 6.30. Vectores, funciones y parámetros. 176
Figura 6.31. Método burbuja. 178
Figura 6.32. Matriz, funciones y parámetros. 180
Figura 6.33. Continuación de la ejecución del programa. 180
Figura 6.34. Determinación de matriz simétrica. 182

1. INTRODUCCIÓN A LA COMPUTACIÓN Y HOJA DE CÁLCULO.

1.1 Introducción a la computación.

Un sistema de cómputo tiene cuatro componentes de **hardware** (componentes físicos de la computadora) principales: el procesador; memoria, dispositivos de entrada/salida y dispositivos de almacenamiento.

- **El procesador:** Es el cerebro de la computadora, la parte que interpreta y ejecuta las instrucciones. (NORTON, 1994)

  El término **unidad de procesamiento central (CPU)** se usa exclusivamente para referirse al procesador de una computadora, ésta contiene la inteligencia de la máquina; es donde se realizan los cálculos y las decisiones.

- **La memoria:** La CPU utiliza la memoria de la computadora para guardar piezas de información mientras trabaja con ellas y como un tablero de recortes para hacer cálculos. (NORTON, 1994)

  Estas piezas de información son representadas electrónicamente en los circuitos de los chips de memoria y, mientras permanezcan en memoria, la computadora puede tener acceso a ellas en forma directa. A esta memoria se le llama **memoria de acceso aleatorio (RAM)**, la RAM almacena información sólo mientras la computadora está encendida. Cuando apagas o arrancas nuevamente la computadora, la información desaparece, a menos que se haya guardado en un dispositivo de almacenamiento.

- **Dispositivos de entrada/salida:** Son los medios por los cuales se comunica la computadora con los usuarios y con otras máquinas o dispositivos.

  Los dispositivos de entrada aceptan datos e instrucciones del usuario. Los dispositivos de salida regresan datos procesados, esto es, información al usuario.

  En la tabla 1.1 se presentan algunos ejemplos de dispositivos de entrada y salida.

Tabla 1.1. Dispositivos de entrada, salida y mixtos.

Dispositivo de entrada	Dispositivo de salida	Dispositivos de E/S
mouse	impresora	Unidades de almacenamiento
teclado	pantalla	router
scanner	plotter	módem
joysticks	bocinas	tarjeta de red
lápiz óptico	auriculares	pantalla táctil
micrófono	proyector	
webcam		

- **Dispositivos de almacenamiento:** La computadora necesita un lugar dónde guardar los datos que no está procesando en ese momento, ese es el propósito del dispositivo de almacenamiento.
  Algunos ejemplos de dispositivo de almacenamiento son: disco duro, discos magnéticos de 3.5 pulgadas, unidad de CD-ROM o lectora, unidad de CD-RW o grabadora, unidad de DVD-ROM o lectora de DVD, lector de tarjetas de memoria, memoria flash entre otros.

1.2 Sistemas operativos.

Un sistema operativo (SO) es un conjunto de programas que comunican al usuario con la computadora y administran sus recursos de manera eficiente. (FERNANDEZ, 2012)

Entre los principales sistemas operativos se pueden destacar los siguientes:

- **Windows.** Estos sistemas operativos pertenecen a la empresa Microsoft. Dispone de una interfaz gráfica sencilla e intuitiva que facilita al usuario la utilización de la computadora. Se usan con computadoras de escritorio y portátiles.

- **Unix.** Est**e** sistema operativo adopta diferentes nombres en función de la empresa que lo comercializa. Por ejemplo: AIX (versión de IBM), Xenix (versión de Microsoft), Sinix (versión de Siemens), Linux, Unix Sco, etcétera. UNIX se suele utilizar en las grandes empresas, donde es necesario que varios empleados estén trabajando a la vez con la misma información, por ejemplo, en los bancos. Sin embargo Linux está empezando a ser considerado por los especialistas en PC como una buena alternativa, ya que es un software libre, es decir, no requiere licencia y permite a los desarrolladores aportar mejoras; a diferencias de los sistemas Windows.

- **Linux.** Es un núcleo libre de sistema operativo (también suele referirse al núcleo como kernel) basado en Unix. Es uno de los principales ejemplos de software libre

y de código abierto. Linux está licenciado bajo la GPL v2 y está desarrollado por colaboradores de todo el mundo.

- **Mac OS.** Es el sistema operativo usado por las computadoras Apple Macintosh.

El sistema operativo es el que controla a la computadora. Las principales funciones de estos sistemas son:

- **Coordinar y manipular los recursos de la computadora (hardware)**, tales como la memoria, la impresora, las unidades de disco, el teclado y el ratón entre otros.

- **Interpretar las instrucciones** que los usuarios introducen mediante un comando escrito (modo texto) o un clic (modo gráfico), según el tipo de sistema.

- **Administrar las tareas** para que su ejecución sea la más adecuada de acuerdo con los recursos con que cuenta la computadora.

1.3 Elementos de Excel.

Una hoja de cálculo es un programa que se utiliza para realizar cálculos matemáticos desde un nivel sencillo hasta operaciones altamente complejas. (GONZÁLEZ, 2002)

Como su nombre indica, las hojas de cálculo se caracterizan por presentar un aspecto de hoja cuadriculada; es decir, un área de trabajo estructurado en columnas y filas. La intersección de las columnas con las filas da lugar a unas casillas denominadas **celdas**. En cada una de estas celdas se puede introducir un dato. La celda puede ser identificada por sus coordenadas columna/fila, a lo que se llama **dirección de celda**, primero va la columna y luego el número de fila (por ejemplo B2). La celda que tengamos seleccionada con el cursor o ratón se le llama **celda activa** y es la que se ve resaltada en la hoja de cálculo.

Una hoja de cálculo, en general no sólo permite trabajar con datos numéricos, sino también con información referente por ejemplo al personal de una empresa, obteniéndose así una **Base de Datos**. Además, a partir de los datos introducidos en la hoja, es posible realizar **Gráficos** que ofrezcan una visión global de la relación entre dichos datos. (GRUPO NORIEGA EDITORES, 2000)

A diferencia de la versión 2007, Excel 2010 sustituye el **Botón de Office** por el menú **Archivo** el cual contiene los elementos para la configuración de los archivos. Sigue mostrando las fichas **Inicio**, **Insertar**, **Diseño de página**, **Fórmulas**, **Datos**, **Revisar** y **Vista**.

En esta interfaz podrá trabajar con diferentes elementos, los cuales se muestran en la figura 1.1.

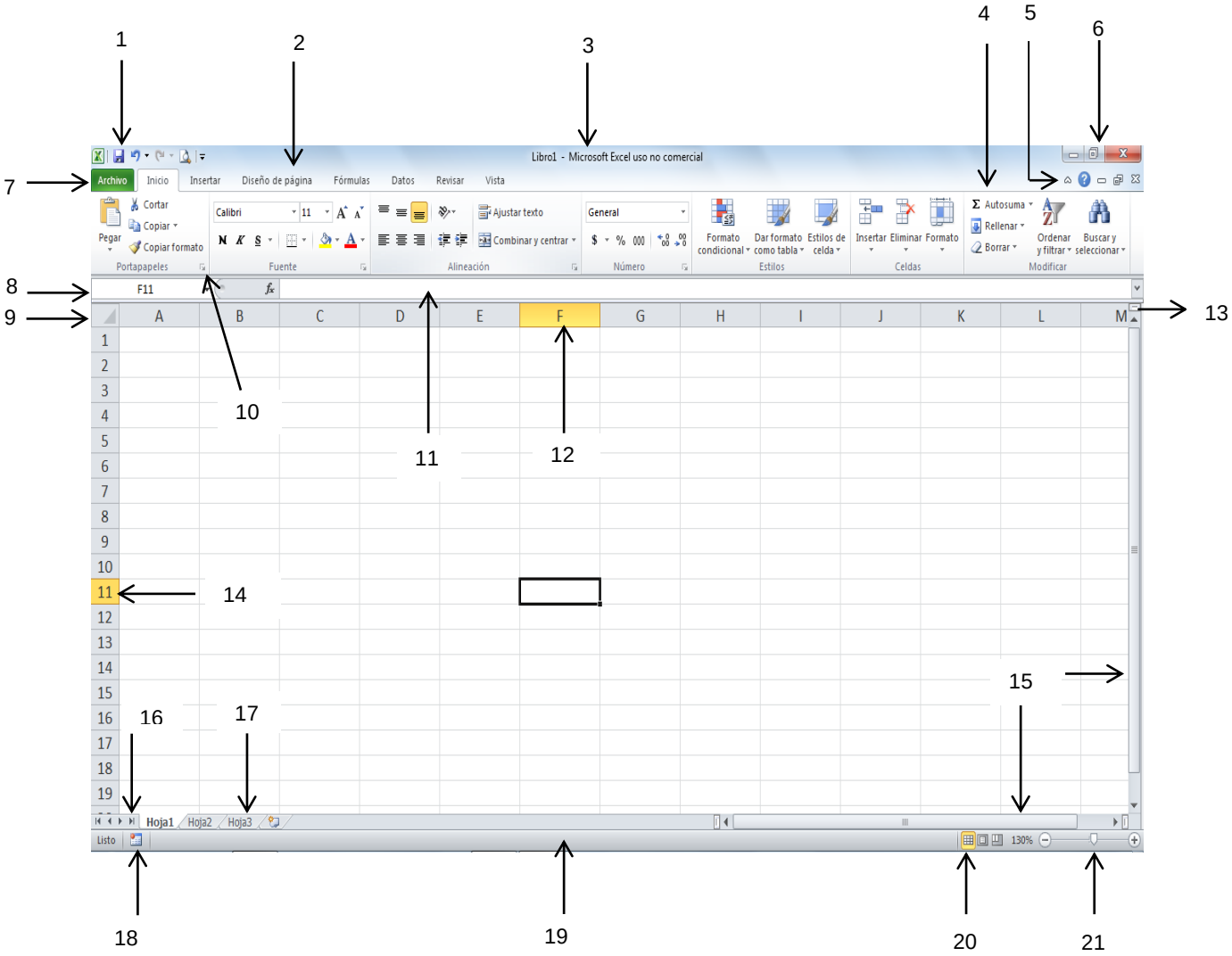


Figura 1.1. Elementos de una hoja de cálculo.

1.- Barra de herramientas de acceso rápido.
2.- Fichas.
3.- Barra de Título.
4.- Grupos.
5.- Minimiza/maximiza Cinta de opciones.
6.- Minimizar, maximizar y cerrar ventana.
7.- Menú Archivo.
8.- Cuadro de nombres.
9.- Botón para seleccionar todo.
10.- Lanzador de cuadros de diálogo.
11.- Barra de fórmulas.
12.- Columnas.
13.- Dividir.
14. Filas.
15.- Barra de desplazamiento horizontal y vertical.
16.- Desplazamiento entre hojas.
17.- Etiquetas de hoja.
18.- Grabar macro.
19.- Barra de estado.
20.- Vistas.
21.- Zoom.

La interfaz está compuesta por una Cinta de opciones la cual a su vez se conforma por tres elementos que son: Fichas, Grupos y Comandos.

Las **Fichas** están en la parte superior de la cinta y cada una de ellas representa las tareas principales que se llevan a cabo en el programa.

Los **Grupos** son conjuntos de acciones relacionados ubicados en las diferentes fichas. Estos agrupan todos y cada uno de los comandos necesarios para llevar a cabo una tarea, permanecen expuestos y disponibles.

Los **Comandos** se encuentran agrupados con base a una tarea y/o actividad en común. Un comando puede ser un botón, un menú o un cuadro en el que debe especificar cierta información.

Barras y Cintas de opciones.

Barra de Título.

Se encuentra en la parte superior del archivo y aquí se muestra el nombre del libro con el cual está trabajando. Cuando un libro no ha sido guardado el nombre será el predeterminado, **Libro1**, en caso de tener más de un libro abierto y sin guardar, el número será consecutivo **Libro2**, etc., este cambiará automáticamente por el nombre que le asigne al guardar.

Botones de Minimizar, Maximizar y Cerrar.

Se encuentra en la barra de título en la parte derecha aparecen los íconos de minimizar ventana, maximizar/restaurar la ventana y cerrar la ventana (Estos tres íconos son típicos de Windows).

Menú Archivo.

Se encuentra debajo de la barra de título o estado, esta barra contiene todos los comandos de la aplicación agrupados en nueve menús. Los cuáles son: **Archivo**, **Inicio**, **Insertar**, **Diseño de página**, **Fórmulas**, **Datos**, **Revisar** y **Vista**.

Barra de Herramientas de acceso rápido.

Microsoft Excel cuenta con barras de herramientas predefinidas que contiene botones que proporcionan un acceso fácil y rápido a los comandos de uso más frecuente.

Se puede personalizar seleccionando el botón deseado de cualquier grupo y ficha. Con el botón derecho del mouse seleccionar: **Agregar a la Barra de herramientas de acceso rápido**.

Esta barra por defecto contiene 4 botones **Guardar**, **Deshacer escritura**, **Repetir escritura** y **Personalizar barra de herramientas de acceso rápido**, pero puede agregar más comandos, para esta acción basta que dé clic en el botón para personalizar el cual está ubicado al final de la barra, debe seleccionar las opciones requeridas, en caso de

que no se encuentre la opción, dé clic en **Más comandos**, se abrirá un cuadro de diálogo en donde se muestran todos los comandos de la aplicación, como se muestra en la figura 1.2.

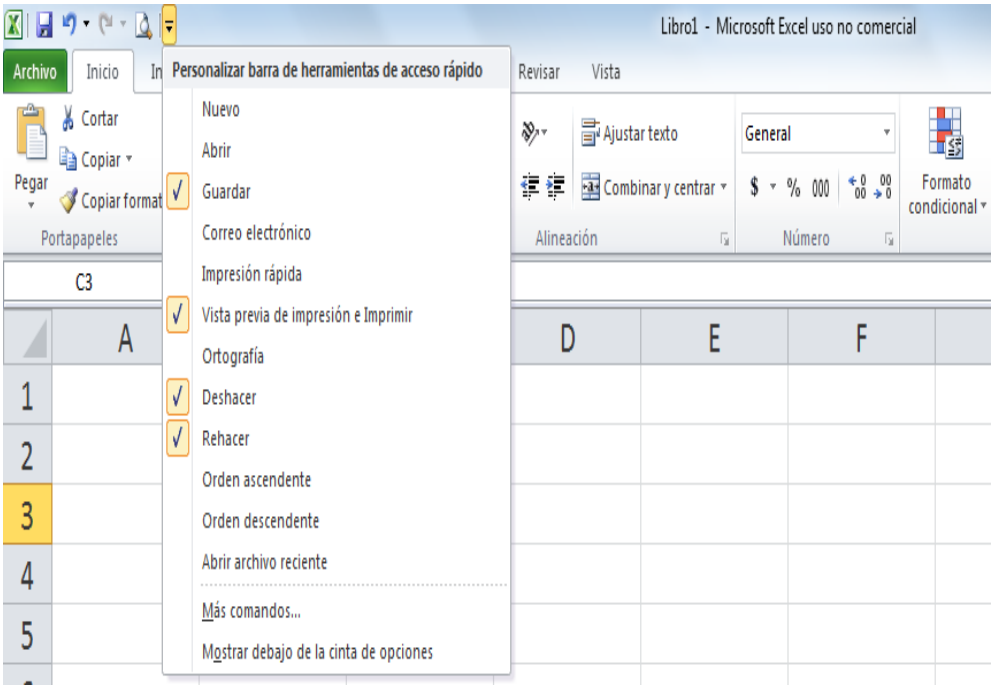


Figura 1.2. **Personalizar barra de herramientas de acceso rápido**.

Lanzador de cuadro de diálogo.

Son pequeños botones ubicados en la esquina inferior derecha del grupo, abren un cuadro de diálogo y proporcionan las opciones extra en relación a la función del grupo desde donde lo está abriendo. No todos contienen lanzador de cuadro de diálogo.

Se denomina cuadro de diálogo porque abre una ventana exclusiva que contiene información específica para el usuario, permitiendo así un diálogo (entre el sistema y el usuario) figura 1.3 y figura 1.4.

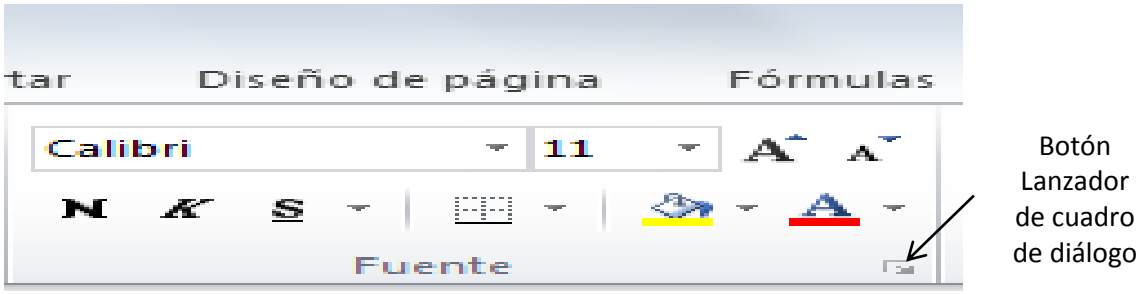


Figura 1.3. **Lanzador de cuadro de diálogo** del grupo formato.

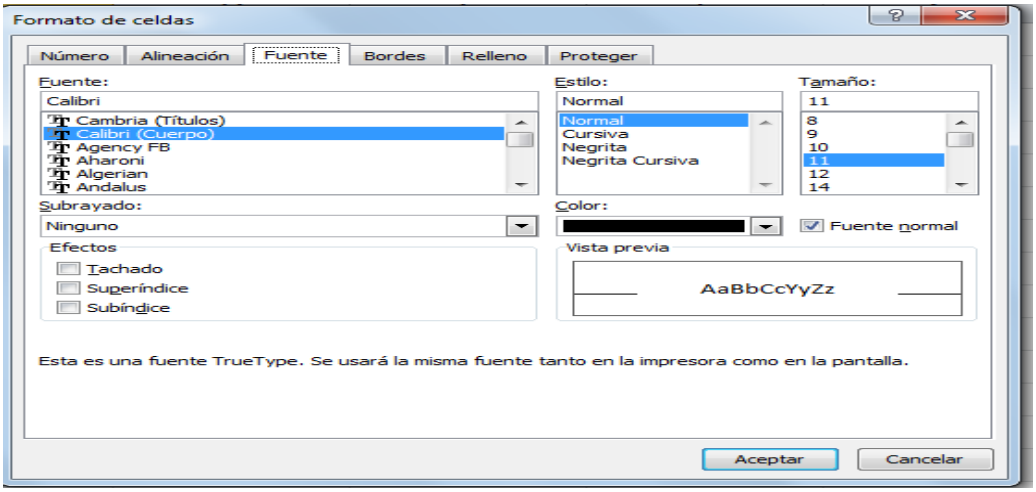


Figura 1.4 Cuadro de diálogo del **Formato celdas**.

Cinta de opciones.

Esta contiene las siguientes fichas: **Archivo**, **Inicio**, **Insertar**, **Diseño de página**, **Fórmulas**, **Datos**, **Revisar** y **Vista**.

La cinta de opciones puede mostrarse de manera completa o bien, ocultarse. En este estado sólo se observan las diferentes fichas. Cuando está maximizada se muestran las herramientas de la ficha seleccionada. Para lograr maximizarla o minimizarla seleccione el botón que se encuentra al final de la **cinta de opciones** y seleccionar el estado necesario, o bien presionar las teclas **Ctrl + F1**.

- **Ficha Inicio**.- Contiene los comandos: **Copiar**, **Pegar**, **Fuente**, **Alineación**, **Número**, **Estilos**, **Celdas**, etc.

- **Ficha Insertar**.- Contiene las herramientas necesarias para insertar diversos objetos: **Imágenes**, **Gráficos**, **Tablas dinámicas**, etc.

- **Ficha Diseño de página**.- Contiene las herramientas para configurar los **Márgenes**, **Configurar página**, **Ajustar área de impresión**, **Temas**, etc.

- **Ficha Fórmulas**.- Se pueden ubicar todas las opciones para el trabajo con funciones: **Lógicas**, **Texto**, **Fecha y hora**, **Financieras**. Y para su revisión **Auditoría**, **Administrador de nombres**, **Funciones**, **Cálculo**.

- **Ficha Datos**.- En Excel una de las funciones más utilizadas es el manejo de información a través de Lista de datos y para ello se pueden aplicar **Filtros**, **Subtotales**, **Ordenar información**, etc.

- **Ficha Revisar**.- Se encuentran las herramientas tradicionales de revisión tales como: **Ortografía**, **Sinónimos**. También puede agregar un **Nuevo comentario**, **Proteger hojas**, **Proteger libros**, etc.

- **Ficha Vista**.- Existen diversas vistas: **Diseño de página**, **Vista previa de salto de página**, **Zoom**, **Organizar ventanas**, etc.

Cuadro de nombres

Tiene varias funciones, principalmente:

- Mostrar ubicación de la celda activa, indicando la columna y la fila.
- Mover la celda activa a una referencia. Al escribir una referencia y presionar la tecla **Enter**, la celda activa se desplazará inmediatamente a esta ubicación.
- Seleccionar un rango. Por ejemplo escriba el rango A1:E7 dentro del **Cuadro de nombres** y presione la tecla **Enter**, automáticamente seleccionará este grupo de celdas.
- Nombrar a un grupo de celdas. Seleccione las celdas (figura 1.5), escriba el nombre en el **Cuadro de nombres** (figura 1.6) y presione **Enter**. Este procedimiento resulta benéfico al utilizar fórmulas o funciones, ya que solamente debe escribir el nombre dentro de la sintaxis de la fórmula y así evitará la selección del rango.

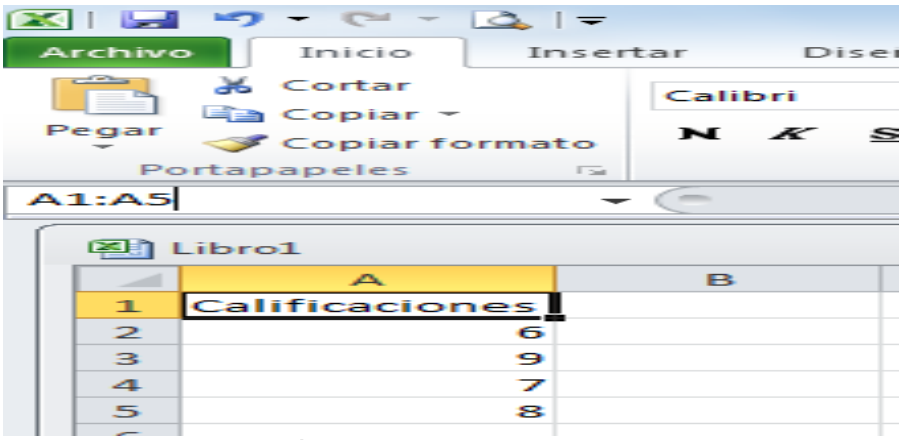


Figura 1.5. Selección de un rango en el **Cuadro de nombres**.

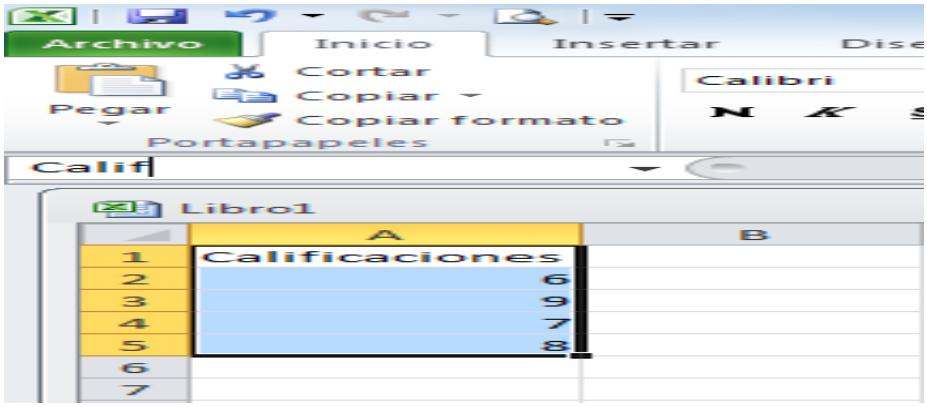


Figura 1.6. Nombrar a un grupo de celdas.

Barra de Fórmulas.

Muestra el contenido de la celda activa. En esta barra también podemos hacer modificaciones al texto de la celda activa. Al comenzar la captura de información, su apariencia muestra el botón cancelar, el cual es equivalente a presionar la tecla **Esc** y el botón introducir, es equivalente a presionar la tecla **Enter**. Figura 1.7.

Insertar función.
El botón insertar función, activa el cuadro de diálogo del mismo nombre, permitiendo seleccionar e insertar alguna función. Figura 1.7.

Figura 1.7. Botón cancelar, botón introducir, botón insertar función.

Etiquetas de hoja.

Excel tiene una gran capacidad para el trabajo con grandes volúmenes de información, especialmente listas o datos. El límite para insertar hojas de cálculo dentro de un libro lo determina la configuración de hardware del equipo, es decir, la capacidad de memoria RAM y del disco duro.

Barra de desplazamiento.

Puede utilizar la **Barra de desplazamiento** vertical y horizontal, para desplazarse a lo largo del libro de trabajo de forma rápida y sencilla, también le permite cambiar el tamaño de la barra horizontal así como dividir la hoja en forma horizontal y vertical. Figura 1.8.

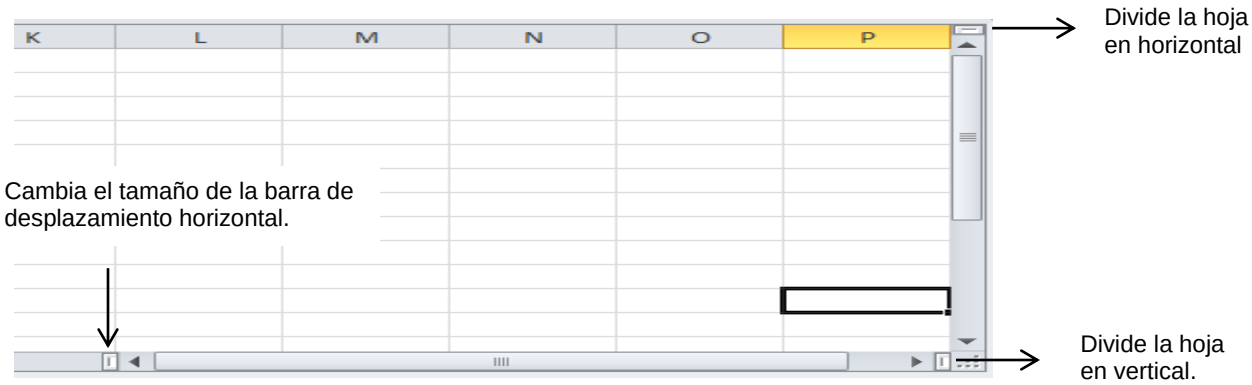


Figura 1.8. **Barra de desplazamiento.**

Barra de Estado.

Está situada en la parte inferior de la barra de edición, muestra los mensajes que corresponden a lo que se está realizando. Sus elementos se muestran en la figura 1.9.

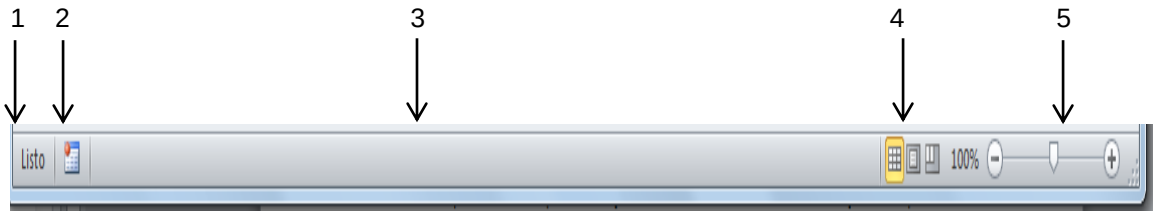


Figura 1.9. **Barra de estado**.

1.- Existen cuatro estados de este botón:

Listo. Significa que Excel puede comenzar a utilizarse.
Modificar: Puede editar el contenido de una celda.
Introducir: Debe volver a presionar **Enter** o la tecla introducir.
Señalar: Puede indicar la celda requerida dentro de una fórmula.

2.- Comienza el proceso para grabar macros al momento de presionarlo.

3.- Zona de cálculo automático. Puede activar **Suma**, **Máximo**, **Mínimo**, **Promedio**, etc. Solamente dé clic en esa zona con el botón derecho.

4.- Vistas, la opción predeterminada es **Vista normal**, con la cual puede trabajar de forma sencilla. **Diseño de página**, es una opción para revisar la información como si estuviera en Word y **Vista previa**, **salto de página** le ayuda a configurar los saltos de página.

5.- Zoom, controla el nivel para observar más de cerca todo el documento o por el contrario verlo más pequeño, sin afectar el tamaño de la fuente.

Guardar libros de trabajo.

Es importante guardar los archivos creados, ya que de lo contrario la información podría perderse de manera total. Para guardar un archivo por primera vez realice los siguientes pasos:

1.- Seleccione el Menú **Archivo** dé clic en **Guardar como**.
2.- Se abrirá un cuadro de diálogo con las opciones de **Nombre del archivo**, **Ubicación** y **Tipo**. Agregue cada elemento y dé clic en **Guardar**. Figura 1.10.

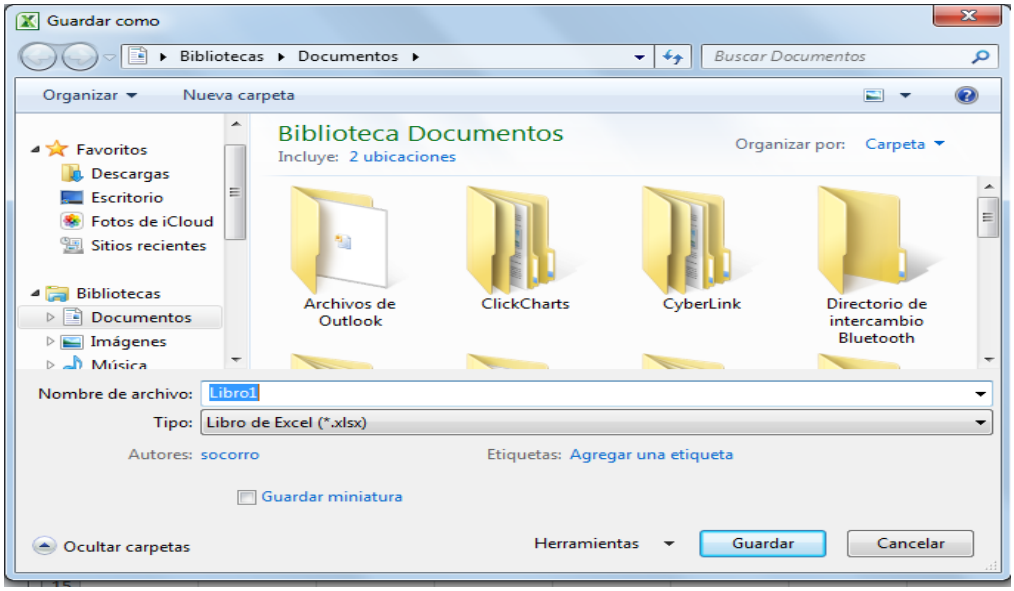


Figura 1.10. Cuadro de diálogo **Guardar como**.

1.4 Fórmulas y funciones.

Fórmulas

Una de las principales características de la hoja de cálculo, es la de operar con los datos introducidos en ella, relacionándolos por medio de fórmulas.

Una fórmula habitualmente está compuesta por números, operadores y direcciones de celdas. También se pueden incluir paréntesis y funciones. En la tabla 1.2 se incluyen los operadores utilizados para crear fórmulas.

Tabla 1.2. Operadores que se utilizan para crear una fórmula.

OPERADOR	DESCRIPCIÓN	PRIORIDAD
+, -	Positivo, Negativo	Mayor
%	Porcentaje	
^	Exponenciación	
*, /	Multiplicación, División	
+, -	Suma, Resta	
&	Concatenación de texto	
=, <>, <, >, <= y >=	Igual, Distinto, Menor, Mayor, Menor o igual, Mayor o igual	Menor

Si en una misma fórmula existe un operador de suma y otro de división, primero se efectuará la división y luego la suma. Si existen varios operadores con la misma prioridad (por ejemplo sumas y restas), éstos se evalúan en el orden en que se encuentran de

izquierda a derecha dentro de la fórmula. Si se desea alterar el orden de evaluación de los operadores se puede hacer uso de los paréntesis.
Todas las fórmulas inician con el signo de igual (=). De lo contrario Excel lo interpretará como texto. En la tabla 1.3 se incluyen algunos ejemplos de fórmulas válidas.

Tabla 1.3. Ejemplos de fórmulas.

FÓRMULA	**EVALUACIÓN**
=10*3	El resultado de la fórmula será 30.
=B3*25%	Calcula el 25% del contenido de la celda B3.
=E6-F6+G6	Resta el contenido de la celda F6 al de E6, y suma el resultado al de la celda G6.
=C8*(A8+B8)	Suma el contenido de la celda A8 con el de B8, y multiplica el resultado con el de C8.
=C12+B12^5	Eleva el contenido de la celda B12 a la quinta potencia, y suma el resultado al contenido de la celda C12.
=BH100^(1/3)	Calcula la raíz cúbica del contenido de la celda BH100
=M23&”enero”	Concatena el texto contenido en la celda M23 con el texto enero (los textos dentro de una fórmula van entre comillas).
=”5/12/03”-“3/12/03	Resta las dos fechas y devuelve el número de días entre ambas fechas (como existe un signo de resta, las comillas no indican texto).
=D17<=200	Muestra verdadero si el contenido de D17 es menor o igual que 200, y falso en caso contrario.

Las fórmulas pueden mostrar errores, los cuales son causados por una mala redacción de las mismas, selección incorrecta de celdas, operadores mal ubicados o desconocidos, etc.

Las principales causas de errores son: Omisión del signo igual, operadores mal ubicados o no agregados, redacción errónea de funciones, uso de comillas inadecuado, uso inadecuado de paréntesis, referencias mal escritas y dividir entre cero.
Dentro de las celdas los errores se pueden presentar como lo muestra la tabla 1.4.

Tabla 1.4. Errores en Excel.

La celda muestra	**Significa**
#####	El ancho de una columna no es suficiente para mostrar todo el contenido o se utilizó un número negativo para la fecha u hora de una celda.
#¡DIV/0!	Cuando un número se divide por cero o por una celda que no contiene ningún valor.
#N/A	Un valor determinado no está disponible para una función o fórmula.
#¿NOMBRE?	No se reconoce el texto especificado en la fórmula.

#¡NUM!	La fórmula o función contiene valores numéricos no válidos.
#¡REF!	La referencia hacia una celda no es válida.
#¡VALOR!	La fórmula contiene celdas con diferentes tipos de datos.

Por ejemplo si escribiéramos =20/0, una regla esencial indica que no es posible dividir nada entre cero, por lo tanto el resultado es un error. Este error se muestra en la figura 1.11.

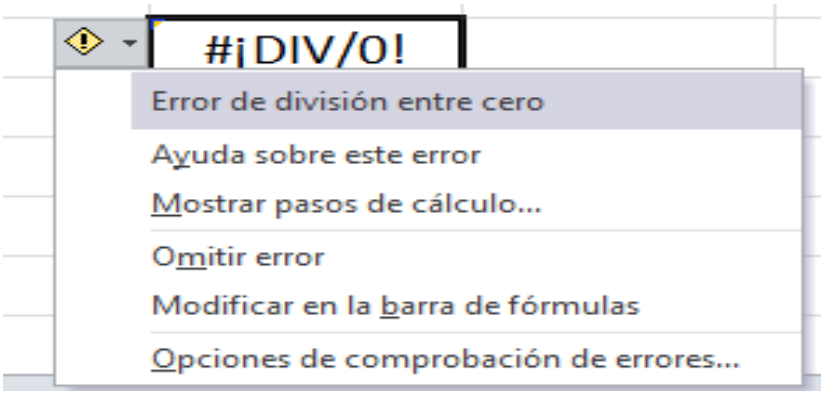


Figura 1.11. División entre cero.

Para desactivar o activar la revisión de errores realice los siguientes pasos:

1. Dé clic en el menú **Archivo**.
2. Seleccione **Opciones**.
3. En la parte de **Fórmulas**, localice la sección, **Reglas de verificación de Excel**.
4. Se muestra una lista de opciones para desactivar o activar los errores comunes.
5. Active o desactive las opciones mostradas.

Para configurar el color del indicador de errores detectados en las fórmulas, realice los siguientes pasos:

1. En el menú de **Archivo** seleccione Opciones.
2. Dé clic en **Fórmulas**.
3. En la sección **Comprobación de errores**.
4. Observe la casilla **Habilitar comprobación de errores en segundo plano**, este activada.
5. Seleccione el color de su preferencia con el botón Color. Acepte.

Auditoría de fórmulas.

Esta herramienta sirve para determinar las celdas involucradas en una fórmula o función. Así como revisar los posibles errores dentro de las fórmulas. Los elementos del grupo Auditoría de fórmulas se muestran en la figura 1.12.

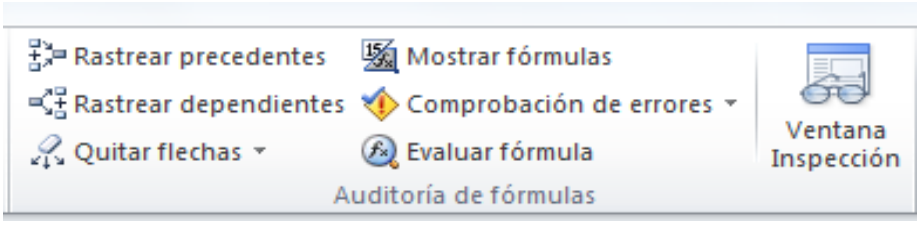


Figura 1.12. **Auditoría de fórmulas**.

Rastrear precedentes.- Muestra flechas que indican la referencia de una fórmula a una o varias celdas. Esta herramienta se encarga de buscar otras fórmulas o celdas cuyo resultado da origen a la fórmula en la celda actual; es decir, en la celda donde ejecutó el comando **Rastrear precedentes**.

Rastrear dependientes.- Son las celdas con una fórmula la cual tiene valores ubicados en otra celda. Esta herramienta se encarga de buscar otras fórmulas o celdas que dependen del resultado de la fórmula en la celda actual, es decir, en la celda donde ejecutó el comando **Rastrear dependientes**.

Quitar flechas.- Retira todas las flechas en la hoja de Excel que se mostraron para rastrear precedentes o dependientes. Lo puede hacer con el botón **Quitar flechas**, ubicado en la ficha **Fórmulas**, grupo **Auditoría de fórmulas**.

Mostrar fórmulas.- Las fórmulas se pueden observar solamente en la **Barra de fórmulas** de la celda activa. Sin embargo ésta herramienta es muy útil toda vez que permite observar todas las fórmulas aplicadas en pantalla.

Evaluar fórmula.- Esta herramienta revisa fórmulas de la celda activa, verificando las referencias utilizadas así como los valores y operadores. Se suele aplicar para fórmulas muy largas o complejas. Para utilizarla seleccione el botón **Evaluar fórmula**, del grupo **Auditoría de fórmulas** en la ficha **Fórmulas**.

Para su uso realice los siguientes pasos:

1. Seleccione la celda para evaluar la fórmula.
2. En la ficha **Fórmulas**, grupo **Auditoría de fórmulas**, seleccione **Evaluar fórmula**.
3. Se mostrará un cuadro de diálogo con la fórmula seleccionada, dé clic en **Evaluar**, examinará el valor de la referencia, el resultado se observará en cursiva.
4. En caso de que la parte subrayada de la fórmula pertenezca a otra fórmula, dé clic en **Paso a paso para entrar**, esto mostrará la siguiente fórmula.

Al localizar el error, cierre el cuadro de diálogo y realice la modificación en la **Barra de fórmulas**.

Para evaluar el conocimiento deberá realizar el ejercicio 4 de la página 36.

Referencias en las fórmulas.

Una fórmula puede contener distintos elementos, entre ellos referencias. Una referencia identifica una celda o rango de celdas en una hoja de cálculo. Las referencias dirigidas a celdas de otros libros se denominan vínculos o referencias externas.

En Excel podemos utilizar referencias relativas o absolutas. Incluso se pueden mezclar y hacer una referencia mixta. Para lograrlo es necesario utilizar el carácter que usualmente se conoce como símbolo de pesos $.

En la figura 1.13 muestra un ejemplo de referencia relativa, donde se utilizan celdas de la misma hoja.

	A	B	C
1	12	22	=A1+B1

Figura 1.13. Referencia relativa.

En la figura 1.14 se muestra un ejemplo de referencia absoluta, en donde se puede ver el contenido absoluto ubicado dentro de la celda señalada. Se distingue por el signo de $.

	A	B	C	D
1	12	22	=A1+B1*D1	8
2	16	90	=A2+B2*D1	

Figura 1.14. Referencia absoluta.

Funciones.

Una función es una fórmula predefinida en Excel, las funciones constan de dos partes. El nombre de la función y los argumentos de ésta.

Los argumentos se escriben a continuación del nombre de la función entre paréntesis y pueden estar constituidos por simples valores o por direcciones de celdas. En caso de necesitar de varios argumentos, éstos irán separados por una coma (,).

La sintaxis es: =FUNCIÓN(dato)

Los siguientes ejemplos dan una idea de las amplias posibilidades que ofrece la utilización de funciones:

=SUMA(B5:B10,C5:C10): suma el contenido de todas las celdas del rango **B5:B10** junto con las del rango **C5:C10**.

=C10 – SUMA(B5:B9)*15%: al contenido de **C10** le resta el 15% de la suma del contenido del rango **B5:B9**.

Hay una gran diversidad de funciones, existen alrededor de 400 funciones divididas en diferentes categorías.

Categorías de Funciones

Excel agrupa las funciones en categorías de acuerdo al tipo de cálculo que realicen. Existen las siguientes categorías:

- **Financieras:** Realizan cálculos sobre anualidades, inversiones, etc.
- **Fecha y Hora:** Realizan cálculos que representan fechas y horas.
- **Matemáticas y trigonométricas:** Realizan cálculos matemáticos y trigonométricos.
- **Estadísticas:** Realizan análisis estadísticos.
- **Búsqueda y referencia:** Localizan celdas en función de su contenido.
- **Base de datos:** Ejecutan cálculos estadísticos en bases de datos.
- **Texto:** suministran información sobre el texto contenido en las celdas y realizan otras operaciones con cadenas de texto.
- **Lógicas:** Calculan el resultado de fórmulas lógicas.
- **Información:** Devuelven información sobre celdas y rangos de celdas.

Dado el gran número de funciones que incluye Excel, la inclusión de una simple lista de las mismas, resultaría tedioso y poco práctico; sobre todo si se tiene en cuenta que la ayuda del programa incluye una excelente información de cada una de ellas.

Argumentos

Como ya se mencionó, los argumentos son los parámetros que necesita una función para realizar el cálculo correspondiente. Los argumentos pueden ser números, textos, celdas, rangos, fórmulas e incluso otras funciones. Según sea el caso deberá tener en cuenta lo siguiente:

- Cuando el argumento sea un texto, éste debe ir entre comillas, por ejemplo **=LARGO(“OAXACA”)**.

- Si el argumento es una fórmula, no es necesario anteponer el signo igual, por ejemplo **=RAIZ(A1*A3)**.

- Si el argumento es otra función, los paréntesis de esta última deben quedar anidados dentro de los paréntesis que incluyen los argumentos de la primera, por ejemplo **=RAIZ(SUMA(B1:B5))**.

- En cualquier caso, al introducir una función debe respetar tanto el número de argumentos que le corresponden como su orden.

Para utilizar una función de Excel se debe extraer de la **Biblioteca de funciones** de la siguiente manera:

1. Seleccione la celda donde desea insertar la función.
2. Seleccione el comando **Insertar Función**. Hay tres lugares distintos donde lo puede ubicar:
    a. En la **Barra de fórmulas**.
    b. En la ficha **Inicio**, en el grupo **Modificar**, en el comando **Autosuma** y **Más funciones**.
    c. En la ficha **Fórmulas**, dentro del grupo **Biblioteca de funciones**.
3. Deberá seleccionar la función requerida. Puede elegir alguna de las **Usadas recientemente** o bien, desplegar la lista **Insertar una función** y buscar la requerida.
4. Agregue los datos requeridos y acepte.

De acuerdo a la función que haya seleccionado, en algunas ocasiones le solicitará agregar condiciones, rangos adicionales, valores, etc.

Al dar clic en el botón **Insertar función**, aparece un cuadro de diálogo como el que se muestra en la figura 1.15.

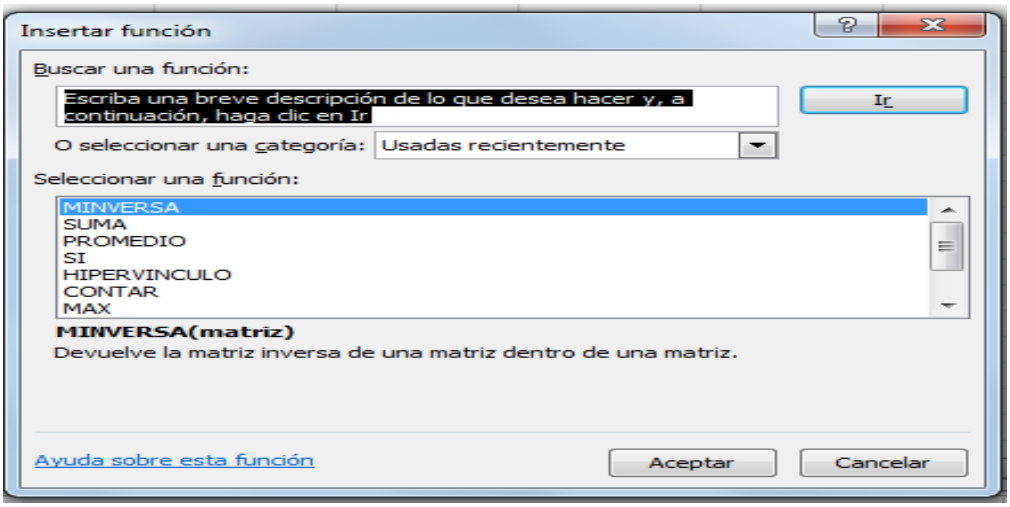


Figura 1.15. Cuadro de diálogo **Insertar función**.

A continuación se mencionan las funciones más utilizadas:

Función PROMEDIO.- Esta función obtiene el promedio (media aritmética) del grupo de celdas seleccionado. La redacción y argumentos son:

=PROMEDIO(número1,[número2],…)

Donde número1 es obligatorio, debe incluir el rango de celdas las cuales contienen los valores a promediar. Puede utilizar diversos rangos hasta un máximo de 255. Ejemplo =PROMEDIO(A1:A35).

Función CONTARA.- Esta función cuenta la cantidad de celdas ocupadas con cualquier tipo de dato. La redacción y argumentos son:

=CONTARA(valor1,[valor2],…)

Donde valor1 es obligatorio. Rango el cual contiene la información a contar. El valor2 es opcional. Rangos adicionales que representan los valores que desea contar, hasta un máximo de 255 argumentos.

Función CONTAR.- Esta función obtiene la cantidad de celdas que contienen datos con números. La redacción y argumentos son:

=CONTAR(valor1;[valor2],…)

Donde valor1 es obligatorio, primer elemento en el que desea contar números, referencia de celda o rango. El valor2 es opcional, hasta 255 elementos, celdas de referencia o rangos adicionales en los que desea contar números.

Para evaluar el conocimiento deberá realizar el ejercicio 5 de la página 37.

Función CONTARA.SI.- Cuenta el número de celdas con algún contenido siempre y cuando cumplan una condición. La redacción y argumentos son:

=CONTARA.SI(rango, criterio)

Donde rango es obligatorio y es el conjunto de celdas con datos a considerar en el conteo. Los valores en blanco no se consideran. El criterio es obligatorio y determina las celdas que se van a contar, es la condición. Los criterios no distinguen entre mayúsculas y minúsculas. En la figura 1.16 se puede observar el uso de esta función.

E3 | fx =CONTAR.SI(B2:B10,"FEMENINO")

	A	B	C	D	E	F
1	Estudiantes	Género				
2	Alma	femenino		Género	Total	
3	Susana	femenino		femenino	6	
4	Daniel	masculino		masculino	3	
5	Rafael	masculino		Total	9	
6	Regina	femenino				
7	Ingrid	femenino				
8	Juan	masculino				
9	Mariana	femenino				
10	Diana	femenino				
11						

Figura 1.16. Función CONTAR.SI.

Para evaluar el conocimiento deberá realizar el ejercicio 6 de la página 37.

Función SUMAR.SI.- Esta función se utiliza para sumar los valores en un rango que cumplen con criterios específicos. La redacción y argumentos son:

=SUMA.SI([rango, criterio, [rango_suma)]

Donde rango es el grupo a considerar. El criterio en forma de número, expresión o texto, que determina las celdas que se van a sumar, puede incluir caracteres. El rango_suma son las celdas a sumar. En la figura 1.17 se puede observar el uso de esta función.

=SUMAR.SI(C2:C8,"norte",E2:E8)

C	D	E
Zonas	**Vendedores**	**Ventas**
norte	Camila	$100,000.00
sur	Raúl	$155,000.00
norte	Nicólas	$250,000.00
sur	Andrea	$100,000.00
sur	Viridiana	$500,000.00
este	Saúl	$670,000.00
norte	Samuel	$100,000.00
ventas zona norte	$450,000.00	

Figura 1.17. Función SUMAR.SI.

Para evaluar el conocimiento deberá realizar el ejercicio 7 de la página 38.

Función PROMEDIO.SI._ Busca la media aritmética de un rango de celdas que cumplan con una condición o criterio. La redacción y argumentos son:

=PROMEDIO.SI([rango, criterio,[rango_promedio])

Donde el rango es el grupo de celdas y deben ser números, o bien nombres, matrices o referencias que contengan números. El criterio en forma de número, expresión o texto, que determina las celdas que se van a sumar. Puede incluir caracteres. El rango_promedio es el grupo de celdas a promediar. En la figura 1.18 se puede observar el uso de esta función.

=PROMEDIO.SI(C2:C8,"norte",E2:E8)

C	D	E
Zonas	**Vendedores**	**Ventas**
norte	Camila	$100,000.00
sur	Raúl	$155,000.00
norte	Nicólas	$250,000.00
sur	Andrea	$100,000.00
sur	Viridiana	$500,000.00
este	Saúl	$670,000.00
norte	Samuel	$100,000.00
Promedio de ventas por zona	$150,000.00	

Figura 1.18. Función PROMEDIO.SI.

Función BUSCARH.- Esta función realiza una búsqueda de un valor específico dentro de la primera columna de una matriz de tabla y regresa, en la misma fila un valor de otra columna de dicha matriz de tabla. La redacción y argumentos son:

=BUSCARH(valor_buscado, matriz_ buscar_en, indicador_filas, ordenado)

Donde el valor_buscado es el valor buscado en la primera columna de la matriz de tabla. La matriz_buscar_en es el rango en donde se realizará la búsqueda. El indicador_filas es el número de columna de matriz_buscar_en, considerando como valores de matriz 1 para la primera fila, 2 para la segunda, etc. El ordenado especifica si BUSCARH va a buscar una coincidencia exacta o aproximada.

Para ejemplificar esta función, observe los siguientes datos:

Se pretende localizar la clave de cada producto de acuerdo a su tamaño.

Producto	Taza	Lapicera	Alcancía	Reloj
Tamaño	Grande	Mediana	Grande	Mediana
Clave				

En este caso se requiere de una matriz, o lista de donde obtener la información

Tamaño	Grande	Mediana
Clave	G-100	M-50

El resultado y la función quedarían como lo muestra la figura 1.19.

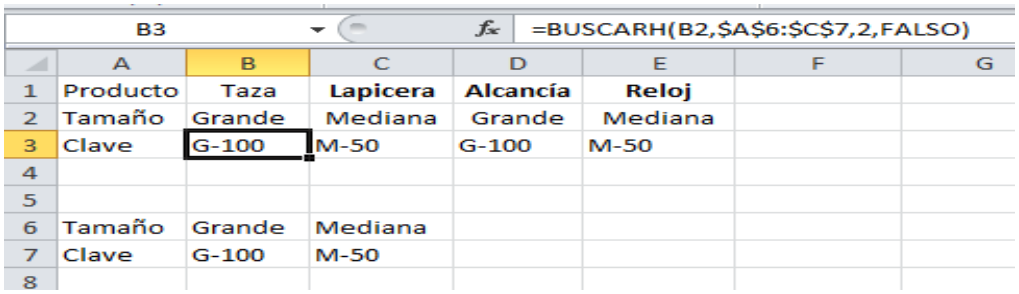


Figura 1.19. Función BUSCARH.

Función BUSCARV.- Busca los datos que están en la primera columna de una base de datos o tabla. En caso de encontrar el valor regresa el dato asociado. La redacción y argumentos son:

=BUSCARV(valor_buscado, matriz_buscar_en, indicador_columnas, ordenado)

Donde valor_buscado es el valor buscado en la primera columna de la matriz de tabla. La matriz_buscar_en es el rango donde se realizará la búsqueda. El indicador_columnas es el número de columna de matriz_buscar_en, considerando como valores de matriz 1 para

la primera columna, 2 para la segunda, etc. El ordenado especifica si BUSCARV va a buscar una coincidencia exacta o aproximada.

Para ejemplificar esta función observe los siguientes datos: Se pretende localizar la Clave de cada producto de acuerdo a su Precio.

Producto	Clave	Precio
Taza		$ 10.00
Lapicera		$ 25.00
Alcancía		$ 25.00
Reloj		$ 10.00

En este caso se requiere de una matriz, o lista de donde obtener la información.

Precio	Clave
$ 10.00	G-100
$ 25.00	M-50

En la figura 1.20 se presenta el cuadro de diálogo de los argumentos de la función BUSCARV.

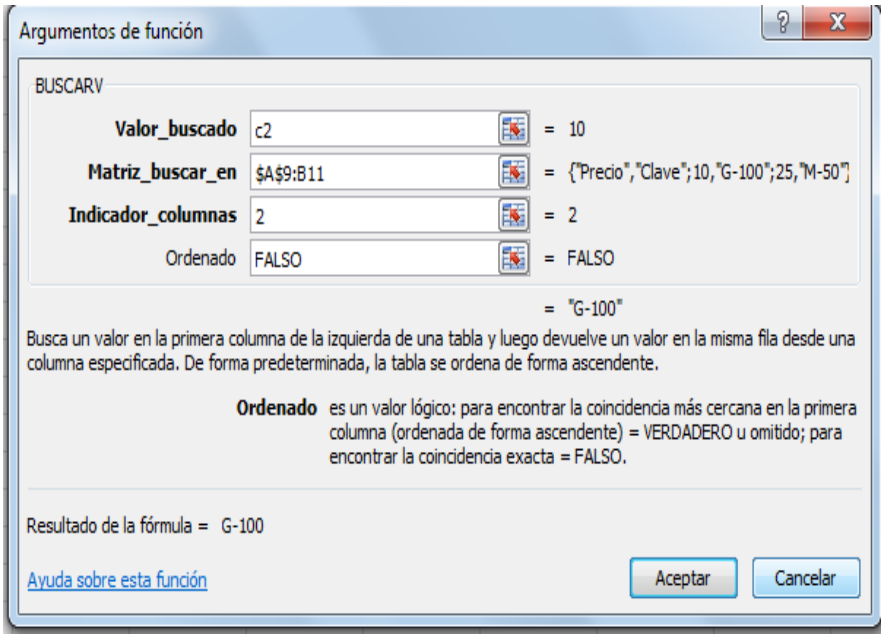


Figura 1.20. Cuadro de diálogo **Argumentos de función**.

El resultado y la función quedarían como la figura 1.21.

	A	B	C
1	Producto	Clave	Precio
2	Taza	G-100	$10.00
3	Lapicera	M-50	$25.00
4	Alcancía	M-50	$25.00
5	Reloj	G-100	$10.00
6			
7			
8			
9	Precio	Clave	
10	$10.00	G-100	
11	$25.00	M-50	
12			

Figura 1.21. Función BUSCARV.

Para evaluar el conocimiento deberá realizar el ejercicio 8 y 9 de las páginas 39 y 40.

Función SI.- Devuelve un valor si la condición indicada se cumple y otro valor si dicho argumento no se cumple. Se usa la función SI para realizar pruebas condicionales en valores y fórmulas. También se puede emplear para obtener un valor, o texto si se cumple con una condición. La redacción y argumentos son:

=SI(prueba_lógica, valor_si_verdadero, valor_si_falso)

Donde prueba_lógica, establece la condición comparando cualquier valor o expresión utilizando los operadores mayor que (>) menor que (<) o igual que (=), diferente (<>), etc. El valor_si_verdadero es el valor o texto que regresará la función en caso de cumplirse la condición. El valor_si_falso se refiere al valor que se devuelve si el argumento no se cumple.

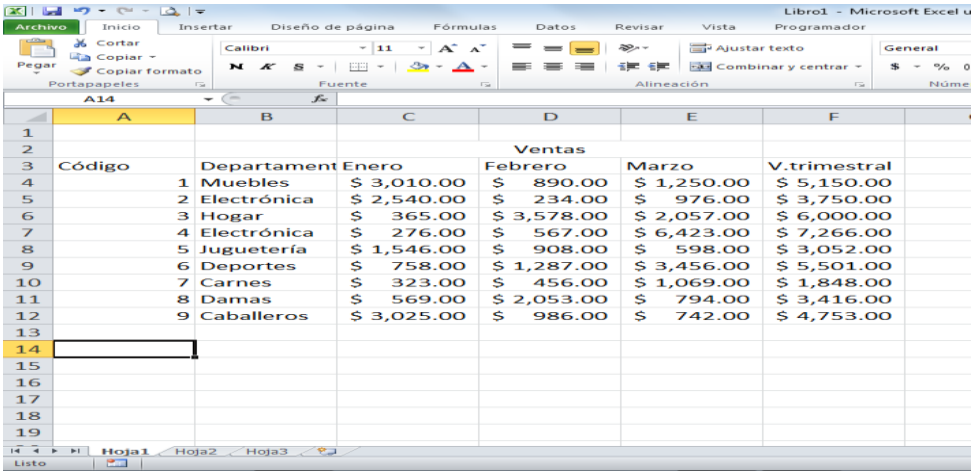
Para evaluar el conocimiento deberá realizar el ejercicio 10 de la página 41

1.5 Tablas dinámicas.

A partir de una Base de Datos, Excel es capaz de generar resúmenes de tipo estadístico que reciben el nombre de Tablas Dinámicas. De esta forma, se elaboran resúmenes de un campo de la Base de Datos, en función de los distintos valores que tomen otros dos, los cuales se representarán en las celdas de la primera fila y primera columna de la tabla respectivamente. (GRUPO NORIEGA EDITORES, 2000)

Con un informe de Tabla dinámica puede resumir, analizar, explorar y presentar un resumen de los datos de la hoja de cálculo o un origen de datos externos. Un informe de Tabla dinámica es especialmente útil cuando tiene una larga lista de cifras para sumar y los datos agregados o subtotales podrían servir para mirar los datos desde perspectivas diferentes y comparar las cifras de datos similares.

Para ejemplificar las Tablas dinámicas crearemos una base de datos (ver figura 1.22).



	A	B	C	D	E	F
1						
2				Ventas		
3	Código	Departament	Enero	Febrero	Marzo	V.trimestral
4	1	Muebles	$ 3,010.00	$ 890.00	$ 1,250.00	$ 5,150.00
5	2	Electrónica	$ 2,540.00	$ 234.00	$ 976.00	$ 3,750.00
6	3	Hogar	$ 365.00	$ 3,578.00	$ 2,057.00	$ 6,000.00
7	4	Electrónica	$ 276.00	$ 567.00	$ 6,423.00	$ 7,266.00
8	5	Juguetería	$ 1,546.00	$ 908.00	$ 598.00	$ 3,052.00
9	6	Deportes	$ 758.00	$ 1,287.00	$ 3,456.00	$ 5,501.00
10	7	Carnes	$ 323.00	$ 456.00	$ 1,069.00	$ 1,848.00
11	8	Damas	$ 569.00	$ 2,053.00	$ 794.00	$ 3,416.00
12	9	Caballeros	$ 3,025.00	$ 986.00	$ 742.00	$ 4,753.00

Figura 1.22. Base de datos.

Después dar clic en la pestaña **Insertar**, y en el ícono Tabla dinámica.

La figura 1.23 muestra un cuadro de diálogo **Crear tabla dinámica**, el cual le solicita introducir la tabla o rango. También seleccionará dónde desea colocar el informe de tabla dinámica, para este caso seleccionará **Nueva hoja de cálculo** y el botón de **Aceptar.** Podrá visualizar una pantalla como la que se muestra en la figura 1.24.

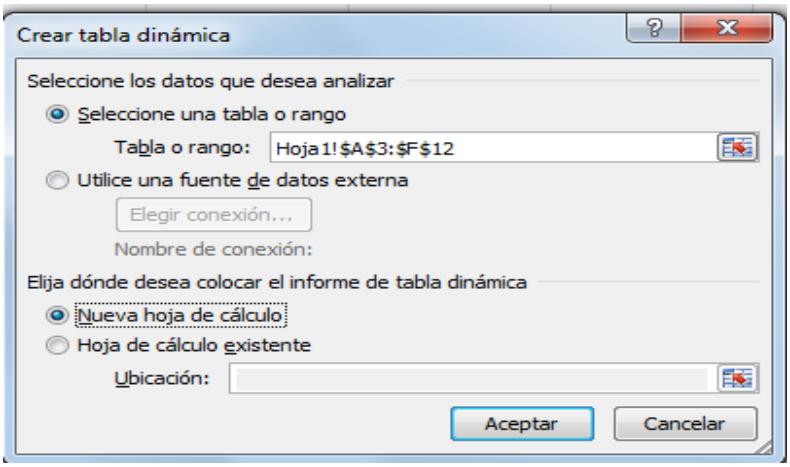


Figura 1.23. Cuadro de diálogo **Crear tabla dinámica.**

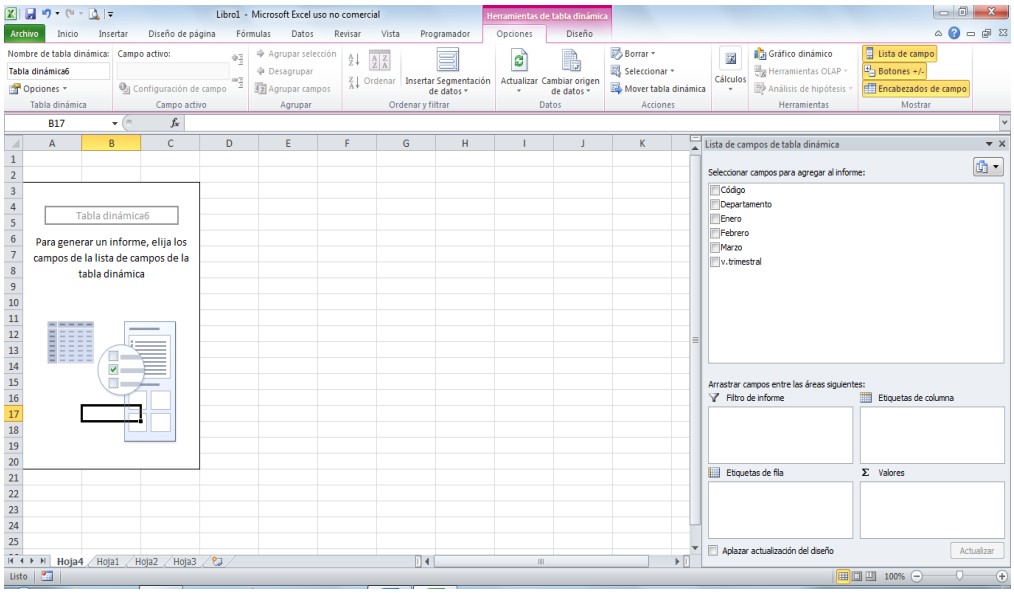


Figura 1.24. Pantalla de tabla dinámica.

Del lado derecho se encuentra la **Lista de campos de tabla dinámica**. En **Seleccionar campos para agregar al informe**, Dar clic a Código, Departamento, Enero, Febrero, Marzo y V. trimestral. La tabla dinámica quedará como lo muestra la figura 1.25.

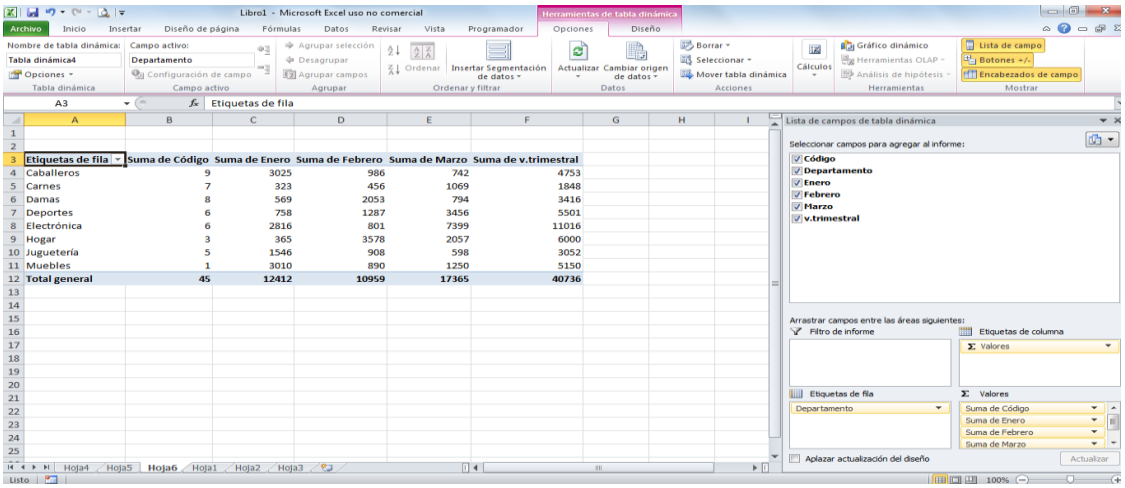


Figura 1.25. Tabla dinámica.

Dar clic en Código para eliminarlo de la Tabla dinámica. Para saber qué departamento vendió más en el mes de enero, realizará un ordenamiento de forma descendente. Dar clic en la celda donde se encuentra la primera cantidad del mes de enero. En el menú de Excel dar clic en la ficha **Herramientas de tablas dinámicas**, seleccione **Opciones** y después presione el ícono de la Z a la A (figura 1.26). La tabla quedará como lo muestra la figura 1.27.

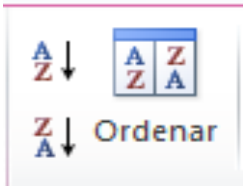
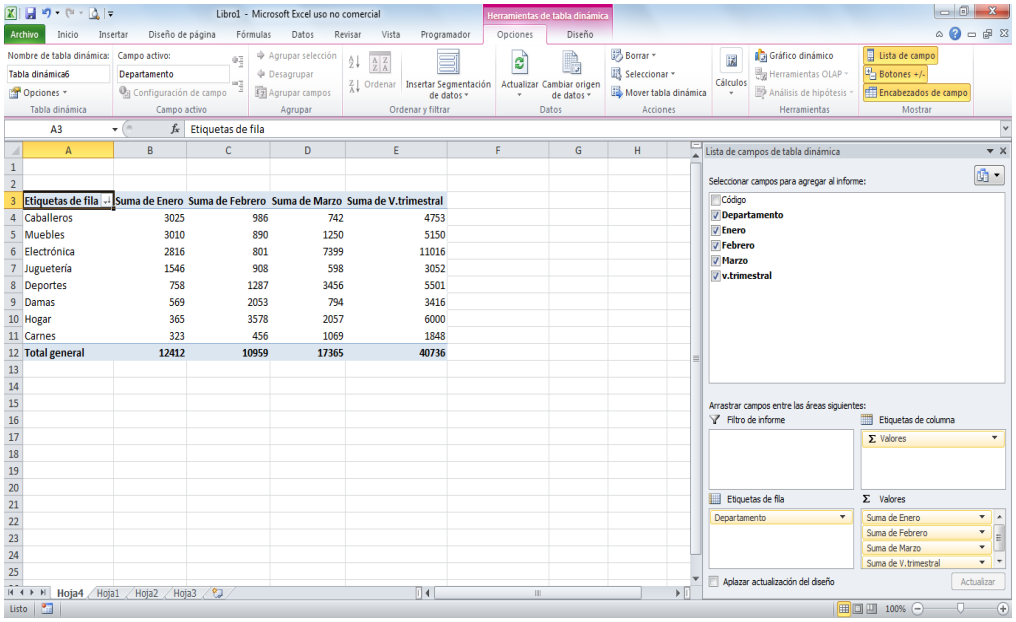


Figura 1.26. Botones para ordenar ascendente y descendente.

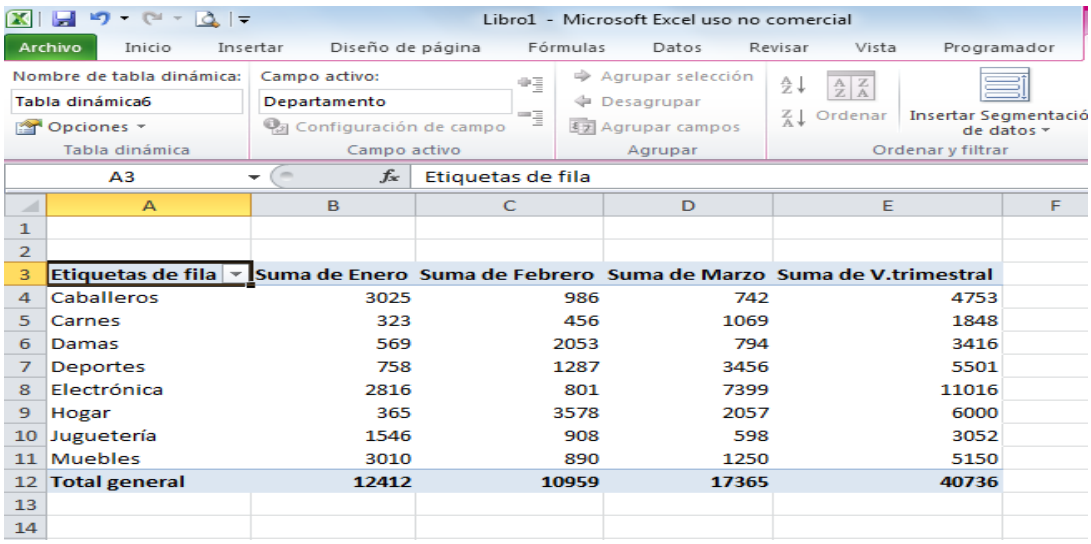


Etiquetas de fila	Suma de Enero	Suma de Febrero	Suma de Marzo	Suma de V.trimestral
Caballeros	3025	986	742	4753
Muebles	3010	890	1250	5150
Electrónica	2816	801	7399	11016
Juguetería	1546	908	598	3052
Deportes	758	1287	3456	5501
Damas	569	2053	794	3416
Hogar	365	3578	2057	6000
Carnes	323	456	1069	1848
Total general	**12412**	**10959**	**17365**	**40736**

Figura 1.27. Tabla dinámica ordenada ascendentemente.

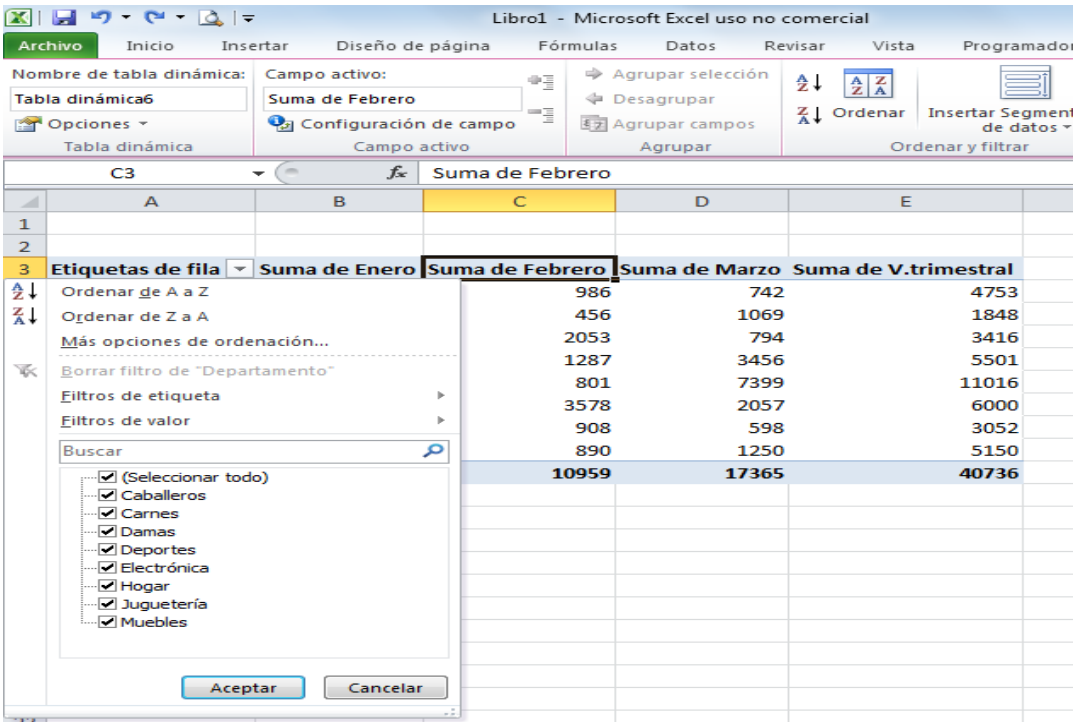
De esta manera se puede observar que el departamento de Caballeros tuvo el mayor número de ventas en el mes de enero.

Si lo que deseamos es saber qué departamento tuvo ventas menores de $1000.00 en el mes de febrero, entonces dar clic en la flecha que se encuentra del lado derecho de **Etiquetas de fila** de la tabla dinámica (figura 1.28). Aparecerá un submenú (figura 1.29).



Etiquetas de fila	Suma de Enero	Suma de Febrero	Suma de Marzo	Suma de V.trimestral
Caballeros	3025	986	742	4753
Carnes	323	456	1069	1848
Damas	569	2053	794	3416
Deportes	758	1287	3456	5501
Electrónica	2816	801	7399	11016
Hogar	365	3578	2057	6000
Juguetería	1546	908	598	3052
Muebles	3010	890	1250	5150
Total general	**12412**	**10959**	**17365**	**40736**

Figura 1.28. Etiquetas de fila.



Etiquetas de fila	Suma de Enero	Suma de Febrero	Suma de Marzo	Suma de V.trimestral
		986	742	4753
		456	1069	1848
		2053	794	3416
		1287	3456	5501
		801	7399	11016
		3578	2057	6000
		908	598	3052
		890	1250	5150
		10959	**17365**	**40736**

Figura 1.29. Submenú de Etiquetas de fila.

Seleccionar **Filtros de valor**, aparece una lista de opciones, seleccionar **Menor que**, como lo muestra la figura 1.30.

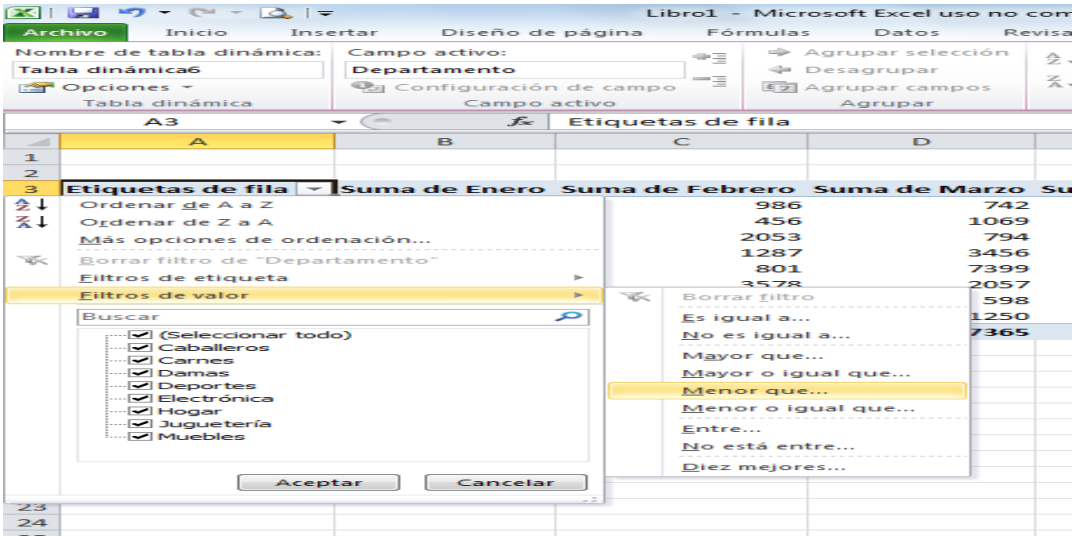


Figura 1.30. Lista de opciones.

Aparecerá un cuadro de diálogo Filtro por valor (Departamento), seleccionar en la flecha Suma de Febrero y escribir 1000 (figura 1.31).

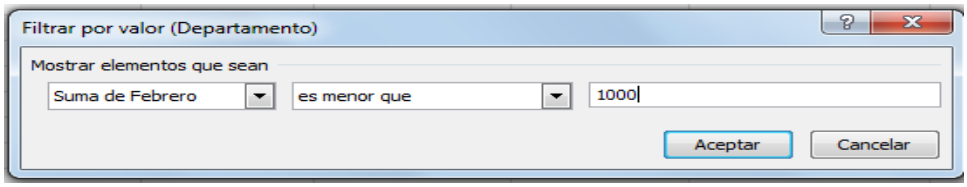
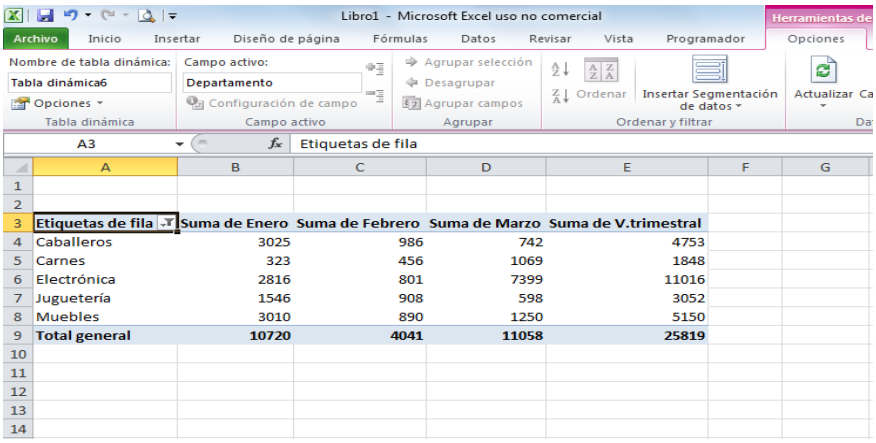


Figura 1.31. Filtrar por valor.

La figura 1.32 muestra cómo quedaría la nueva tabla dinámica.



Etiquetas de fila	Suma de Enero	Suma de Febrero	Suma de Marzo	Suma de V.trimestral
Caballeros	3025	986	742	4753
Carnes	323	456	1069	1848
Electrónica	2816	801	7399	11016
Juguetería	1546	908	598	3052
Muebles	3010	890	1250	5150
Total general	**10720**	**4041**	**11058**	**25819**

Figura 1.32. Tabla dinámica filtrada.

1.6 Macros.

Una macro es un conjunto de instrucciones que posteriormente podrá reproducir de forma automática. (GRUPO NORIEGA EDITORES, 2000)

Por medio de las macros se puede automatizar cualquier proceso de Microsoft Excel, acceso a todo tipo de opciones de los menús, introducción de datos, creación de gráficos, etc.

La forma más fácil de crear una macro es utilizando la grabadora de macros. El proceso consiste en ejecutar, una tras otra, las órdenes que se desean registrar en la macro, para que la grabadora de macros vaya generando las instrucciones macros equivalentes.

Desafortunadamente, la mayoría de las macros creadas mediante la grabadora de macros requieren modificaciones que será necesario realizar manualmente. También cuando necesite automatizar procesos complejos, puede ser que la grabadora de macros no sirva y sea necesario construir toda la macro tecleando las correspondientes instrucciones.

Las macros pueden ser una herramienta poderosa si están correctamente escritas, pero pueden ser desastrosas si no es así. Una macro mal diseñada puede destruir datos o borrar toda la hoja de cálculo sin dar tiempo de reaccionar. Es conveniente tener todo previsto, diseñando y probando las macros sobre una copia de las hojas de cálculo y no sobre las originales.

Como grabar una macro.

Se puede crear una macro utilizando el lenguaje de programación VBA, pero el método más sencillo es utilizar **la grabadora de macros** que guardará todos los pasos realizados para ejecutarlos posteriormente.

La **grabadora de macros** almacena cada acción que se realiza en Excel, por eso es conveniente planear con antelación los pasos a seguir de manera que no se realicen acciones innecesarias mientras se realiza la grabación. Para utilizar la grabadora de macros debes ir a la ficha **Vista** y seleccionar el ícono **Macros** y el comando **Grabar macro** (figura 1.33)**.**

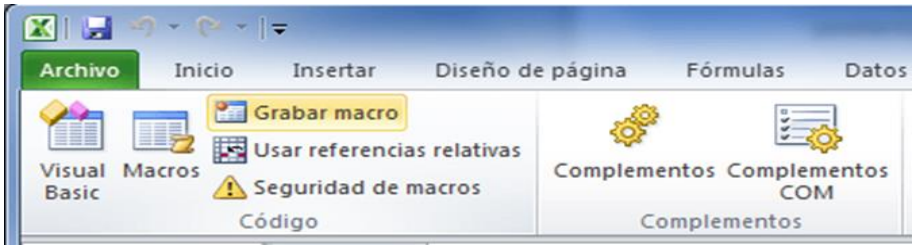


Figura 1.33. Ficha Vista.

Al pulsar el botón se mostrará el cuadro de diálogo **Grabar macro** como lo muestra la figura 1.34.

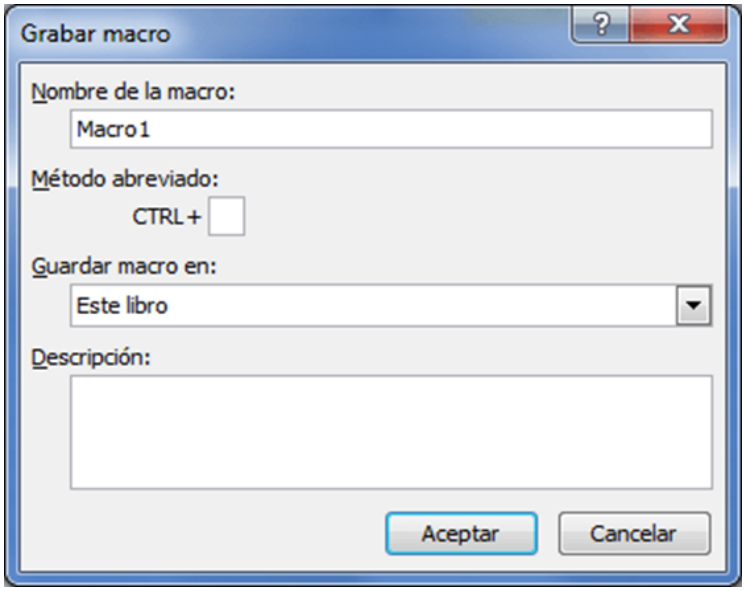


Figura 1.34. Cuadro de diálogo **Grabar macro**.

En el cuadro de texto **Nombre de la macro** deberá colocar el nombre que identificará de manera única a la macro que se creará. Opcionalmente podrá asignar un método abreviado de teclado mediante el cual podrá ejecutar la macro con la combinación de teclas especificadas.

La lista de opciones **Guardar macro en** permite seleccionar la ubicación donde se almacenará la macro.

- **Este libro.** Guardar la macro en el libro actual.
- **Libro nuevo.** La macro se guarda en un libro nuevo y que pueden ser ejecutadas en cualquier libro creado durante la sesión actual de Excel.
- **Libro de macros personal.** Esta opción permite utilizar la macro en cualquier momento sin importar el libro de Excel que se esté utilizando.

Además puede colocar una **Descripción** para la macro que va a crear. Finalmente pulse el botón Aceptar para iniciar con la grabación de la macro. Al terminar de ejecutar las acciones planeadas deberá pulsar el botón **Detener grabación** para completar la macro (ver figura 1.35).

Figura 1.35. Ícono para **Detener Grabación.**

Ejecutar macros.

Una vez que se generan las macros éstas pueden ser utilizadas, para ejecutarlas:

1.- Dar clic en el botón **Macros**, del grupo **Macros**, de la ficha **Vista**.
2.- Seleccionar la opción **Ver macros**.
3.- Dar clic en la macro deseada de la lista.
4.- Dar clic en ejecutar.

Otra forma sencilla de ejecutar una macro es utilizando la tecla Ctrl + letra (la letra con la que se grabó la macro).

Eliminar macros.

Para eliminar las macros que ya no son utilizadas se deben realizar los siguientes pasos:

1.- Dar clic en el botón **Macros**, del grupo **Macros**, de la ficha **Vista**.
2.- Seleccionar la opción **Ver macros**.
3.- Dar clic en la macro de la lista deseada.
4.- Dar clic en **Eliminar**, para que se lleve a cabo la acción.

En la figura 1.36 se puede observar el cuadro de diálogo para borrar una macro.

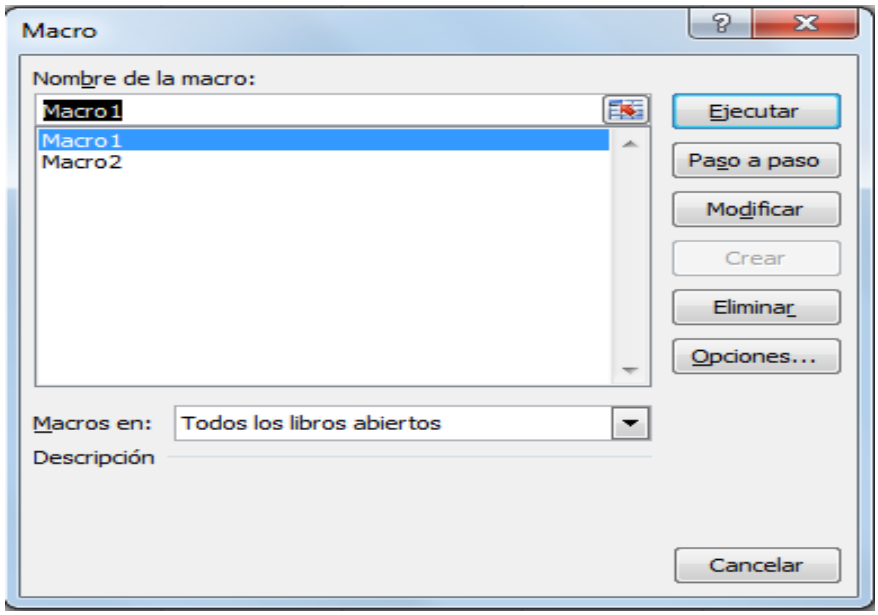


Figura 1.36. Cuadro de diálogo **Macro** para eliminar una macro.

1.7 Aplicaciones.

1.- Hacer la siguiente base de datos en Excel, utilizando formato de celdas para las calificaciones, Promedio y Promedio General con un entero y un decimal, ancho de la columna nombre ampliar a 20, centrar calificaciones y título de PROGRAMACIÓN estilo arial 14 y negrita, títulos de Nombre, Calificación1, Calificación2 y Promedio estilo arial 11, función PROMEDIO para el promedio general, función SUMA y fórmula para el Promedio.

PROGRAMACIÓN

Nombre	Calificación1	Calificación2	Promedio
Juan	8.0	8.9	8.5
María	7.3	8.0	7.7
Pedro	8.9	7.0	8.0
Luis	10.0	6.0	8.0
Ana	9.6	6.5	8.1
Bertha	8.5	9.0	8.8
José	9.9	10.0	10.0
Promedio general	8.9	7.9	8.4

2.- Hacer una macro en Excel, que al ejecutarla salga una base de datos como la del ejercicio 1.

3.- Con la siguiente base de datos, crear una tabla dinámica

Producto	Existencia	V. Semana1	V. Semana2	V. Semana3	V. Semana4	Total
Bolígrafo	100	5	9	16	0	70
Lápiz	100	6	10	20	1	63
Cuaderno	50	3	7	0	2	38
Goma	35	8	3	7	2	15
Caja de clip	20	0	5	6	0	9
Cartulina	30	10	2	4	6	8
Sacapuntas	20	2	8	3	1	6
Caja de colores	30	1	0	0	4	25
Plumones	25	1	4	2	7	11
Juego geométrico	15	1	3	2	5	4
Bicolor	26	1	7	3	4	11
Paquete de hojas bond	15	4	1	2	0	8
folder carta	46	6	0	8	6	26
folder oficio	53	2	0	5	4	42

4.- Capture la siguiente información a partir de la celda A1 de la Hoja 1 y realice los cambios indicados.

ALGORITMOS Y LENGUAJES DE PROGRAMACIÓN
Nombre del instructor:

		EJERCICIOS					
No	Nombre	E1	E2	PromE	Tareas	Evaluación Final	Calificación Final
1	Abreu	8.0	9.0		9.0	7.0	
2	Acuña	7.0	7.0		10.0	6.0	
3	Aguilar	6.7	9.0		9.0	9.0	
4	Alatriste	9.0	9.0		8.0	8.0	
5	Alcaraz	8.0	10.0		9.0	10.0	
6	Altamirano	7.0	10.0		10.0	7.5	
7	Becerril	9.0	9.0		10.0	9.0	

1. Utilizando una fórmula, obtenga el promedio de los EJERCICIOS para todos los usuarios. Coloque el resultado en la columna PromE.
2. Utilizando una fórmula, obtenga la Calificación Final. Debe tomar en cuenta el Promedio de ejercicios (PromE), las Tareas y Evaluación Final. Cada uno de éstos elementos representa el 20%, 30% y 50%. La fórmula quedaría así: =(E6*0.2)+ (F6*0.3)+ (G6*0.5).
3. Obtenga el Promedio de cada concepto, después de la última fila con información.
4. Coloque la celda activa en el resultado del promedio de las Tareas y active la herramienta **Rastrear precedentes**.
5. Coloque la celda activa en la primera calificación de la lista. Active la herramienta **Rastrear dependientes**.
6. Coloque la celda activa en la última Calificación Final, observe en la Barra de fórmulas lo mostrado, =(E12*0.2)+ (F12*0.3)+ (G12*0.5). Borre el número 12 del último paréntesis y acepte. Se mostrará un error. Observe en el signo de exclamación la ayuda sobre ese error.
7. Cambie la configuración de Excel, para mostrar los errores de las fórmulas en color Oro.
8. Corrija manualmente el error de la fórmula.
9. Guarde el archivo con el nombre “Calificaciones” con las opciones predeterminadas.
10. Cierre el archivo.

5.- En un archivo nuevo capture la información mostrada a partir de la celda A1 en la hoja 1 y realice los siguientes cambios:

Cambio de divisas

Par de divisas	Precio
USD a MXN	12.7237
EUR a MXN	16.8611
GBP a MXN	20.0931
BRL a MXN	7.3656
ARS a MXN	2.9277

1. Utilice la función adecuada para indicar cuántas divisas hay en esta lista.
2. Utilice la función adecuada para indicar cuántos precios hay en esta lista.
3. Utilice la función adecuada para indicar el promedio de precios.
4. Guarde el archivo con el nombre de “Contar” con todas las opciones predeterminadas.

6.- En un archivo nuevo capture la información mostrada y realice los siguientes cambios.

	A	B	C	D
1	Producto	Costo en dólares	Precio en pesos	Origen
2	iPod classic	300.00		Importado
3	iPod touch	250.00		Importado
4	iPad	555.00		Importado
5	iPhone	390.00		Importado
6	Tablet Android	199.00		Importado
7	Smartphone BB	200.00		Importado
8	Smartphone MT	550.00		Importado
9	Smartphone SS	120.00		Importado
10	Smartphone AL	90.00		Importado
11	Smartphone K	800.00		Nacional
12	Smartphone X	990.00		Nacional
13	Smartphone NK	320.00		Importado
14	Smartphone LG	390.00		Importado

1. En la columna Precio en pesos obtenga el costo de cada producto. Considerando como tipo de cambio el precio del dólar al día de hoy.
2. En la celda F2 capture la siguiente información:

Origen	
Importado	
Nacional	

3. Obtenga con la función CONTAR.SI, el total de productos nacionales y el total de productos importados.
4. En la F8, escriba el texto “teléfonos de más de 500 dólares”.

5. En la celda F9, obtenga con la función CONTAR.SI, la cantidad de teléfonos con un costo igual o mayor a 500 dólares.
6. Utilice la función adecuada para contar el número de productos y coloque el resultado en la celda F11.
7. Utilice la función adecuada para obtener un precio medio de todos los productos.
8. Guarde el archivo con el nombre CONTAR_SI con todas las opciones predeterminadas.

7.- Capture la siguiente información a partir de la celda A1 en un libro nuevo y realice los cambios indicados.

FACTURA	CLIENTES	ZONA	VALOR FACTURA
5896	Amparo	Satélite	$7640
5896	Arturo	Centro	$8200
4789	Blanca	Coapa	$1450
9632	Catalina	Coapa	$8340
4523	César	Centro	$2480
1117	Claudia	Coyoacán	$9870
1120	Celia	Coyoacán	$8620
4523	Daniela	Centro	$8420
4523	Enrique	Coyoacán	$3140
1125	Silvia	Satélite	$2460
1130	Teresa	Satélite	$5210
1125	Amelia	Polanco	$6780
1120	Vicente	Coyoacán	$4120
1125	María	Polanco	$7410
4789	Yolanda	Coapa	$2560

1. En la celda D17 inserte una función que sume el VALOR FACTURA de la ZONA Centro. Utilice la función SUMAR.SI, desde la ficha **Fórmulas**.
2. Ahora en la celda D18 sume el VALOR FACTURA únicamente de la ZONA Satélite, escriba en la celda C18 el texto “Total Satélite”. Utilice la función SUMAR.SI, desde la ficha **Fórmulas**.
3. En la columna E, escriba de título “Impuestos”.
4. En esta última columna creada inserte una fórmula para cada renglón de la columna D que multiplique su valor por 0.16 (para obtener el impuesto correspondiente). Coloque la celda activa en donde requiere el resultado e inserte la fórmula.
5. Al final de la columna E obtenga el PROMEDIO de los impuestos.
6. Guarde el archivo con el nombre SUMAR_SI, con las opciones predeterminadas y cierre el archivo.

8.- Capture la siguiente información en un nuevo archivo en la hoja 1 a partir de A1 y realice los cambios solicitados.

Nombre	Destino	Precio VTP
BARRANCAS DEL COBRE	Ciudad	
CANCÚN	Playa	
CHICHÉN ITZA	Sitio Arqueológico	
GUADALAJARA	Ciudad	
ISLA HOLBOX	Playa	
LOS CABOS	Playa	
MANZANILLO	Playa	
MAZATLÁN	Playa	
MONTERREY	Ciudad	
PALENQUE	Sitio Arqueológico	
PLAYA DEL CARMEN	Playa	
PUERTO ESCONDIDO	Playa	
RIVIERA MAYA	Sitio Arqueológico	
SAN CRISTOBAL DE LAS CASAS	Ciudad	
TULÚM	Sitio Arqueológico	
ZACATECAS	Ciudad	

A partir de la celda E1, capture la siguiente información.

Destino	Precio VTP
Ciudad	$ 5,200.00
Playa	$ 7,800.00
Sitio Arqueológico	$ 6,200.00

1. Aplique un borde a todos los datos capturados, utilice la opción: “Todos los bordes”.
2. Copie los datos de la columna Nombre y péguelos en una Hoja 2 en la celda A1. Para pegarlos utilice la herramienta Trasponer de la lista desplegable del botón **Pegar**. Realice los mismos pasos para la columna Destino y Costo. Los datos comenzarán en A2 y A3 respectivamente.
3. En relación a la matriz ubicada en la celda E1, copie los datos de la columna Destino y péguelos en una Hoja 2 en la celda A6.
4. Cambie el nombre de las hojas de la siguiente forma, Hoja 1 por Datos origen. Hoja 2 por Costos.
5. Agregue un Comentario en la celda A7 con el texto “Precio VTP” con el texto, “Incluye transportación, hospedaje, alimentos y propinas. Aplican restricciones”.
6. Obtenga en la celda A9 el total de destinos, utilice para ello la función adecuada.
7. Obtenga en las celdas A10, A11 y A12 el total de destinos de Ciudad, Playa y Sitio Arqueológico, utilice para ello la función adecuada.

8. En la hoja Costos obtenga los precios de cada destino, para cada celda de Precio VTP de acuerdo la matriz de la misma hoja. Utilice para ello la función BUSCARH.
9. Obtenga en la celda A14, el promedio de los Precio VTP.
10. Guarde el archivo con el nombre “BUSCARH” con todas las opciones predeterminadas.

9.- Capture la siguiente información en un nuevo archivo en la hoja 1 a partir de A1 y realice los cambios solicitados.

Alumno	Calificación	Resultado
Jaime	10	
Rosa	2	
Manuel	6	
Gustavo	8	
Marcelo	5	
Hugo	9	
Janet	10	
Gerardo	7	

A partir de la celda E1, capture la siguiente información.

Calificación	Resultado
1	No acreditado
2	No acreditado
3	No acreditado
4	No acreditado
5	No acreditado
6	Suficiente
7	Suficiente
8	Acreditado
9	Acreditado
10	Excelente

1. Aplique un borde a todos los datos capturados, utilice la opción: **Todos los bordes.**
2. Obtenga el Resultado de cada alumno, en relación a su Calificación de acuerdo a la matriz de la misma hoja. Utilice para ello la función BUSCARV.
3. Guarde el archivo con el nombre BUSCARV con todas las opciones predeterminadas.

10.- Capture los datos mostrados en un libro nuevo y realice los cambios indicados

Compra de material

Material	Precio	Estatus
Pintura	30	
Impermeabilizante	40	
Barniz	55	
Solvente	20	

1. Utilice la función SI para agregar el texto “Dentro del presupuesto”, para las cantidades menores o iguales a 30 y para las mayores agregar el texto “Presupuesto excedido”.
2. Guarde el archivo con el nombre “FUNCION_SI” con todas las opciones predeterminadas.

2. DESARROLLO DE LÓGICA ALGORÍTMICA.

2.1 Metodología para la solución de problemas.

Los humanos realizamos cotidianamente series de pasos, procedimientos o acciones que nos permiten alcanzar algún resultado o resolver algún problema. En realidad todo el tiempo estamos aplicando algoritmos para resolver problemas.

Un algoritmo es un método para resolver un problema mediante una serie de pasos precisos, definidos y finitos. (JOYANES AGUILAR, 2005)

Un algoritmo debe ser *preciso*, esto quiere decir que indica el orden de realización en cada paso, debe ser *definido* esto es, que si se sigue dos veces, se obtiene el mismo resultado cada vez y debe ser *finito* (tiene un fin) que significa un determinado número de pasos.

Muchas veces aplicamos algoritmos sin darnos cuenta, inconscientemente o automáticamente. Esto sucede cuando el problema al que nos enfrentamos ya lo hemos resuelto con anterioridad un gran número de veces.

Por ejemplo, si tenemos que abrir una puerta. Lo hemos hecho tantas veces que no enumeramos los pasos para alcanzar este objetivo. Igual ocurre cuando nos subimos a un auto o cuando nos cepillamos los dientes, lo hacemos de manera automática.

Ejemplo 1: Algoritmo para cambiar el neumático desinflado de un automóvil.

1.- Inicio.
2.- Bajar la herramienta y el neumático de repuesto del automóvil.
3.- Aflojar los birlos del neumático desinflado.
4.- Acomodar el gato.
5.- Levantar el automóvil.
6.- Quitar los birlos del neumático.
7.- Quitar el neumático desinflado.
8.- Colocar el neumático de repuesto.
9.- Fijar los birlos del neumático de repuesto.
10.- Bajar el automóvil.
11.- Apretar los birlos del neumático de repuesto.
12.- Quitar el gato.
13.- Guardar el neumático desinflado y la herramienta.
14.- Fin.

Ejemplo 2: Algoritmo para calcular el área de un triángulo.

1.- Inicio.
2.- Solicitar (leer) los datos (base y altura).
3.- Multiplicar la base por la altura y el resultado dividirlo entre dos, y guardarlo en una variable.
4.- Mostrar (imprimir) el resultado almacenado en la variable.
5.- Fin.

El proceso para la resolución de un problema es el siguiente: Análisis del problema, Construcción del algoritmo y verificación del algoritmo.

- **Análisis del problema.** Requiere una clara definición, donde se contemple exactamente lo que debe hacer el algoritmo y el resultado o solución que se desea.
  Para identificar y definir bien un problema es conveniente responder a las siguientes preguntas:
  ¿Qué entradas se requieren?
  ¿Cuál es la salida deseada?
  ¿Qué método produce la salida deseada?

- **Construcción del algoritmo.** En esta etapa se determina *cómo* hace el algoritmo la tarea solicitada, que métodos se van utilizar (top-down, bottom up, modular).

- **Verificación del algoritmo.** En esta etapa se le hacen pruebas al algoritmo con una amplia variedad de datos de entrada, con la finalidad de verificar si se obtiene el resultado deseado o tiene errores para que éstos sean corregidos.

Por otra parte, un algoritmo consta de tres secciones o módulos principales: Datos de entrada, procesamiento de los datos e impresión de resultados.

2.2 Metodología para el diseño de software:

Top down, bottom up, modular y programación estructurada.

- **Diseño descendente (top_down).** Las instrucciones están ordenadas de arriba abajo, lo que facilita el diseño del algoritmo, el entendimiento del código y por lo tanto el mantenimiento del mismo.

- **Diseño Ascendente (bottom up).** En este diseño las partes individuales se diseñan con detalle y luego se enlazan para formar componentes más grandes, que a su vez se enlazan hasta que se forma el sistema completo.

- **Programación modular.** La programación estructurada divide un problema en secciones o partes (módulos), esto permite resolverlo de manera sencilla y en forma paralela si es necesario, es decir por un equipo de personas.

  La descomposición de un programa en módulos independientes más simples se conoce también como el método de divide y vencerás. Se diseña cada módulo con independencia de los demás, siguiendo un método ascendente o descendente se llegará hasta la descomposición final del problema en módulos en forma jerárquica.

- **Programación estructurada.** Significa escribir un programa de acuerdo a las siguientes reglas:
  - El programa tiene un diseño modular.
  - Los módulos son diseñados de modo descendente.
  - Cada módulo se codifica utilizando las tres estructuras de control básicas: secuencia, selección y repetición.

- La programación estructurada significa también programación sin GOTO (C no requiere del uso de esta sentencia). Esta programación utiliza un número limitado de estructuras de control que hace que se reduzca la complejidad de los programas y, por consiguiente, reducen los errores; hace los programas más fáciles de escribir, verificar, leer y mantener.

  - **Estructuras de control básicas.** Estas estructuras controlan el orden de las instrucciones. En la programación estructurada existen tres tipos de estructuras de control: la secuencial, selectiva y la repetitiva.

    - **Estructura de control secuencial.** Las instrucciones se ejecutan en orden, una por una desde la primera hasta la última, es decir el programa ejecuta todas las instrucciones del programa en el orden establecido sin saltarse ninguna de ellas.

    - **Estructura de control selectiva.** Mediante una condición que puede ser verdadera o falsa se elige una opción, la cual realiza una acción (o varias acciones). La condición puede ser simple o compuesta.

    - **Estructura de control repetitiva.** Una acción se repite una cantidad definida o indefinida de veces mientras una condición sea verdadera.

Elementos y reglas de la representación gráfica y manuscrita de los algoritmos (diagrama de flujo, diagrama N-S, diagrama estructurado y pseudocódigo).

Para representar un algoritmo se debe utilizar algún método, esto permitirá que un algoritmo pueda ser codificado indistintamente en cualquier lenguaje. Para lograr este objetivo se necesita que el algoritmo sea representado gráfica o numéricamente. Los métodos usuales son:

Diagrama de flujo.
Diagrama N-S (Nassi-Schneiderman).
Pseudocódigo.

Pseudocódigo. Nació como un lenguaje similar al inglés y era un medio de representar básicamente las estructuras de control de programación estructurada. Se considera un primer borrador, dado que el pseudocódigo tiene que traducirse posteriormente a un lenguaje de programación. La ventaja del pseudocódigo es que en su uso, en la planificación de un programa, el programador se puede concentrar en la lógica y en las estructuras de control y no preocuparse de las reglas de un lenguaje específico. También es fácil de modificar si se descubren errores en la lógica del programa. Otra ventaja es que puede ser traducido fácilmente a lenguajes estructurados como Pascal, C, FORTRAN 77/)), C++, Java,C#, etc.

El pseudocódigo original utiliza para representar las acciones sucesivas palabras reservadas en inglés (similares a sus homónimas en los lenguajes de programación), tales como *start, end, stop, if-then-else, while-end, repeat-until,* etc. La escritura exige normalmente la indentación (sangría en el margen izquierdo) de diferentes líneas.

Ejemplo:

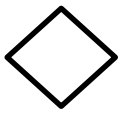
```
start
        //cálculo de impuestos y salarios
        read nombre, horas, precio_hora
        salario_bruto = horas * precio_hora
        tasas = 0.25 * salario_bruto
        salario_neto = salario_bruto – tasas
        write nombre, salario_bruto, tasas, salario_neto
end (JOYANES AGUILAR, 2005)
```

El pseudocódigo comienza con la palabra start y finaliza con la palabra end, en inglés (en español, inicio y fin).

La línea precedida por // se denomina comentario. Es información para el lector del programa y no realiza ninguna instrucción ejecutable, sólo tiene efecto de documentación interna del programa.

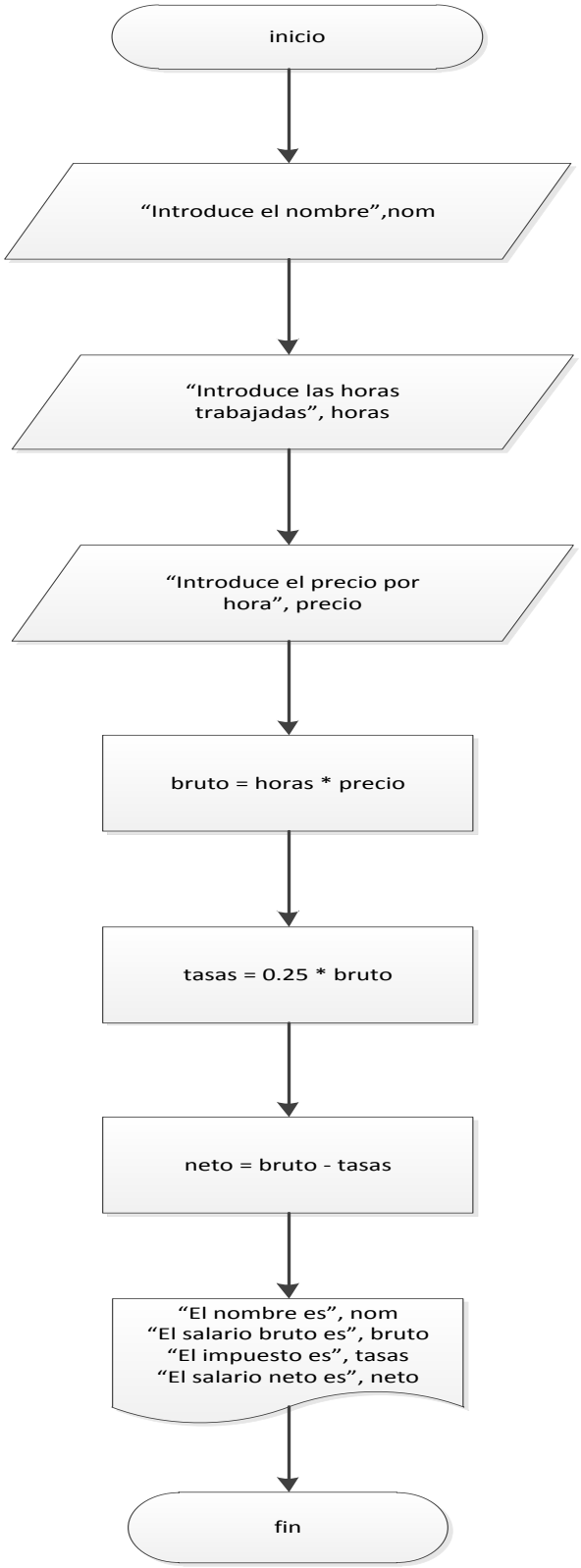
Diagrama de flujo. Es una de las técnicas de representación gráfica de algoritmos más antigua y a la vez más utilizada. Dicha representación gráfica se lleva acabo cuando varios símbolos (que indican diferentes procesos en la computadora) se relacionan entre sí mediante líneas que indican el orden en que se deben ejecutar las instrucciones para obtener los resultados deseados. En la tabla 2.1 podemos observar los símbolos que se utilizan y que han sido reglamentados por el Instituto Nacional de Normalización Estadounidense (ANSI).

Tabla 2.1. Símbolos de diagrama de flujo.

Símbolos	**Función**
	Terminal.- Representa el comienzo, y el final de un programa.
	Entrada/Salida.- Cualquier tipo de introducción de datos en la memoria desde los periféricos (entrada), o registro de la información procesada en un periférico (salida).
	Proceso.- Cualquier tipo de operación que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones aritméticas, de transferencia, etc.
	Decisión.- Indica la realización de una comparación de valores.
	Conector. Sirve para enlazar dos partes cualesquiera del diagrama dentro de la misma página.
	Línea de flujo.- Indica el sentido de ejecución de las operaciones.
	Conector.- Conexión entre dos puntos del diagrama situado en páginas diferentes.
	Impresión. Se utiliza en ocasiones en lugar del símbolo de E/S.

Ejemplo de diagrama de flujo

←



Diagramas de Nassi-Schneiderman (N-S). También conocido como diagrama de Chapin, es como un diagrama de flujo en el que se omiten las flechas de unión y las cajas son contiguas. Las acciones sucesivas se escriben en cajas sucesivas y, como en los diagramas de flujo, se pueden escribir diferentes acciones en una caja. (JOYANES AGUILAR, 2005)

Ejemplo

leer nombre, horas, precio
calcular salario = horas * precio
calcular impuestos = 0.25 * salario
calcular neto = salario - impuestos
imprimir nombre, salario, impuestos, neto

En la figura 2.1 se puede observar la representación de la estructura condicional.

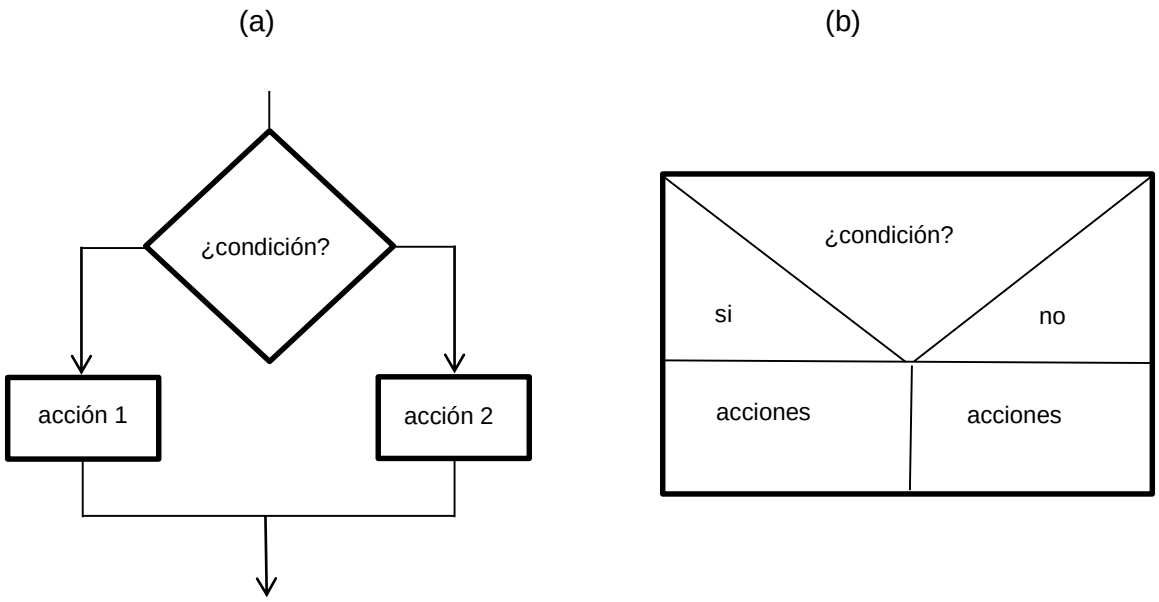
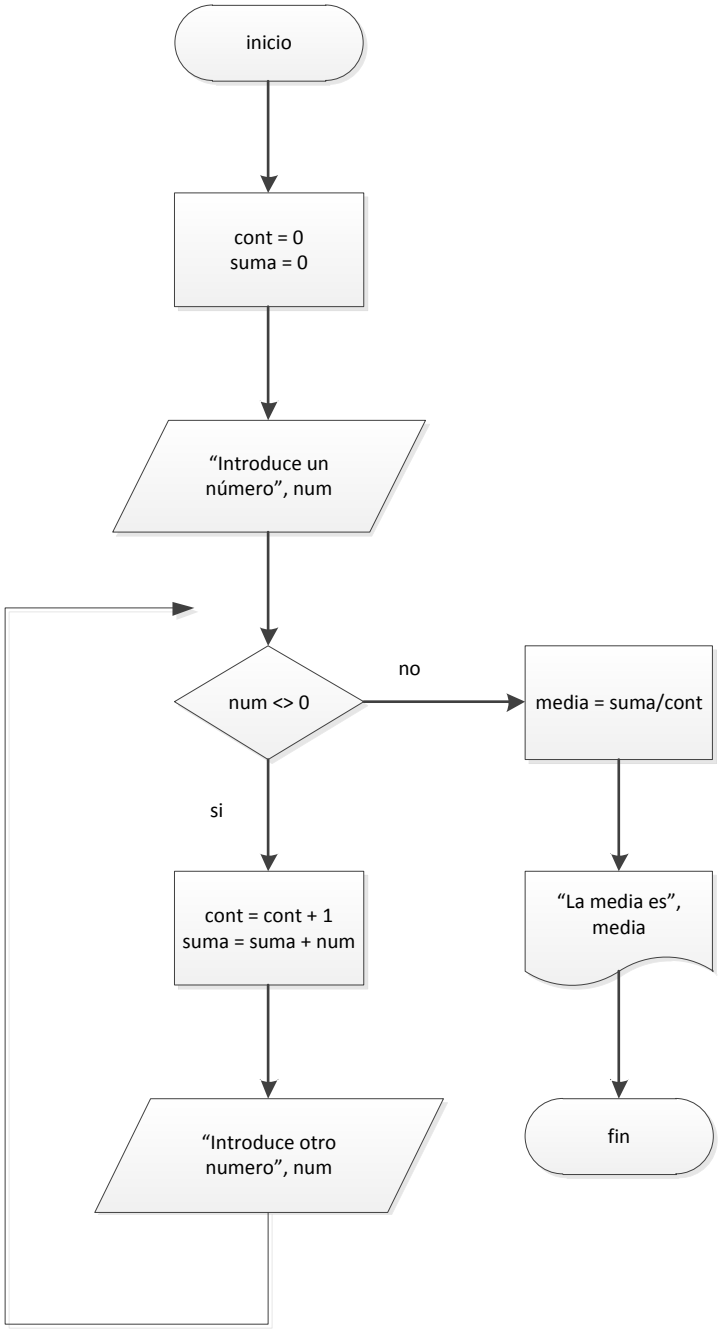


Figura 2.1. Estructura condicional o selectiva: (a) diagrama de flujo: (b) diagrama N-S.

A continuación se presenta un ejemplo de diagrama de flujo estructurado y su pseudocódigo correspondiente.
Calcular la media de una serie de números positivos, suponiendo que los datos se leen desde una terminal. Un valor de cero (como entrada) indicará que se ha alcanzado el final de la serie de números positivos.

Diagrama de flujo



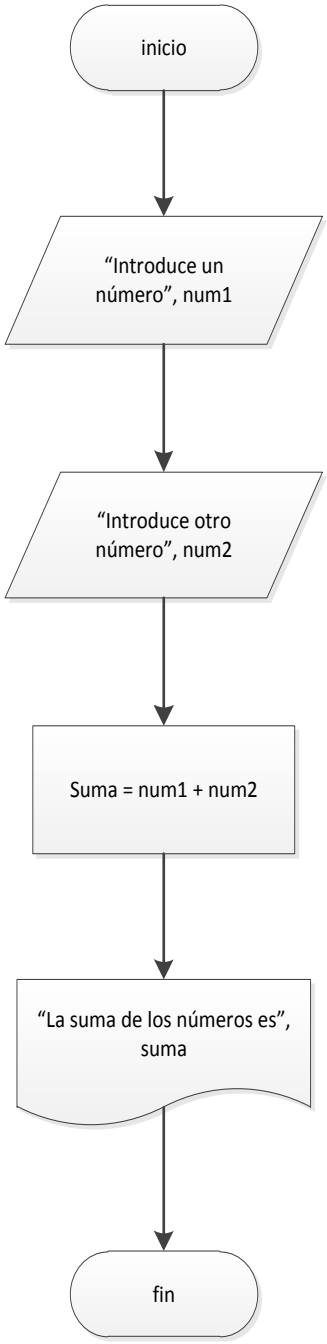
Pseudocódigo

```
inicio
        cont = 0
        sum = 0
        imprimir “Introduce un número”
        leer num
        mientras (num <> 0)
        inicio
                cont = cont + 1
                suma = suma + num
                imprimir “Introduce otro número”
                leer num
        fin
        media = suma / cont
        imprimir “La media es: “, media
fin
```

2.3 Implementación de algoritmos secuenciales.

1.- Realizar el pseudocódigo y diagrama de flujo, que lea dos números enteros, los sume e imprima el resultado de la suma.

Diagrama de flujo



Pseudocódigo

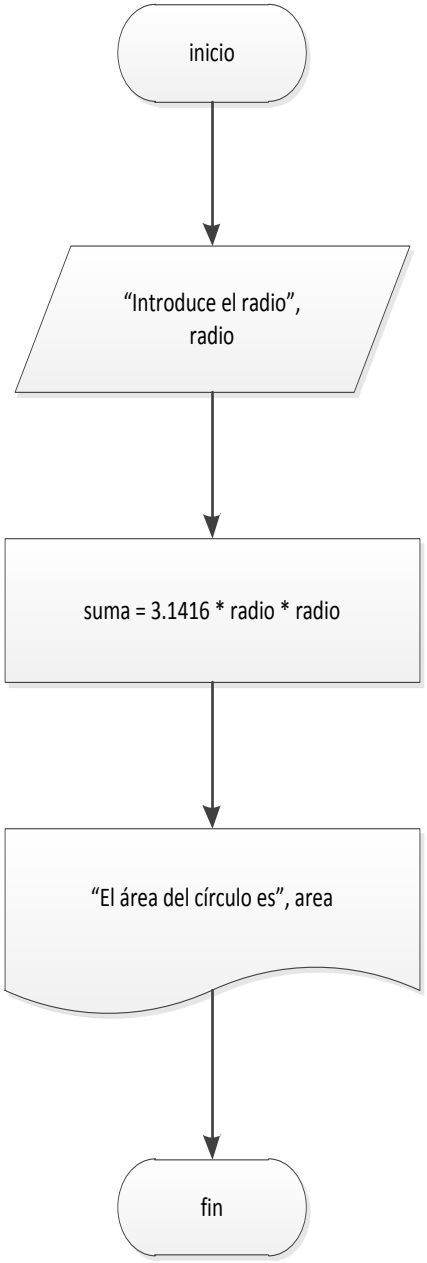
inicio

imprimir "Introduce un número"
leer num1
imprimir "Introduce otro número"
leer num2
suma = num1 + num2
imprimir "La suma de los números es", suma

fin

2.- Realizar el pseudocódigo y diagrama de flujo, que lea el valor del radio de un círculo, calcule el área e imprima ésta.

Diagrama de flujo



Pseudocódigo

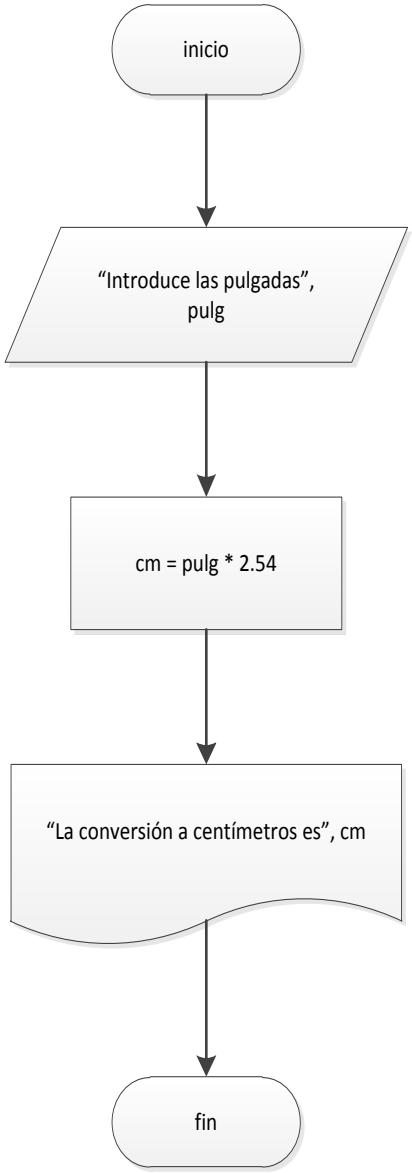
inicio

imprimir "Introduce el radio"
leer radio
area = 3.1416 * radio * radio
imprimir "El área del círculo es ",area

fin

3.- Realizar el pseudocódigo y diagrama de flujo, que lea una cantidad en pulgadas y la convierta en centímetros e imprima el resultado.

Diagrama de flujo



Pseudocódigo

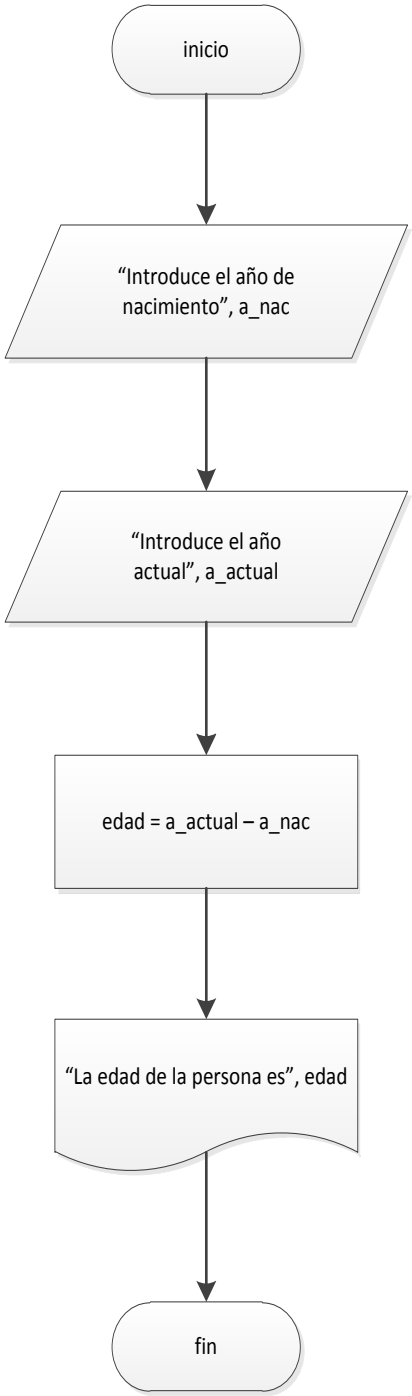
inicio

imprimir “Introduce las pulgadas”
leer pulg
cm = pulg * 2.54
imprimir “La conversión a centímetros es” ,cm

fin

4.- Realizar el pseudocódigo y diagrama de flujo, que introduzca el año de nacimiento de una persona y calcule e imprima la edad de ésta.

Diagrama de flujo



Pseudocódigo

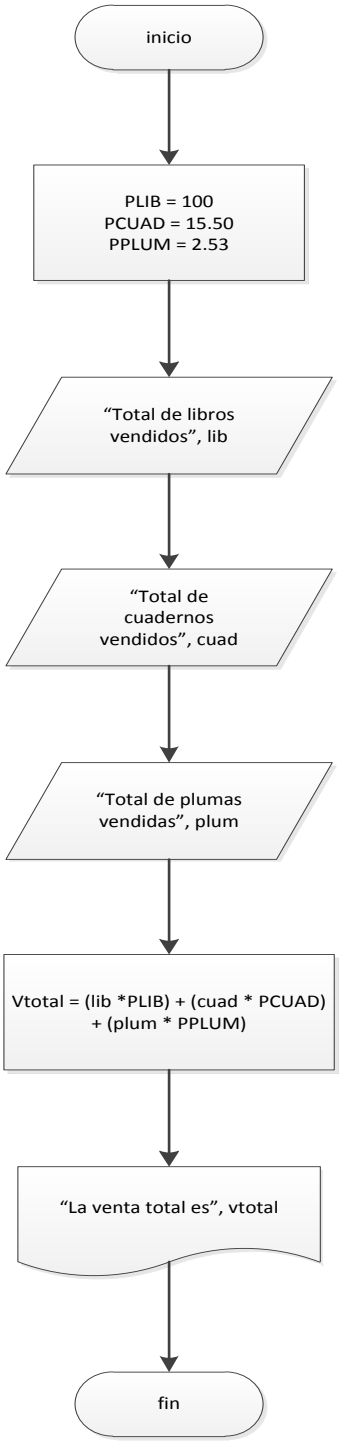
inicio

imprimir "Introduce el año de nacimiento"
leer a_nac
imprimir "Introduce el año actual"
leer a_actual
edad = a_actual – a_nac
imprimir "La edad de la persona es" ,edad

fin

5.- Realizar el pseudocódigo y diagrama de flujo, que realice lo siguiente: Una tienda vende libros a $100.00, cuadernos a $15.50 y plumas a $ 2.35. Calcule e imprima el monto total de la venta según el número de artículos vendidos.

Diagrama de flujo



Pseudocódigo

```
inicio
        PLIB = 100
        PCUAD = 15.50
        PPLUM = 2.53
        imprimir “Total de libros vendidos”
        leer lib
        imprimir “Total de cuadernos vendidos”
        leer cuad
        imprimir “Total de plumas vendidas”
        leer plum
        vtotal = lib * PLIB + cuad * PCUAD + plum * PPLUM;
        imprimir “La venta total fue de:”,vtotal
fin
```

2.4 Pruebas y depuración.

Se denomina prueba de escritorio a la comprobación que se hace de un algoritmo para saber si está bien hecho. Esta prueba consiste en tomar datos específicos como entrada y seguir la secuencia indicada en el algoritmo hasta obtener un resultado, el análisis de estos resultados indicará si el algoritmo está correcto o si por el contrario hay necesidad de corregirlo o hacerle ajustes. Es Importante realizar una prueba de escritorio para eliminar variables no necesarias, crear variables faltantes, ocupar los ciclos adecuados, y principalmente llegar al objetivo del problema.

Básicamente, una prueba de escritorio es una ejecución "a mano" del algoritmo, por lo tanto se debe llevar registro de los valores que va tomando cada una de las variables involucradas en el mismo.

A continuación se toman los tres primeros ejercicios (pseudocódigos) del tema anterior (2.3) para aplicarle una prueba de escritorio.

Pseudocódigo 1

inicio

    imprimir "Introduce dos números"
    leer num1, num2
    suma = num1 + num2
    imprimir "La suma de los dos números es", suma

fin

Prueba de escritorio

Ejecución	Num1	Num2	suma	imprime
1	4	8	12	La suma de los dos números es 12
2	-7	-6	-13	La suma de los dos números es -13
3	25	9	34	La suma de los dos números es 34

Pseudocódigo 2

Inicio

    imprimir "Introduce el radio"
    leer radio
    area = 3.1416 * radio * radio
    imprimir "El área del círculo es ",area

fin

Prueba de escritorio

Ejecución	radio	area	imprime
1	5	78.54	El área del círculo es 78.54
2	4.5	63.62	El área del círculo es 63.62

Pseudocódigo 3

inicio

imprimir “Introduce la cantidad en pulgadas”

leer pulg

cm = pulg * 2.54

imprimir “La conversión a centímetros es” ,cm

fin

Prueba de escritorio

Ejecución	pulg	cm	imprime
1	6	15.24	La conversión a centímetros es 15.24
2	10	25.4	La conversión a centímetros es 25.4

Ejercicios para practicar

Elaborar pseudocódigo, diagrama de flujo y prueba de escritorio de los siguientes ejercicios:

1.- Leer el radio de una esfera y calcular e imprimir el volumen.
2.- Calcular e imprimir el área y perímetro de un rectángulo.
3.- Leer el lado de un triángulo equilátero. Calcular e imprimir el perímetro, la altura y el área de dicho triángulo.
4.- Calcular e imprimir el importe total a pagar de la compra de tres productos considerando que hay un descuento de 10% al total de la compra.
5.- Leer una cantidad de pesos (número entero) y calcular e imprimir a cuántos billetes de 1000, 500, 200, 100, 50, 20 y monedas de 10, 5, 1 peso equivale.
6.- Leer el precio por hora y horas trabajadas de un empleado. Calcular e imprimir el sueldo semanal.
7.-Leer la clave de un empleado y los seis primeros sueldos del año, calcule el ingreso total semestral y el promedio mensual, e imprima la clave del empleado, el ingreso total y el promedio mensual.
8.- Leer el radio de un círculo, calcule e imprima tanto su área como la longitud de su circunferencia.
9.- Suma de los número pares comprendidos entre 2 y 100
10.- Dados tres números, determinar si la suma de cualquier pareja de ellos es igual al tercer número. Si se cumple esta condición, imprimir “iguales”, en caso contrario imprimir “distintas”.
11.- Leer los tres lados de un triángulo e imprimir “escaleno”, ”equilátero” o “isósceles”.
12.- Leer tres calificaciones de un alumno. Calcular el promedio si éste es mayor de 69 imprimir “acreditado” de lo contrario imprimir “no acreditado”.

3. INTRODUCCIÓN A LA PROGRAMACIÓN DE UN LENGUAJE ESTRUCTURADO.

3.1 Introducción a la programación.

La programación estructurada es un conjunto de técnicas que nos permite desarrollar programas fáciles de escribir, verificar, leer y mantener e incluye: **Diseño descendente (top_down), Estructuras de datos.** Es un conjunto de datos dónde se almacena y se puede acceder a elemento individuales de datos, **Programación modular y Estructuras de control.**

3.2 Estructura básica de un programa.

En la programación estructurada se recomienda tener un orden en los programas. La estructura de un programa en lenguaje C es la siguiente:

```
Declaración de archivos de cabecera (librerías)
Prototipo de las funciones
Declaración de variables globales y constantes
main()  // Programa principal
{
        Declaración de constantes (locales)
        Declaración de variables (locales)
        Cuerpo del programa (estructuras de control)
}
Declaración de funciones
{
}
```

Declaración archivos de cabecera o encabezado (librerías o biblioteca). Indican al compilador que se incluyan las líneas de sentencias que están dentro del archivo que se declara; son archivos proporcionados por el fabricante del compilador, y suelen declarar funciones, variables y constantes que van a ser utilizadas por las sentencias que el programador va a manejar en las siguientes líneas del programa.

Para llamar a un archivo de cabecera es necesario hacer uso de la directiva #include, que tiene la siguiente sintaxis:

```
#include nombre_archivo_cabecera
```

Donde nombre_archivo_cabecera es el nombre de un archivo que contiene las declaraciones y definiciones para un propósito específico. Este archivo debe tener la extensión .h.

Ejemplos:

#include <stdio.h>	contiene declaraciones de rutinas de entrada/salida.
#include <math.h>	contiene declaraciones de funciones matemáticas.
#include <conio.h>	consola y puertos de E/S.

Prototipo de funciones. Es la declaración de una función que sólo contiene la cabecera; a diferencia de la declaración completa, al final lleva punto y coma. El prototipo le informa al compilador que existe una función que va a regresar un determinado tipo de dato y qué parámetros utilizará.

Declaración de variables globales y constantes. Tienen el objetivo de identificar los datos que se utilizan en las operaciones y cálculos que realiza el programa y que se almacenan en la memoria de la computadora. Como son variables globales esto quiere decir que pueden ser utilizadas en todo el programa.

Programa principal o función principal main(). Contiene el flujo del programa llamando a las funciones necesarias para su funcionamiento. La función main() indica dónde empieza el programa, cuyo cuerpo principal es el conjunto de instrucciones delimitadas por dos llaves (una que abre {, y otra que cierra }).

Declaración de funciones. Todas las funciones que no están definidas en las librerías tienen que ser declaradas. Las funciones son un conjunto de instrucciones que realizan una tarea específica y que retornan un dato como resultado de las operaciones realizadas.

Es importante tener claro que lo mínimo que debe contener un programa en lenguaje C es un archivo de cabecera y el main(). Por ejemplo:

```
#include <stdio.h>
main()
{
        printf("Bienvenidos");
}
```

3.3 Tipos de datos.

Los diferentes objetos de información con los que un algoritmo o programa trabaja se conocen colectivamente como datos. Todos los datos tienen un tipo asociado con ellos; el tipo de un dato es el conjunto (rango) de valores que puede tomar durante el programa. (CORONA NAKAMURA & ANCONA VALDEZ, 2011)

Los datos que procesa una computadora se clasifican en simples y estructurados.
Los tipos de datos simples se caracterizan por ocupar una casilla de memoria. Ejemplos de datos simples son: los enteros, los reales y los caracteres. En la tabla 3.1 podemos observar los tipos de datos simples en lenguaje C, así como su rango.

Tabla 3.1 Tipos de datos simples.

Tipos de datos	**Descripción**	**Rango**
int	Entero	-32,768 a 32,767
float	Real	3.4×10^{-38} a 3.4×10^{38}
long	Entero de largo alcance	-2,147,483,648 a 2,147,483,647
double	Real de doble precisión	1.7×10^{-308} a 1.7×10^{308}
char	carácter	Abecedario, números o símbolos especiales

Los tipos de datos estructurados se caracterizan por el hecho de que con un nombre se hace referencia a un grupo de casillas de memoria, es decir, un dato estructurado tiene varios componentes, ejemplos los arreglos, las cadenas de caracteres y registros.

3.4 Identificadores.

Los datos que procesará la computadora, ya sean simples o estructurados, se deben almacenar en casillas o celdas de memoria para utilizarlos posteriormente. A estas casillas o celdas de memoria se les asigna un nombre para reconocerlas, a esto se le llama **identificador**.

El nombre de un identificador se forma por medio de letras, dígitos y el carácter de subrayado (_). Siempre se debe iniciar con una letra. El lenguaje de programación C distingue entre minúsculas y mayúsculas, por lo tanto **SUMA** y **suma** son identificadores diferentes. La longitud más común de un identificador es de tres caracteres, y generalmente no excede los siete caracteres. En C, dependiendo del compilador que se utilice, es posible generar nombres de identificadores más grandes.

Existen nombres que no pueden ser utilizados por ser palabras reservadas del lenguaje C, los cuales se presentan a continuación en la tabla 3.2:

Tabla 3.2. Palabras reservadas del lenguaje C.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatil
do	if	static	while

3.5 Almacenamiento, direccionamiento y representación en memoria.

Variables. Es un espacio en la memoria que el programador reserva con el fin de almacenar esos datos "desconocidos" cuando empieza la ejecución de un programa o que pueden ir cambiando durante ese proceso.

Para poder reservar tantos espacios como se requieran es necesario declarar las variables. La sintaxis para la declaración de variables en lenguaje C es la siguiente:

tipo_de_dato identificador;

En la tabla 3.3 se muestran algunos ejemplos.

Tabla 3.3. Ejemplos de declaración de variables.

Ejemplos	**Acción**
int a;	Se reserva un espacio en la memoria llamado "a", con capacidad para un entero.
float b, c, d;	Se reservan 3 espacios en la memoria para guardar 3 números reales, a los cuales se hace referencia mediante "b", "c" y "d" respectivamente.
char j;	Se reserva un espacio de memoria para poder almacenar cualquier carácter y se pueda hacer referencia a este espacio mediante el identificador "j".

La memoria es un conjunto de celdas direccionables; es decir se puede tener acceso a cada una mediante su dirección y en ellas puede almacenar datos. En la figura 3.1 muestra de manera descriptiva como quedaría la memoria luego de haber declarado gráficamente las variables de los ejemplos anteriores.

0	1	2	3 a	4 a	5	6	7	8	9 j
10 b	11 b	12 b	13 b	14	15 c	16 c	17 c	18 c	19
20	21	22 d	23 d	24 d	25 d	26	27	28	29…

Figura 3.1. Representación gráfica de la memoria.

Los espacios se asignaron arbitrariamente, dado que en la realidad las variables no necesariamente quedarán de manera consecutiva en la memoria; lo que importa es que exista espacio suficiente para las variables a declarar.

En conclusión una variable es un espacio en la memoria cuyo contenido puede cambiar durante la ejecución de un programa.

Constantes. Es un espacio en memoria que recibe un valor por primera vez y generalmente no se modifica durante la ejecución de un programa.

Una constante se utiliza cuando se conoce de antemano el valor de algún dato, pero además se sabe que este dato no debe cambiar.

En lenguaje C, las constantes se crean utilizando la directiva del preprocesador *#define* (constantes simbólicas definidas) o bien la palabra reservada *const* (constantes simbólicas declaradas).

La sintaxis para declarar una constante simbólica definida es la siguiente:

```
#define   identificador   valor
```

Ejemplos:

```
#define   SALUDO   “Hola a todos”
#define   PI   3.1416
#define   NP   1506
#define   CAR   “a”;
```

El preprocesador de C sustituye los valores “Hola a todos”, 3.1416, 1506 y “a” cada vez que se encuentra en un programa el nombre de las constantes simbólicas SALUDO, PI, NP y CAR respectivamente.

Observe que las constantes creadas con #define, no es necesario especificarles el tipo de dato que deberán tener; estas constantes adquieren su tipo dependiendo de la naturaleza del valor asignado.

Cuando se utiliza #define para crear una constante, en realidad los valores no ocupan un espacio en la memoria como en el caso de las variables, sino que el compilador sustituye cada ocurrencia del nombre simbólico por su respectivo valor.

Cuando declaramos una constante con la palabra reservada const, el dato si ocupa un espacio en memoria como si fuera una variable, excepto por que su contenido no cambia. Su sintaxis es la siguiente:

```
const  tipo_de_dato   identificador   =   valor;
```

Ejemplos:

```
const   int   NP  =  1506;
const   float   PI  =  3.1416;
const   char   CAR  =  'a';
const   char   SALUDO[ ]  =  "hola a todos";
```

3.6 Proposición de Asignación.

El operador de asignación permite evaluar una expresión y asignar el resultado en una variable. (CORONA NAKAMURA & ANCONA VALDEZ, 2011)

Su sintaxis es: Identificador = expresión;

Con la asignación anterior le indicamos a la computadora que evalúe la expresión y la almacene en la variable que define el identificador.

Ejemplos:

```
Num = 3;          calif1 = 8.5;          res_1 = 4 * 5;
suma = a + b;     letra = 'A';
```

3.7 Operadores, operandos y expresiones.

Un operador es un símbolo que permite relacionar dos datos en una expresión y evaluar el resultado de la operación. (CORONA NAKAMURA & ANCONA VALDEZ, 2011)

Los operadores se utilizan para realizar operaciones, entre los que se encuentran los operadores aritméticos, relacionales, lógicos, también existen los operadores aritméticos simplificados y los operadores de incremento y decremento.

Los operadores aritméticos nos sirven para realizar operaciones entre operandos (datos): números, constantes o variables. El resultado de una operación aritmética siempre es un número. En la tabla 3.4 se presentan algunos ejemplos de su uso, considere que *x* es una variable de tipo entero y *v* es una variable de tipo real.

Tabla 3.4. Operadores aritméticos.

Operador aritmético	Operación	Ejemplos	Resultados
+	Suma	*x* = 2.5 + 3; *v* = 2.5 + 3;	*x* = 5 *v* = 5.5
-	Resta	*x* = 5.5 - 3; *v* = 5.5 – 3;	*x* = 2 *v* = 2.5
*	Multiplicación	*x* = 5.5 * 3; *v* = 5.5 * 3;	*x* = 15 *v* = 16.5

/	División	*x* = 5 / 4;
		x = 5.0 / 4.0;
		v = 5 / 4;
		v = 5.0 / 4;
			x = 1
			x = 1
			v = 1.0
			v = 1.25

%	División con residuo	*x* = 15 % 2; *v* = (15 % 2) / 2;	*x* = 1 *v* = 0.0

Cuando una expresión contiene varios operadores aritméticos debemos respetar la jerarquía de los operadores y aplicarlos de izquierda a derecha. Si una expresión contiene subexpresiones entre paréntesis, éstas se evalúan primero.

En la tabla 3.5 se muestra la jerarquía de los operadores aritméticos de mayor a menor en orden de importancia.

Tabla 3.5. Jerarquía de los operadores aritméticos.

Operador	Operación
*, /, %	Multiplicación, división, división con residuo
+, -	Suma, resta

Operadores aritméticos simplificados es una forma de simplificar el uso de los operadores aritméticos. En la tabla 3.6 se muestra la forma simplificada de su uso, considere que *x* es una variable de tipo entero.

Tabla 3.6 Ejemplos de operadores simplificados.

Operador aritmético	Forma simplificada de uso	Ejemplo	Equivalencia
+	+=	*x* = 6; *x* += 5;	*x* = 6; *x* = *x* + 5;
-	-=	*x* =10; *x* -= 3;	*x* = 10; *x* = *x* - 3;
*	*=	*x* = 5; *x* *= 3	*x* = 5; *x* = *x* * 3;
/	/=	*x* = 25; *x* /= 3;	*x* = 25; *x* = *x* / 3;
%	%=	*x* = 20; *x* %= 12;	*x* = 20; *x* = *x* % 12;

Los operadores de incremento (++) y decremento (--) son propios del lenguaje C y se utiliza para simplificar y clarificar la escritura de los programas. Se pueden usar antes o después de la variable. Los resultados son diferentes como se puede apreciar en la tabla 3.7. Considere que *x* y *y* son variables de tipo entero.

Tabla 3.7. Operadores de incremento y decremento.

Operador	Operación	Ejemplos	Resultados

++	Incremento	*x* = 7;
		y = *x*++;
		
		x = 7;
		y = ++*x*;
			x = 7
			y = 7
			x = 8
			
			x = 7
			y = 8
			x = 8

--	Decremento	*x* = 6; *y* = *x*--; *x* = 6; *y* = --*x*;	*x* = 6 *y* = 6 *x* = 5 *x* = 6 *y* = 5 *x* = 5

Las expresiones lógicas o booleanas, están constituidas por números, constantes o variables y operadores lógicos o relacionales. El valor que pueden tomar estas expresiones es 0 (cero) en caso de ser falso y diferente a cero en caso de ser verdadero. Se utilizan frecuentemente en las estructuras selectivas o en las repetitivas. En las estructuras selectivas se emplean para seleccionar un camino determinado y en las estructuras repetitivas se usan para determinar si se continúa con el ciclo o se interrumpe el mismo.

Los operadores relacionales se utilizan para comparar dos operandos, que pueden ser números, caracteres, cadena de caracteres, constantes o variables, éstos a su vez pueden ser de los tipos expresados anteriormente. En la tabla 3.8 se pueden observar los operadores relacionales que se utilizan en lenguaje C.

Tabla 3.8. Operadores relacionales.

Operador relacional	Operación
= =	Igual a
! =	Diferente de
<	Menor que
>	Mayor que
< =	Menor o igual que
>=	Mayor o igual que

En la tabla 3.9 se pueden ver algunos ejemplos de cómo utilizar los operadores relacionales.

Tabla 3.9 Ejemplos del uso de operadores relacionales.

Ejemplo	**Resultado**	**Valor binario**
res = ‘j’ == ‘z’;	Falso	res = 0
res = ‘v’ != ‘x’;	Verdadero	res = 1
res = 3 < 19;	Verdadero	res = 1
res = 62 > 11;	Verdadero	res = 1
res = 18 <= 6;	Falso	res = 0
res = 45 >= 10;	Verdadero	res = 1

Los operadores lógicos permiten formular condiciones complejas a partir de condiciones simples, tales como, conjunción (&&), disyunción (||) y negación (!).

La tabla de verdad de estos operadores se presenta en la tabla 3.10 con las posibles combinaciones y resultados.

Tabla 3.10 Tabla de verdad de los operadores lógicos.

A	B	(!A)	(!B)	(A || B)	(A && B)
Verdadero (1)	Verdadero (1)	Falso (0)	Falso (0)	Verdadero (1)	Verdadero (1)
Verdadero (1)	Falso (0)	Falso (0)	Verdadero (1)	Verdadero (1)	Falso (0)
Falso (0)	Verdadero (1)	Verdadero (1)	Falso (0)	Verdadero (1)	Falso (0)
Falso (0)	Falso (0)	Verdadero (1)	Verdadero (1)	Falso (0)	Falso (0)

En la tabla 3.11 se presentan algunos ejemplos de los operadores lógicos.

Tabla 3.11. Ejemplos del uso de los operadores lógicos.

Ejemplo	Resultado	Valor binario
((8 > 5) && (10 < 4))	Falso	0
((16 > 15) && (6 > 3))	Verdadero	1
((6 > 3) || (9 < 4))	Verdadero	1
((7 > 10) || (5 < 12))	Verdadero	1
!(7 > 2)	Falso	0
!(11 < 5)	Verdadero	1

Operador condicional (?:)

Este operador sustituye la sentencia **if…else**. La sintaxis es la siguiente:

<Expresión booleana> ? <instrucción_si> : <instrucción_no>

Ejemplo: a = (b < 2) ? 5 : 10;

es equivalente a:

```
if (b < 2)  a = 5;
    else a = 10;
```

La variable a toma el valor de 5 cuando b < 2, y el 10 en caso contrario.

3.8 Prioridad de operadores, evaluación de expresiones.

Se llama **orden de prioridad** al orden en que la computadora realiza las diferentes operaciones.

Prioridad de los operadores:

- Paréntesis ( ). Todas las expresiones entre paréntesis se evalúan primero. Las expresiones con paréntesis anidados se evalúan de adentro hacia afuera.

- Dentro de una misma expresión o subexpresión, se evalúan en el siguiente orden:
  Potencia (^ o **)
  *, /, %
  +, -

Los operadores en una misma expresión con igual nivel de prioridad se evalúan de acuerdo a la tabla 3.12, en ella se aprecia la prioridad de todos los operadores en lenguaje C.

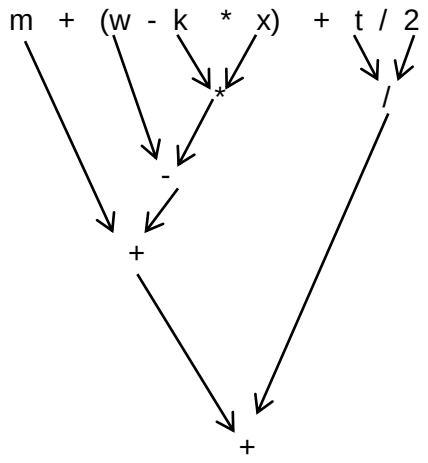
Tabla 3.12 Prioridad de los operadores en lenguaje C.

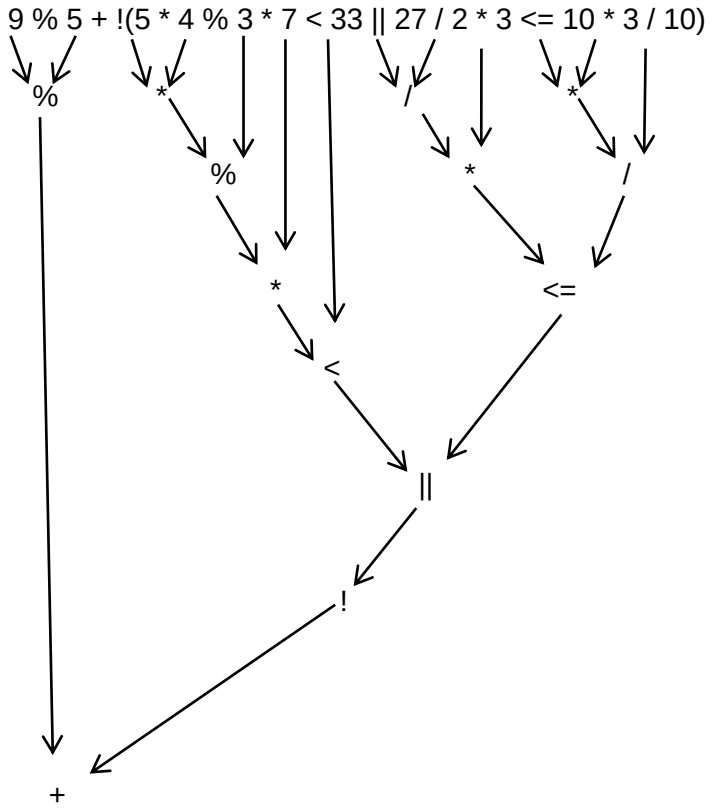
Categoría del operador	Operadores	Regla asociativa
Agrupación	( )	Izquierda a derecha
Operadores monarios	++, --, !,	Derecha a izquierda
Multiplicación, división y residuo	*, /, %	Izquierda a derecha
Suma y resta aritmética	+, -	Izquierda a derecha
Operadores de relación	<, <=, >, >=	Izquierda a derecha
Operadores de igualdad	= =, !=	Izquierda a derecha
Conjunción	&&	Izquierda a derecha
Disyunción	||	Izquierda a derecha
Operador condicional	?	Derecha a izquierda
Operadores de asignación	=, +=, -=,*=, /=, %=	Derecha a izquierda

Una expresión es el resultado de unir operandos mediante operadores. Los operandos pueden ser variables, constantes u otras expresiones; y los operadores, aritméticos, lógicos o relacionales. El resultado de una expresión es un dato numérico o un valor

lógico. Para agrupar las expresiones utilizamos los paréntesis. (CORONA NAKAMURA & ANCONA VALDEZ, 2011)

En los siguientes ejemplos se observa cómo se evalúa una expresión matemática en lenguaje C:





Conversión de tipos.

Cuando se evalúa una expresión aritmética que contiene diferentes tipos de datos, el compilador convierte todos a un tipo único. La conversión se hace operación a operación y se lleva a cabo según el tipo que esté más a la derecha de la siguiente relación:

char → int → unsigned int → long → unsigned long → float → double → long double

Ejemplo:

```
char c;
int i;
long l;
float f;
double d;
x = (i * c % l) – (d / f) + f;
```

Tomando en cuenta el orden de prioridad, primero se evalúa el contenido de los paréntesis, por lo tanto en los primeros paréntesis evaluamos los tipos de izquierda a derecha:

i * c es un *int*, ya que el tipo int está a la derecha del tipo char en la relación.
i % l es un *long*, el tipo long está a la derecha del tipo int de la relación.
d /f es un *double*, ya que el tipo double está a la derecha del tipo float.

Evaluando nuevamente de izquierda a derecha y queda de la siguiente forma:

long – double + float = double + float = double

A continuación se analizará el siguiente programa para una mejor comprensión

```
#include <stdio.h>
main()
{
        printf("división entera: 5/2 es % 7d\n", 5/2);
        printf("división flotante: 5.0/2.0 es % 7.2f\n", 5.0/2.0);
        printf("división mixta: 5.0/2 es % 7.2f\n", 5.0/2);
}
```

El resultado de la primera división es 2, ya que los dos valores son enteros; en las dos últimas divisiones el resultado será 2.50 ya que es suficiente que un valor sea real.

Conversión forzada de tipos de datos.

Además de la conversión automática de tipos de datos, En lenguaje C es posible forzar la conversión de un tipo de dato en otro diferente. A esta conversión forzada se le llama *casts.* La sintaxis es: *(tipo) expresión.*

En el siguiente fragmento de programa se ejemplifica la conversión forzada:

```
int a = 3, b = 2;
float c, d;
c = a / b;          //el tipo de c es int, ya que a y b son tipo int; el resultado es 1
d = (float) a /b;   //el valor de d es 1.5
```

3.9 Elaboración de programas.

1.- Realizar un programa en lenguaje C, que lea dos números enteros, los sume e imprima el resultado de la suma.

```
#include <stdio.h>
#include <conio.h>
```

Declaración de archivos de cabecera

```
int num1, num2, res;
```

Declaración de variables

```
main()
{
    printf("Introduce un número ");
    scanf("%d",&num1);
    printf("Introduce otro número ");
    scanf("%d",&num2);
    res = num1 + num2;
    printf("El resultado de la suma es: %d", res);
    getch();
}
```

Programa principal

Cuerpo del programa
La llave { indica el inicio del cuerpo del programa, y la llave } indica el fin del cuerpo del programa

La función *printf()* pertenece a la biblioteca estándar de C y se encuentra en el archivo *stdio.h*, Esta función sirve para imprimir en pantalla como se aprecia en el programa anterior en la sentencia: printf("Introduce un número ");

La función *scanf()* pertenece a la biblioteca estándar de C (como *printf()*), y permite leer datos desde el teclado y almacenarlos en una variable.
La sentencia scanf("%d",&num1);

Consta de dos partes:

- El nombre de la función: *scanf()*.
- Los argumentos. Son valores que aparecen entre paréntesis separados por una coma, en este caso hay dos: “%d” y &num1.

El primer argumento siempre va dentro de las comillas: *%d*, le indica a *scanf()* que tome del teclado un número entero. El segundo argumento, *&num1*, indica en qué variable se almacenará el dato leído. El símbolo *&* es necesario para *scanf()* ya que indica la dirección de la memoria en donde se ubica la variable en cuestión.

2.- Realizar un programa en lenguaje C, que lea el valor del radio de un círculo, calcule el área e imprima ésta.

```
#include <stdio.h>
#include <conio.h>

const float PI = 3.1416;
float radio, area;

main()
{
    printf("Introduce el valor del radio ");
    scanf("%f",&radio);
    area = PI * radio * radio;
    printf("El área del circulo es %f ",area);
    getch();
}
```

3.- Realizar un programa en lenguaje C, que lea una cantidad en pulgadas y la convierta en centímetros e imprima el resultado.

```
#include <stdio.h>
#include <conio.h>

float cm;
int pulg;

main()
{
    printf("Conversión de pulgadas a centímetros\n ");
    printf("Introduce las pulgadas ");
    scanf("%d",&pulg);
```

```
    cm = pulg * 2.54;
    printf("%d pulgadas = %7.3f centímetros\n ", pulg, cm);
    getch();
}
```

4.- Realizar un programa en lenguaje C, que introduzca el año de nacimiento de una persona y calcule e imprima la edad de ésta.

```
#include <stdio.h>
#include <conio.h>

int an, ac, edad;

main()
{
    printf("Introduce el año de nacimiento ");
    scanf("%d",&an);
    printf("Introduce el año actual ");
    scanf("%d",&ac);
    edad = ac - an;
    printf("La edad de la persona es %d ",edad);
    getch();
}
```

5.- Realizar un programa en lenguaje C, que realice lo siguiente: Una tienda vende libros a $100.00, cuadernos a $15.50 y plumas a $ 2.35. Calcule e imprima el monto total de la venta según el número de artículos vendidos.

```
#include <stdio.h>
#include <conio.h>

#define PLIB 100
#define PCUAD 15.50
#define PPLUM 2.35

int lib, cuad, plum;
float vtotal;

main()
{
    printf("Total de libros vendidos\n ");
    scanf("%d",&lib);
    printf("Total de cuadernos vendidos\n ");
```

```
    scanf("%d",&cuad);
    printf("Total de plumas vendidas\n ");
    scanf("%d",&plum);
    vtotal = lib * PLIB + cuad * PCUAD + plum * PPLUM;
    printf("La venta total fue de: $ %.2f\n ",vtotal);
    getch();
}
```

Ejercicios para practicar

Elaborar programa en lenguaje C de los siguientes ejercicios:

1.- Leer el radio de una esfera y calcular e imprimir el volumen.
2.- Calcular e imprimir el área y perímetro de un rectángulo.
3.- Leer el lado de un triángulo equilátero. Calcular e imprimir el perímetro, la altura y el área de dicho triángulo.
4.- Calcular e imprimir el importe total a pagar de la compra de tres productos considerando que hay un descuento de 10% al total de la compra.
5.- Leer una cantidad de pesos (número entero) y calcular e imprimir a cuántos billetes de 1000, 500, 200, 100, 50, 20 y monedas de 10, 5, 1 peso equivale.
6.- Leer el precio por hora y horas trabajadas de un empleado. Calcular e imprimir el sueldo semanal.
7.-Leer la clave de un empleado y los seis primeros sueldos del año, calcule el ingreso total semestral y el promedio mensual, e imprima la clave del empleado, el ingreso total y el promedio mensual.
8.- Leer el radio de un círculo, calcule e imprima tanto su área como la longitud de su circunferencia.
9.- Leer una cantidad en pesos y convertirla en dólares.
10.- Considere las variables: int i=7, j=4, k; float f; char c, realice la corrida a mano de las siguientes expresiones, escribiendo a la derecha los valores (siga el orden establecido).

1. i += j++;	i =	j =
2. j += ++i + 7;	j =	i =
3. k = (j++ < i) ? i – 2:j*2;	k =	j =
4. j *= i + k;	j =	
5. j /= i – 3;	j =	
6. j = i++ - 5;	j =	i =
7. j += i % (k – 1);	j =	
8. f = (float) i / 3;	f =	
9. k = j – ‘7’ + 2;	k =	
10. c = 2 * j + 1/2;	c =	
11. k = 3 != 7;	k =	
12. k = j =5;	k =	j =
13. i + j = k;	k =	

4. ESTRUCTURAS SELECTIVAS Y DE REPETICIÓN.

La programación estructurada produce programas más fáciles de entender y, por lo tanto, más fáciles de probar, depurar, modificar, e incluso comprobar en un sentido matemático. (DEITEL & DEITEL, 2004)

El lenguaje C tiene siete instrucciones de control; Secuencia, tres tipos de selección y tres tipos de repetición. Cada programa en lenguaje C está formado por la combinación de tantas instrucciones de control como sea adecuado para el algoritmo que implementa el programa.

La estructura secuencial es directa la cual se explicó en el capítulo 2. La estructura de selección se implementa en una de las siguientes formas:

Instrucción **if** (selección simple).
Instrucción **if…else** (selección doble).
Instrucción **switch** (selección múltiple).

La estructura de repetición se implementa en una de las tres siguientes formas:

Instrucción **while**.
Instrucción **do…while**.
Instrucción **for**.

4.1 Selectiva simple.

La estructura selectiva if permite que el flujo de un programa siga por un camino específico si se cumple una condición determinada. Si al evaluar la condición el resultado es verdadero, entonces se sigue por un camino específico y se ejecuta una operación o acción o un conjunto de ellas. Por otra parte, si el resultado de la evaluación es falso, entonces se pasa(n) por alto esa(s) operación(es). En ambos casos se continúa con la secuencia normal del programa. (CAIRÓ BATTISTUTTI, 2006)

Observe la figura 4.1 que representa la estructura selectiva mediante un diagrama de flujo.

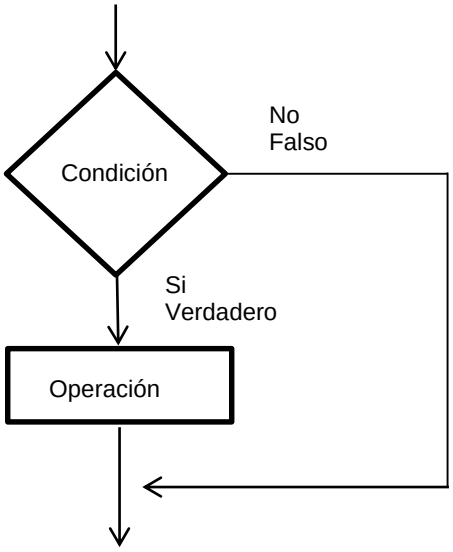


Figura 4.1. Estructura selección simple if.

En lenguaje C la sintaxis de la estructura de selección simple es de la siguiente forma:

if (condición)
instrucción 1;

Para ejemplificar la estructura de selección simple se escribirá un programa en lenguaje C, que al recibir como dato el promedio de un alumno, imprima “aprobado” en caso de que el promedio sea mayor o igual a 70.

```
#include <stdio.h>
#include <conio.h>

int prom;

main()
{
     printf("Introduce el promedio:  ");
     scanf("%d", &prom);
     if (prom >= 70)
        printf ("\n APROBADO");
     getch();
}
```

Si se requiere ejecutar un bloque de instrucciones, cuando se cumpla la condición entonces la sintaxis es la siguiente:

```
if (condición)
        {
                instrucción 1;
                instrucción 2;
                instrucción 3;
        }
```

Tomando el programa anterior, si además deseamos que imprima el promedio que obtuvo el alumno, la estructura quedaría de la siguiente forma:

```
if (prom >= 70)
    {
      printf ("\n APROBADO");
      printf ("\nEl promedio es: %d ",prom);
    }
```

Si existe más de una instrucción para realizar, es necesario utilizar inicio y fin para agrupar las instrucciones, es su alcance sintáctico, si no se usa el inicio y fin sólo se ejecuta la primera instrucción, mientras que las siguientes instrucciones se realizarán siempre. (CORONA NAKAMURA & ANCONA VALDEZ, 2011)

Al final del capítulo en el inciso 4.8 se encuentran los programas para practicar.

4.2 Selectiva doble.

Estructura de control que dirige a la computadora para ejecutar una acción si la condición es verdadera, y otra acción en caso de que sea falsa. (CORONA NAKAMURA & ANCONA VALDEZ, 2011)

Existen dos formas de representarlo, dependiendo del número de instrucciones que se desean realizar si la condición se cumple o no. La sintaxis de las dos formas se presenta a continuación:

Forma 1

```
if (condición)
        instrucción1;
else
        instrucción 2;
```

Forma 2

```
if(condición)
        {
                instrucción 1;
                instrucción 2;
        }
else

        {
                instrucción 3;
                instrucción 4;
        }
```

En el siguiente fragmento de programa se representa la forma 1:

```
if (prom>=70)
        printf (“ACREDITADO”);
else
        printf (“NO ACREDITADO”);
```

Ejemplo: Suponga que el precio unitario de las pelotas de béisbol es de $6.50 si se compran 10 o más y $7.00 en caso de comprar menos. El programa quedaría de la siguiente forma:

```
#include <stdio.h>
#include <conio.h>

int num;
float total;

main()
{
    printf("Introduce el número de pelotas a comprar:  ");
    scanf("%d", &num);
    if (num >= 10)
      total = num * 6.50;
    else
       total =num * 7.00;
    printf ("\nEl total a pagar es: $%.2f ",total);
    getch();
}
```

Al ejecutar el programa, si se introduce un valor de 10, el resultado es $65.00 como lo puede observar en la figura 4.2. Por el contrario si el valor que se introduce es 8 el resultado será $56.00 (figura 4.3).

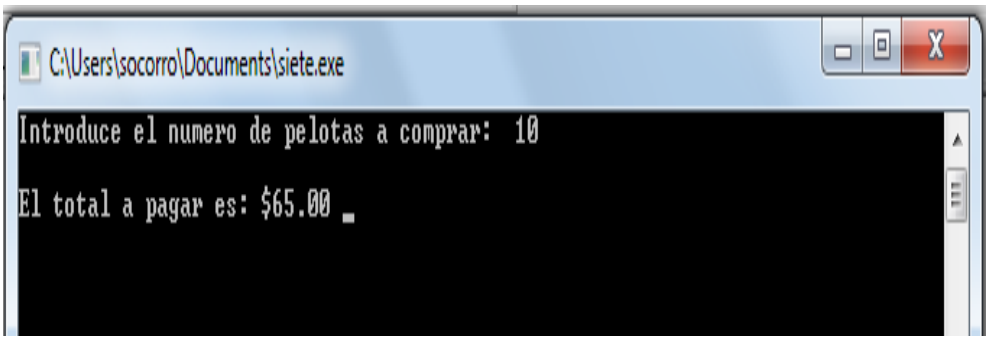


Figura 4.2. La condición if se cumple.

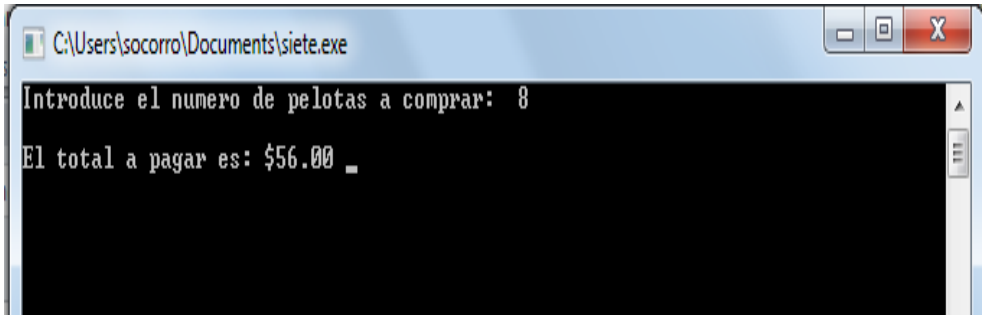


Figura 4.3. La sentencia else se ejecuta.

La forma 2 se representa con el siguiente ejemplo: Hacer un programa que lea dos números e imprima el número mayor así como el texto “El primer número es el mayor” o “El segundo número es el mayor”. Los números deben ser diferentes.

```
#include <stdio.h>
#include <conio.h>

int num1,num2;

main()
{
    printf("Introduce el primer número ");
    scanf("%d", &num1);
    printf("Introduce el segundo número ");
    scanf("%d", &num2);
    if (num1 > num2)
        {
```

```
            printf("El número mayor es %d\n", num1);
            printf("El primer número es el mayor\n");
        }
    else
        {
            printf("El número mayor es %d\n", num2);
            printf("El segundo número es el mayor\n");
        }
    getch();
}
```

En la figura 4.4 se puede observar el resultado si se cumple la condición, en caso contrario la figura 4.5 muestra los resultados que se obtiene.

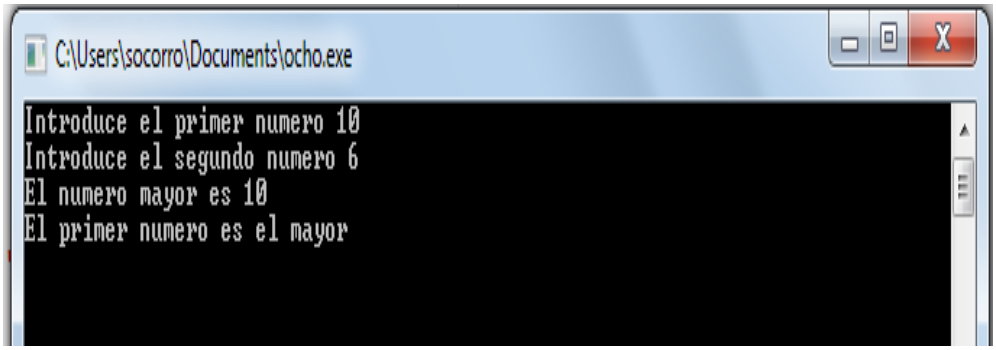


Figura 4.4. La condición if se cumple y realiza bloque de instrucciones.

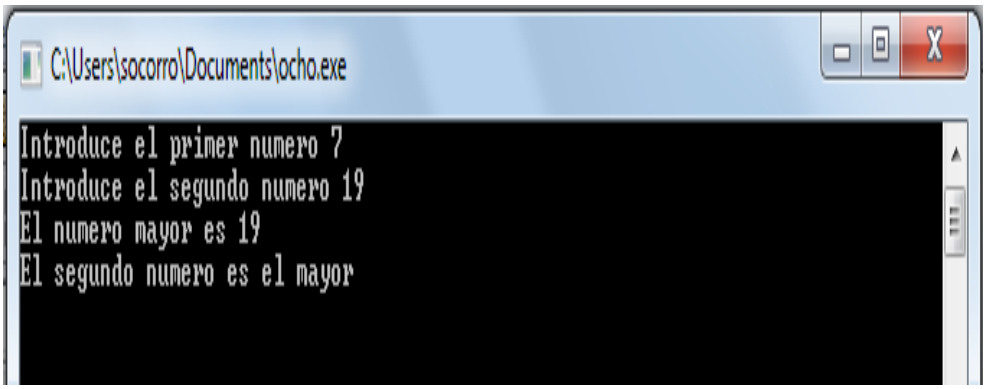


Figura 4.5. Ejecución del bloque de instrucciones de la sentencia else.

Al final del capítulo en el inciso 4.8 se encuentran los programas para practicar.

4.3 Selectiva anidada.

El anidamiento o escalonamiento consiste en insertar (anidar) una estructura dentro de otra.

La expresión “if anidados” se refiere a que podemos utilizar una sentencia **if** dentro de otra sentencia **if**. Esto se emplea cuando se tienen que cumplir varias condiciones para poder ejecutar una acción.

Un if anidado es una sentencia **if** que es el objeto de otro **if** o **else**. La razón por la que los if anidados son tan problemáticos es que resulta confuso saber qué **else** se asocia con qué **if**.

A continuación se presenta la sintaxis del if anidado:

```
if (condición 1)
        Instrucción 1;
else
        if (condición 2)
                Instrucción 2;
        else    if (condición 3)
                        Instrucción 3;
                        . . .
                else
                        . . .
        instrucción n;
```

En el siguiente ejemplo se aplica la estructura if anidada: Hacer un programa que al ejecutarlo presione una tecla e imprima que tipo de tecla es (letra mayúscula, letra minúscula, dígito o carácter especial).

```
#include <stdio.h>
#include <conio.h>

char dato;

main()
{
    printf("OPRIME UNA TECLA:     ");
    scanf("%c", &dato);
    if((dato >= 48)&& (dato <= 57))
        printf("\nES UN DIGITO");
```

```
    else
       if((dato >= 65)&&(dato <=90))
              printf("\nES UNA LETRA MAYUSCULA");
       else
          if((dato >= 97)&& (dato <= 122))
                 printf("\nES UNA LETRA MINUSCULA");
          else
               printf("\nES UN CARACTER ESPECIAL");
    getch();
}
```

En el programa se utilizan tres estructuras selectivas dobles anidadas para representar las cuatro opciones posibles. El código ASCII de los dígitos está entre 48 y 57, las letras mayúsculas están entre el 65 y el 90, las letras minúsculas entre el 97 y el 122 y los caracteres especiales están entre varios rangos: del 0 al 47, 58 al 64, 91 al 96, el último rango del 123 al 255.

La primera condición revisa si la variable dato es un dígito; si es, imprime ES UN DIGITO; si no es entonces revisa si es una letra mayúscula; si es imprime ES UNA LETRA MAYUSCULA; si no es mayúscula, entonces revisa si es minúscula; si es, imprime ES UNA LETRA MINUSCULA, y por consiguiente si no fue dígito, mayúscula o minúscula entonces imprime ES UN CARÁCTER ESPECIAL. La figura 4.6 muestra un posible resultado.

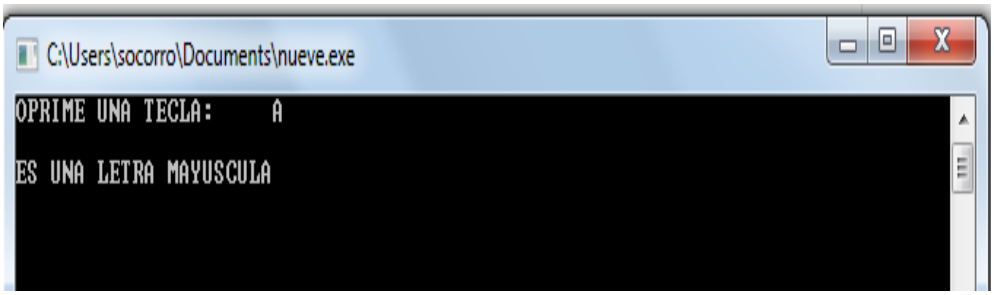


Figura 4.6. Ejecución del programa.

4.4 Selectiva múltiple.

En ocasiones, un algoritmo contiene series de decisiones en las que se evalúan una variable o expresión de manera separada para cada uno de los valores integrales constantes que puede asumir, y se llevan a cabo diferentes acciones. A esto se le llama selección múltiple. (DEITEL & DEITEL, 2004)

El lenguaje C proporciona la instrucción de selección múltiple **switch**, para manejar la toma de decisiones. La instrucción **switch** consiste en una serie de etiquetas **case** y un caso opcional **default**.

La sintaxis de la estructura selectiva múltiple **switch** en lenguaje C se presenta a continuación:

```
switch (selector)
{
        case  <valor1>          : <acción1>;
                        break;
        case  <valor2>          : <acción2>;
                        break;
        case  <valorN>          : <acciónN>;
                        break;
}
acciónN+1;
```

Es importante señalar que la estructura selectiva **switch** es muy flexible, lo que permite que se pueda aplicar de diferentes formas. Por ejemplo, si el selector tomara un valor distinto de los comprendidos entre 1 y N, entonces se debe seguir el camino etiquetado con **default**. La sintaxis es la siguiente:

```
switch (selector)
{
        case  <valor1>          : <acción1>;
                        break;
        case  <valor2>
        case  <valor3>
        case  <valor4>          : <acción2>;
                        break;
        case  <valor5>
        case  <valor6>          : <acción3>;
                        break;
        default                 : <acciónx>;
                        break;
}
acciónY;
```

Ejemplo: Hacer un programa en lenguaje C que, al recibir como datos dos variables de tipo entero, obtenga el resultado de la siguiente función:

$$f(T)=\begin{cases} T/5 & \text{Si OP = 1} \\ T ** T & \text{Si OP = 2} \\ 6 * T/2 & \text{Si OP = 3, 4} \\ 1 & \text{Para cualquier otro caso} \end{cases}$$

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

int OP, T;
float RES;

main()
{
    printf("ingrese la opción del cálculo y el valor entero:  ");
    scanf("%d %d", &OP,&T);
    switch(OP)
    {
        case 1: RES = T/5;
            break;
        case 2: RES = pow(T,T); /*la función pow está definida en la biblioteca math.h*/
            break;
        case 3:
        case 4: RES = 6 * T/2;
            break;
        default: RES = 1;
             break;
    }
    printf("El resultado:     %7.2f",RES);
    getch();
}
```

En la figura 4.7 se puede observar la ejecución del programa, se introducen los valores 1 y 10 con la finalidad de verificar el resultado para el caso 1. Al terminar de hacer las operaciones del caso 1 sale de la estructura selectiva múltiple switch, debido a la sentencia break.

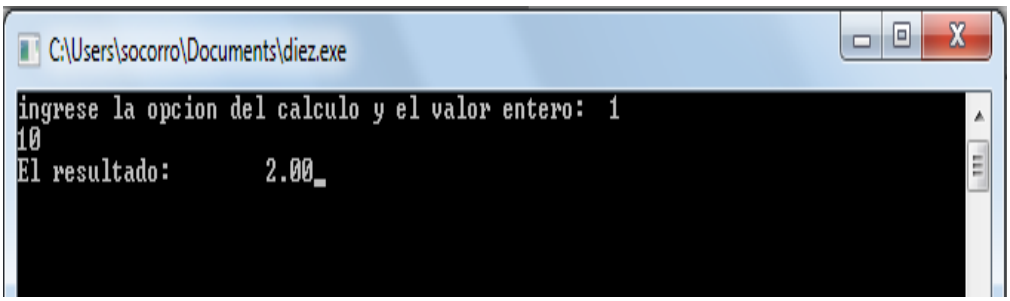


Figura 4.7. Caso 1.

En la figura 4.8 se presenta el caso 2 se prueba con los valores 2 y 5. Al igual que el caso 1 al encontrar sentencia break sale de la estructura selectiva múltiple switch.

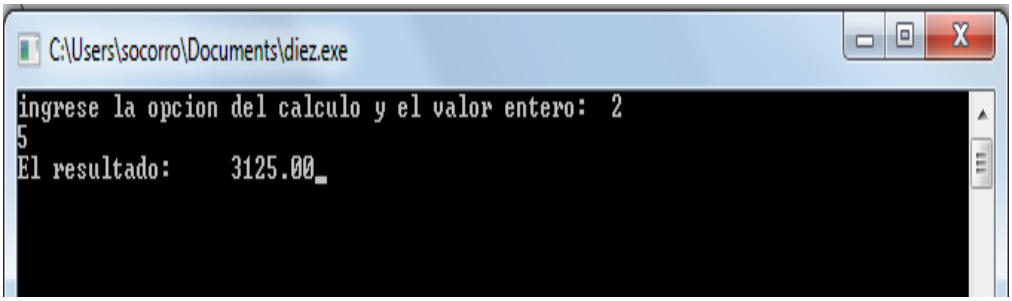


Figura 4.8. Caso 2.

Para el caso 3 o 4 el programa ejecutará las mismas operaciones, se introducen los valores 4 y 20, el resultado se muestra en la figura 4.9.

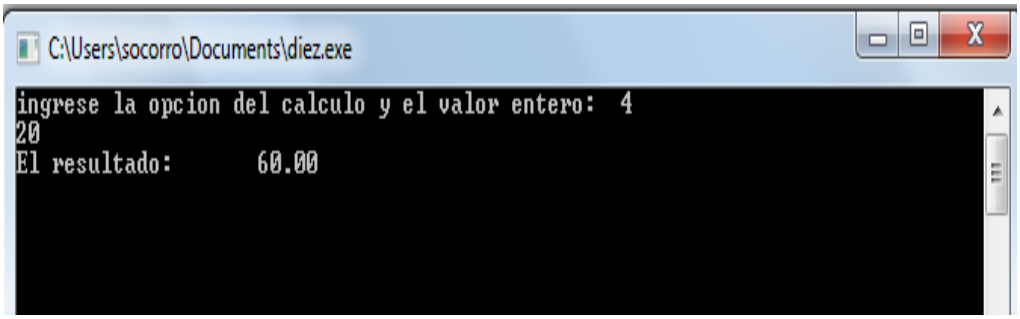
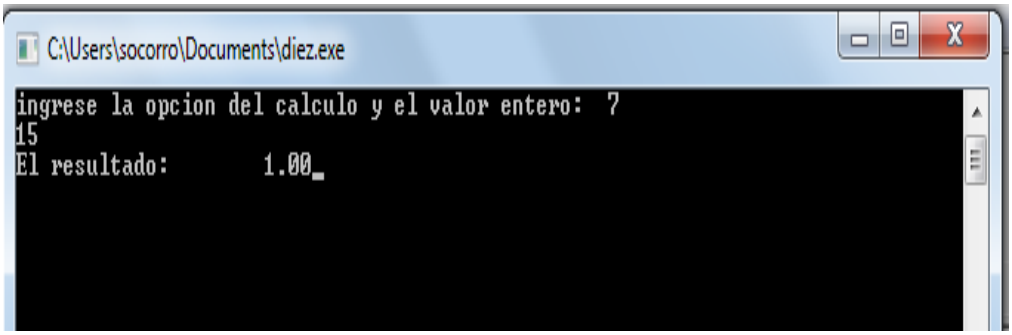


Figura 4.9 Caso 3 o 4.

Por último, se realiza la sentencia **default**, esto ocurre cuando la opción seleccionada no se encuentra en ninguno de los casos. Por ejemplo al introducir la opción de cálculo 7, ésta no existe en ninguno de los casos por lo tanto se ejecuta el default (figura 4.10).



4.10. Ejemplo del default.

4.5 Repetir mientras.

La mayor parte de los programas incluyen repeticiones o ciclos. Un ciclo es un grupo de instrucciones que la computadora ejecuta en forma repetida, en tanto se conserve verdadera alguna condición de continuación del ciclo. (DEITEL & DEITEL, 2004)

El ciclo **while** permite controlar a la computadora para que se ejecute la sentencia que hay dentro del ciclo mientras se evalué como verdadera una condición. El ciclo puede incluir una sola sentencia o un bloque de sentencias. Después de ejecutar el bloque de sentencias, la computadora evalúa nuevamente la condición. Si la condición es verdadera, entonces la computadora ejecuta el ciclo de nuevo. Tan pronto como la condición sea falsa, la computadora deja de ejecutar las sentencias que contenga el ciclo y continuará ejecutando el programa desde la sentencia que haya a continuación del ciclo. Para controlar un ciclo **while** se puede utilizar un **contador** o un **centinela**.

Repetición controlada por un contador.

La repetición controlada por un contador se denomina a veces *repetición definida*, porque con anticipación se sabe con exactitud cuántas veces se ejecutará el ciclo. Se utiliza *una variable de control* para contar el número de repeticiones. La variable de control es incrementada (normalmente en 1), cada vez que se ejecuta el grupo de instrucciones. Cuando el valor de la variable de control indica que se ha ejecutado el número correcto de repeticiones, se termina el ciclo y la computadora continúa ejecutando el enunciado siguiente al de la estructura de repetición. (DEITEL & DEITEL, 2004)

Por lo tanto, la repetición controlada por un contador requiere:

1. El *nombre* de una variable de control (o contador del ciclo).
2. El *valor inicial* de la variable de control.
3. El *incremento (o decremento)* con el cual, cada vez que se termine un ciclo, la variable de control será modificada.
4. La condición que compruebe la existencia del *valor final* de la variable de control (es decir, si se debe o no seguir con el ciclo).

En la estructura **while**, la condición de continuación de ciclo se prueba al principio del ciclo, antes de ejecutarse el cuerpo del mismo.

Su sintaxis puede ser cualquiera de las siguientes:

```
while (condición)
        enunciado;
```

```
while (condición)
        {
        enunciado 1;
        enunciado 2;
        enunciado n;
}
```

A continuación para ejemplificar la repetición controlada por un contador se realizará un programa en lenguaje C que, calcule e imprima los números del 1 al 10.

```
#include <stdio.h>
#include <conio.h>

int contador=1;                /*variable de control con valor inicial*/

main()
{
    printf("IMPRESION DE LOS NUMEROS DEL 1 AL 10\n\n");
    while (contador <=10)              /*condición que evalúa si debe o no seguir el ciclo*/
    {
        printf("%4d\n",contador);
        ++contador;                        /*incremento de la variable control*/
    }
    getche();
}
```

En la figura 4.11 se puede apreciar la ejecución del programa.

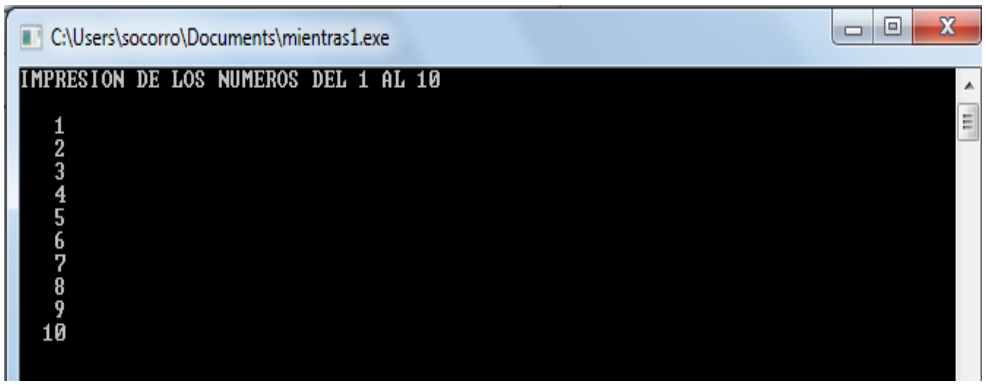


Figura 4.11. Ciclo mientras controlado por un contador.

Ejemplo: Hacer un programa en lenguaje C que, al recibir como datos las tres calificaciones de un alumno, calcule e imprima el promedio de éstas.

```
#include <stdio.h>
#include <conio.h>

int cal, suma=0, num=1;
float prom;

main()
{
    printf("C  A  L  I  F  I  C  A  C  I  O  N  E  S\n\n");
    while (num <= 3)      /*condición que evalúa si debe o no seguir el ciclo*/
    {
        printf("INTRODUCE LA CALIFICACION %d:  ",num);
        scanf("%d", &cal);
        suma = suma + cal;
        ++num;        /*incremento de la variable control*/
    }
    prom = suma/3;
    printf("\n EL PROMEDIO DEL ALUMNO ES %.2f ",prom);
    getche();
}
```

En la figura 4.12 se muestra una posible ejecución del programa anterior.

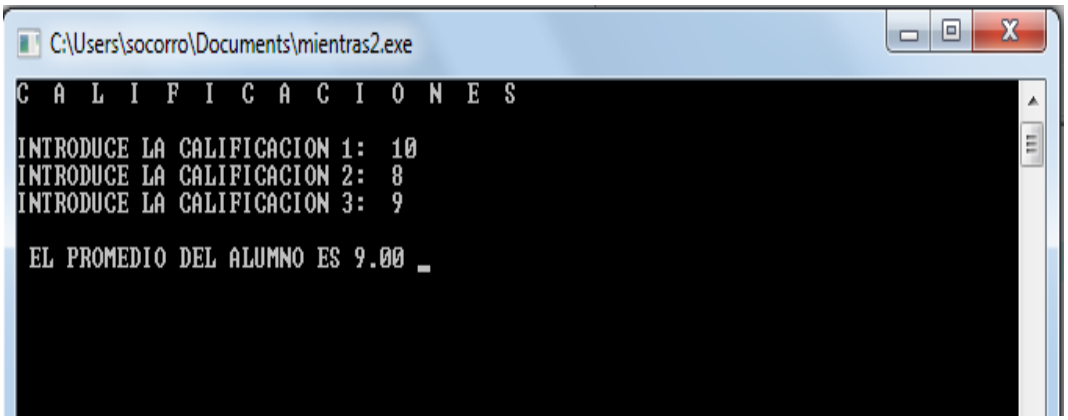


Figura 4.12.Ejecución del programa con repetición mientras.

Repetición controlada por un centinela.

La repetición controlada por un centinela a veces se denomina *repetición indefinida*, porque no se sabe con anticipación cuántas veces el ciclo se ejecutará. Los valores centinela se utilizan para controlar la repetición cuando el número preciso de repeticiones, no es conocido con anticipación, y el ciclo incluye enunciados que deben obtener datos cada vez que éste se ejecuta. El valor centinela indica “fin de datos”. El centinela es

introducido una vez que al programa se le han proporcionado todos los elementos normales de datos. Los centinelas deben ser diferentes a los elementos normales de datos. (DEITEL & DEITEL, 2004)

Para ejemplificar la repetición controlada por un centinela se realizará un programa en lenguaje C: Suponga que el usuario quiere encontrar la suma de 15 + 47 + 43 + 64 +29.

Con el siguiente programa, el usuario puede ingresar un número cada vez y después introducir el centinela de datos -1 para indicar que ya no hay más datos. La computadora no agregará el -1 a la suma, sino que lo tomará como una señal de salida del ciclo.

```
#include <stdio.h>
#include <conio.h>

int num, suma=0;

main()
{
    printf("Ciclo Mientras con centinela\n\n");
    printf("Escriba un numero o -1 para parar ");
    scanf("%d",&num);
    while (num != -1)
    {
        suma = suma + num;
        printf("Escriba un numero o -1 para parar ");
        scanf("%d",&num);
    }
    printf ("\nLa suma es: %d",suma);
    getch();
}
```

La ejecución del programa se muestra en la figura 4.13.

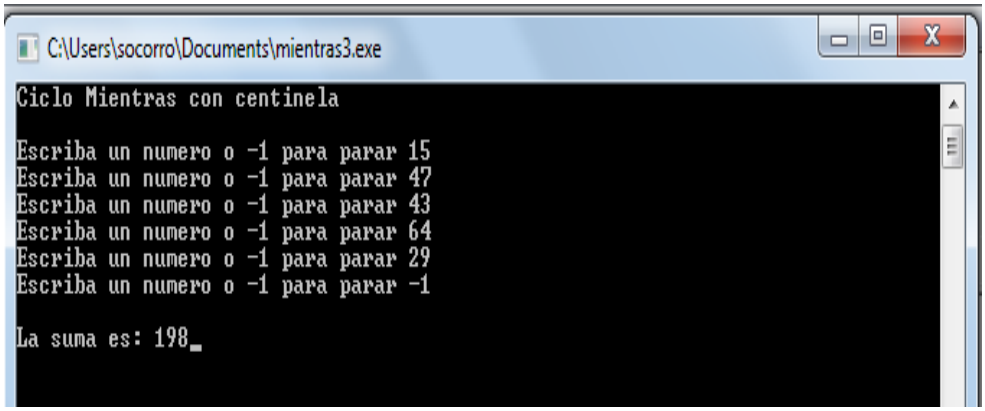


Figura 4.13. Ciclo controlado por un centinela.

En el siguiente programa se desea leer un dato numérico x cuyo valor ha de ser mayor que cero para calcular la función $f(x) = x * \log(x)$.

La variable *x_positivo*, se utiliza como un indicador que representa que el dato leído es mayor que cero. Por consiguiente, antes de que el ciclo se ejecute y se lea el dato de entrada, la variable *x_positivo* se inicializa a falso (0). Cuando se ejecuta el ciclo, éste debe continuar ejecutándose mientras el número leído sea negativo o cero y en consecuencia la variable *x_positivo* tenga el valor falso y se debe detener el ciclo cuando el número leído sea mayor que cero y en este caso el valor de la variable *x_positivo* se debe cambiar a verdadero (uno). En consecuencia la condición del ciclo debe ser *!x_positivo* ya que esta condición es verdadera cuando *x_positivo* es falso. A la salida del ciclo se calcula el valor de la función. (JOYANES AGUILAR, 2005)

El programa se escribe a continuación:

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

float f,x;
int x_positivo = 0;

main()
{
    while (!x_positivo)
    {
        printf("\n Valor de x: ");
        scanf("%f",&x);
        x_positivo = (x > 0.0);
    }
    f = x * log(x);
    printf("\nf(%.1f) = %.3e", x,f);
    getch();
}
```

La ejecución del programa se muestra en la figura 4.14.

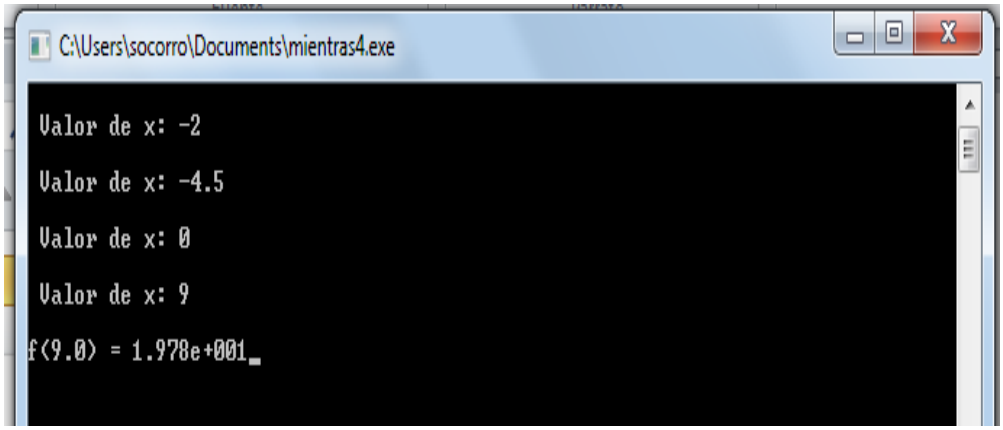


Figura 4.14. Programa que resuelve la función $f(x) = x * \log(x)$.

Ciclos controlados por tarea.

En la mayor parte de los casos, el programador tal vez quiera que la computadora siga ejecutando el cuerpo del ciclo mientras falta por concluir alguna tarea, aunque ni el programador ni el usuario saben cuántas ejecuciones serán necesarias. En tales casos, el programador debe diseñar una condición while para comprobar si la tarea ha sido terminada. (HENNEFELD, 1992)

Ejemplo: La suma de los cuadrados 1 + 7 + 9 + 16 +25 + …en cierto momento es mayor que 1000. Escriba un programa para encontrar un entero cuyo cuadrado hace que la suma sea mayor de 1000 y que despliegue la suma en ese momento.

```
#include <stdio.h>
#include <conio.h>

int n=0, suma=0;

main()
{
    while (suma <= 1000)
    {
        n = n + 1;
        suma = suma + (n * n);
    }
    printf("La suma empieza a pasar de 1000 cuando se agrega %d al cuadrado\n ",n);
    printf("\nLa suma es: %d\n",suma);
    getch();
}
```

La ejecución del programa se muestra en la figura 4.15.

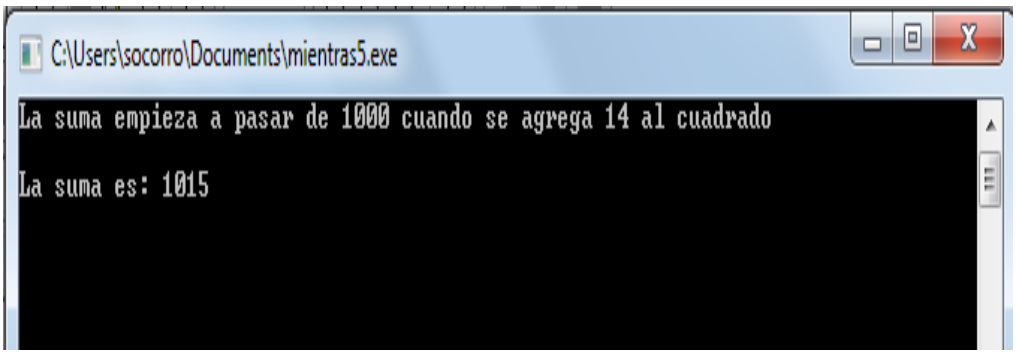


Figura 4.15. Ciclo controlado por una tarea.

4.6 Repetir hasta.

La estructura de repetición **do-while** es similar a la estructura while. La estructura **do-while** prueba la condición de continuación del ciclo, después de ejecutar el cuerpo del ciclo y, por lo tanto, el cuerpo del ciclo se ejecutará por lo menos una vez. Cuando termina **do-while**, la ejecución continuará con el enunciado que aparezca después de la cláusula while.

La sintaxis de **do-while** es la siguiente:

```
do {
        enunciado 1;
        enunciado 2;
        enunciado n;
while (condición);
```

En la estructura **do-while** no es necesario utilizar llaves, si en el cuerpo existe sólo un enunciado. Sin embargo, por lo regular las llaves se utilizan, para evitar confusión entre las estructuras **while** y **do-while**.

El ciclo **do-while** también puede ser controlado por un **contador** o por un **centinela**.
Repetición controlada por un contador.

A continuación para ejemplificar la repetición controlada por un contador se realizará el mismo programa en lenguaje C que se realizó en el tema anterior (4.5).
Calcule e imprima los números del 1 al 10 utilizando la estructura de repetición **do-while**.

```
#include <stdio.h>
#include <conio.h>

int contador=1;  /*variable de control con valor inicial*/

main()
{
     printf("IMPRESION DE LOS NÚMEROS DEL 1 AL 10\n\n");
     do{
          printf("%4d\n",contador);
          ++contador;                /*incremento de la variable control*/
       }while (contador <=10);       /*condición que evalúa si debe o no seguir el ciclo*/
     getche();
}
```

En la figura 4.16 se muestra la ejecución del programa, que es exactamente igual al programa con la estructura **while**.

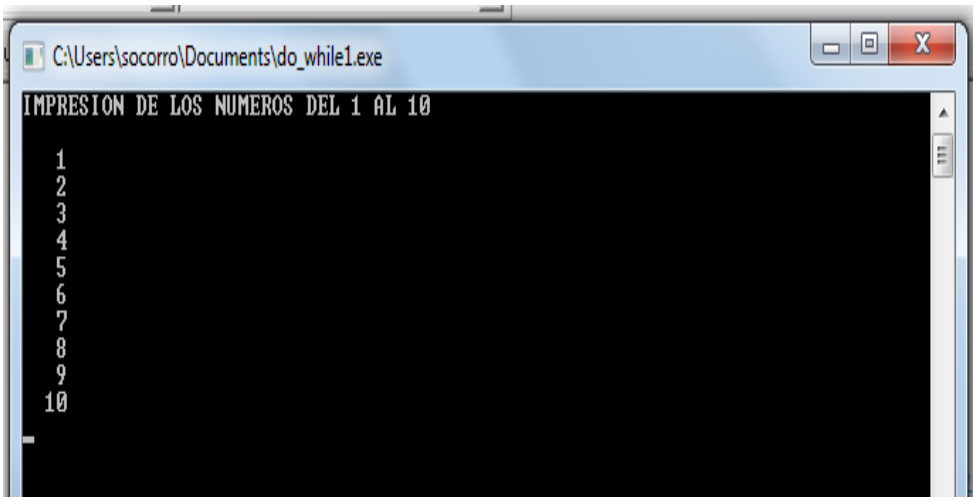


Figura 4.16. Estructura **do-while** controlada por un contador numérico.

Ejemplo: Hacer un programa en lenguaje C que, imprima las letras minúsculas del alfabeto utilizando la estructura **do-while** controlada por un **contador**.

```
#include <stdio.h>
#include <conio.h>

char car = 'a';  /*variable de control con valor inicial*/

main()
{
```

```
    printf("IMPRESION DE LAS LETRAS MINÚSCULAS DEL ALFABETO\n\n");
    do{
        printf("%4c\n",car);
        car++;              /*incremento de la variable control*/
      }while (car <= 'z');      /*condición que evalúa si debe o no seguir el ciclo*/
    getche();
}
```

La ejecución del programa se muestra en la figura 4.17.

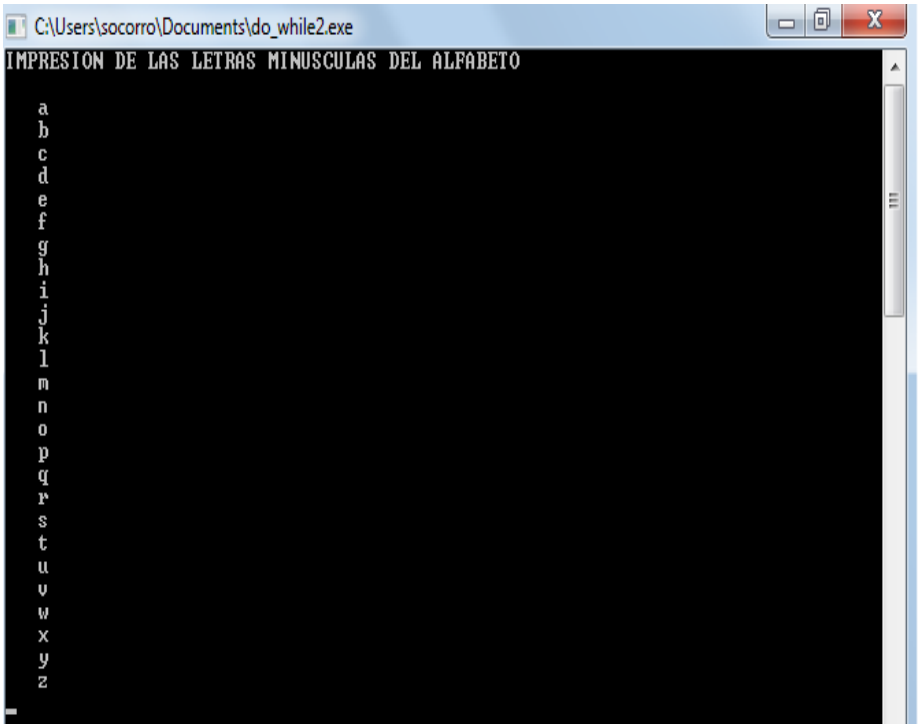


Figura 4.17. Estructura **do-while** controlada por un contador alfabético.

Ciclo controlado por un centinela.

Ejemplo: Calcular la sumatoria de un conjunto de números, mientras el número que se capture en cada ciclo sea diferente de 0. Al final imprimir el resultado respectivo.

```
#include <stdio.h>
#include <conio.h>

int num, acum = 0;

main()
{
```

```
    printf("SUMATORIA DE NÚMEROS ENTEROS\n\n");
    do{
        printf("Escribe un número entero: ");
        scanf("%d",&num);
        acum = acum + num;
      }while (num != 0);
      printf ("\nLa sumatoria es: %d", acum);
    getche();
}
```

La ejecución del programa se muestra en la figura 4.18, en este caso el centinela es el valor 0.

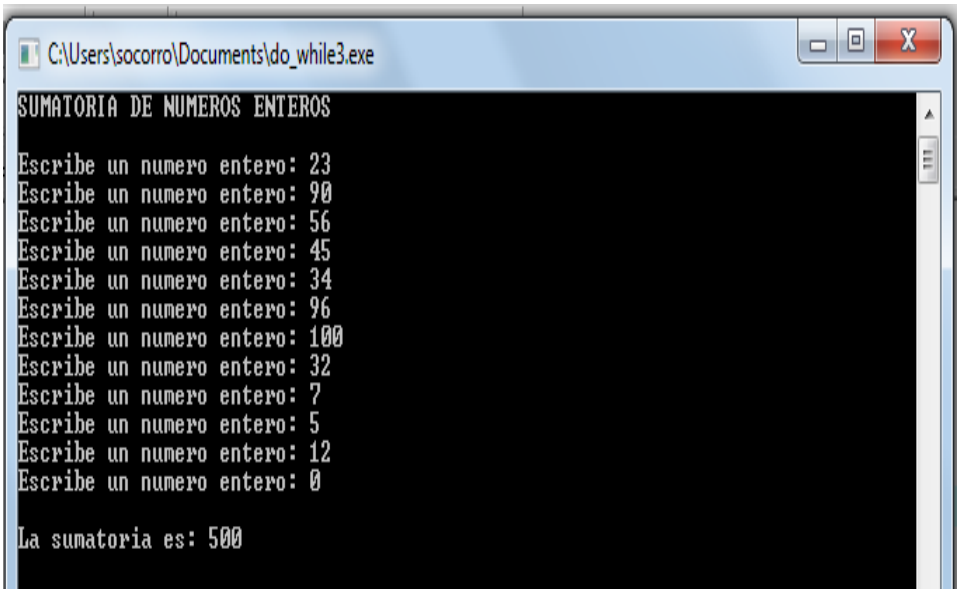


Figura 4.18. Programa controlado por un centinela.

Ciclo controlado por una tarea.

Ejemplo: Las potencias de 2 son 1, 2, 4, 8, 16, 32, 64, etc. Para obtener la siguiente potencia de 2, se multiplica la anterior por 2. Hacer un programa que imprima la primera potencia de 2 que es más de 1000.

```
#include <stdio.h>
#include <conio.h>

int pot=1;

main()
```

```
{
    printf("POTENCIAS DE 2\n\n");
    do{

        printf("%4d\n",pot);
        pot = pot * 2;


      }while (pot<= 1000);
      printf ("\nLa primera potencia de 2 que es más de 1000 es: %d", pot);
    getche();
}
```

La ejecución del programa se muestra en la figura 4.19.

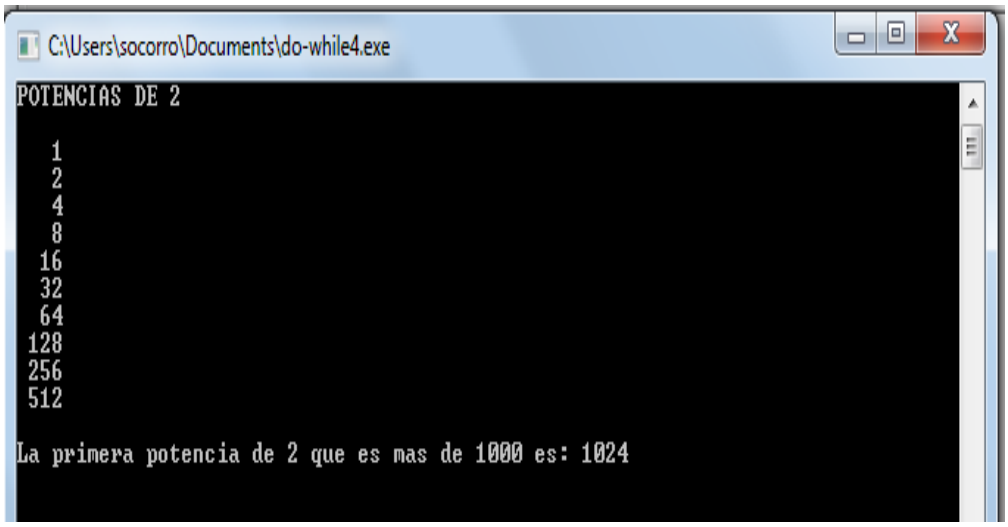


Figura 4.19. Ciclo controlado por una tarea.

4.7 Repetir desde.

El ciclo **for** del leguaje C es más potente que los ciclos **for** de otros lenguajes de programación clásicos como BASIC, Pascal o FORTRAN ya que ofrece más control sobre la inicialización y el incremento de las variables de control del ciclo. (JOYANES AGUILAR, 2005)

El ciclo for es la estructura repetitiva más utilizada y simple de manejar, ya que repite un conjunto de instrucciones un número determinado de veces. Una de sus aplicaciones principales son los arreglos, los cuales se analizarán en el capítulo 5.

El ciclo **for** es el más adecuado para implementar ciclos controlados por un contador. El ciclo **for** se diferencia del ciclo **while** en que las operaciones de control del ciclo se sitúan en un solo sitio: la cabecera del ciclo.
La sintaxis del ciclo **for** es:

for (Iniacilización; Condición; Incremento)
sentencias;

Inicialización.- Que inicializa la variable de control del ciclo.

Condición.- Que contiene una expresión relacional o lógica que determina cuándo finalizará el ciclo. La condición puede ser simple o compuesta.

Incremento.-Que incrementa o decrementa la variable o variables de control del ciclo.
Sentencias.- Acciones o sentencias que se ejecutarán por cada iteración del ciclo.
Las tres componentes (inicialización, condición e incremento) pueden tener una o varias instrucciones, las cuales deben ir separadas por comas. La condición nos lleva al valor final.

Cualquiera de los componentes en los paréntesis se puede omitir, incluso los tres, pero los separadores (punto y coma) deben aparecer siempre.

En el ciclo **for**, se ejecutará la inicialización, posteriormente se evaluará la condición y, en caso de ser verdadera, se ejecutará la instrucción o instrucciones que compone el ciclo. Después se realizará el incremento (o decremento) y se volverá a verificar la condición. Cuando la condición se vuelve falsa, en la siguiente evaluación se terminará el ciclo **for**. Si la condición nunca se vuelve falsa, la estructura nunca terminará y el ciclo se repetirá indefinidamente hasta que se detenga en forma manual.

Características del ciclo for

1. Se debe conocer por anticipado el valor de la variable inicial y final antes de entrar al ciclo.
2. La condición se evalúa antes del bloque de instrucciones. Si la condición es verdadera se ejecuta el bloque, y si es falsa se sale y pasa el control a la instrucción siguiente al bloque.
3. No se debe cambiar el valor de la(s) variable(s) de control, del valor inicial ni del valor final dentro del ciclo.
4. Se puede incrementar o decrementar la variable de control según se requiera.
5. El incremento o decremento de la variable de control es automático.
6. Sólo si existe más de una instrucción dentro del ciclo se necesitan las llaves ({-}).

7. Puede tener una o varias expresiones de inicialización, de condición y de incremento; estas expresiones se separan mediante puntos y comas.
8. Puede tener alguna o todas las expresiones vacías: for(; ;).
9. Si la condición está vacía, tenemos un ciclo infinito. (CORONA NAKAMURA & ANCONA VALDEZ, 2011)

Se realizará un programa sencillo para empezar y ver el uso del ciclo **for**. Se desea imprimir la palabra Hola 10 veces y el valor de la variable de control, el programa quedaría de la siguiente manera:

```
#include <stdio.h>
#include <conio.h>

int i;

main()
{
    for (i=1; i<=10; i++)
    {
       printf("HOLA! \n");
       printf("El valor de i es: %d\n",i);
    }
    getch();
}
```

La ejecución del programa se vería como lo muestra la figura 4. 20.

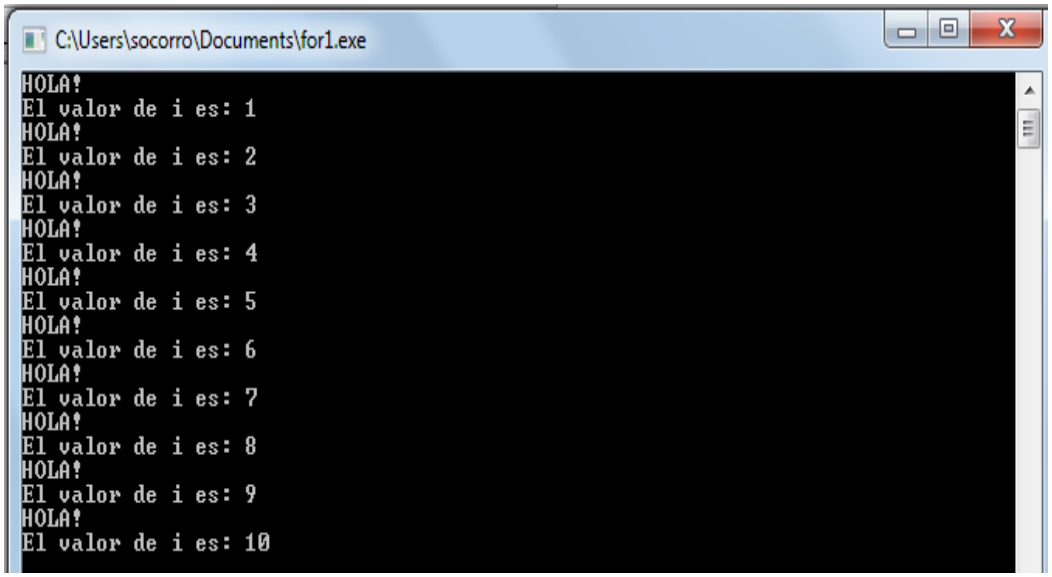


Figura 4.20. Programa con ciclo **for**.

Ejemplo: Elevar un número X a una potencia Y e imprimir el resultado.

```
#include <stdio.h>
#include <conio.h>

int cont, x, y, res=1;

main()
{
    printf("Escriba el numero que desea elevar a una potencia: ");
    scanf("%d",&x);
    printf("Escriba la potencia ",x);
    scanf("%d",&y);

    for (cont=1; cont <= y; cont++)
       res = res * x;
    printf("\nEl %d elevado a la potencia %d es: %d\n",x,y,res);
    getch();
}
```

En el código del programa podemos observar que la sentencia del ciclo **for** no necesita llaves ya que sólo es una instrucción. La ejecución del programa se muestra en la figura 4.21.

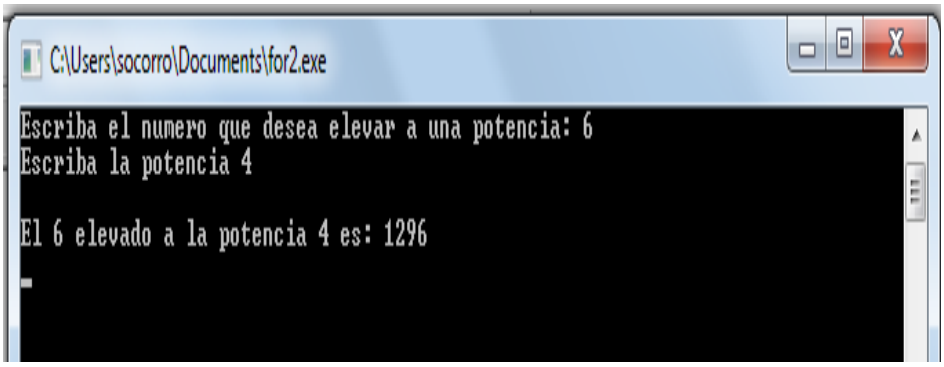


Figura 4.21. Ejecución de programa con ciclo **for**.

En el siguiente programa se utilizará la variable de control en decremento y se utilizará una variable entera larga ya que tiene un rango mayor que la entera normal. Calcular el factorial de un número entero e imprimir el resultado.

```
#include <stdio.h>
#include <conio.h>

int i, x;
```

```
long int fact = 1;

main()
{
    printf("Escribe el número para calcular su factorial: ");
    scanf("%d",&x);
    for (i = x; i > 1; i--)
    {
       fact = fact * i;
       printf ("\n%ld",fact);
       }
    printf ("\n El factoria de %d es %ld", x, fact);
    getche();
}
```

En la figura 4.22 se muestra la ejecución del programa.

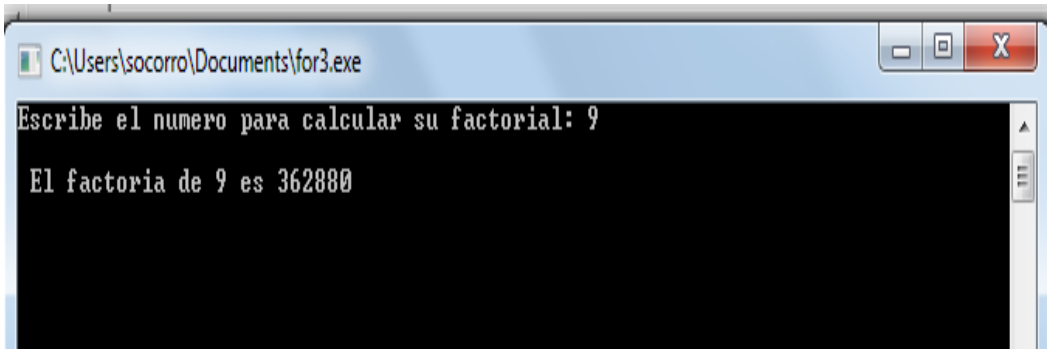


Figura 4.22. Programa con ciclo for descendente.

4.8 Elaboración de programas.

1. Hacer un programa en lenguaje C que, al recibir como dato el precio de un producto importado, incremente el 12% el mismo si es inferior a $1,500.00 y que además escriba el nuevo precio del producto.
2. Hacer un programa en lenguaje C que, al recibir como dato dos números enteros, determine si el primero es divisible por el segundo.
3. Hacer un programa en lenguaje C que ordene dos números con un solo if.
4. Hacer un programa en lenguaje C que, calcule el promedio de un alumno que cursó tres materias. Si su promedio es mayor que 95 se le va asignar una calificación de 100. Imprimir cuál fue la calificación promedio del alumno.
5. Hacer un programa en lenguaje C que, lea el año y calcule e imprima si es bisiesto o no.
6. Hacer un programa en lenguaje C, que lea 4 números. Imprima el mayor, el menor y la suma de ambos.

7. Hacer un programa en lenguaje C, que lea 3 números. Ordene e imprima de manera ascendente.
8. Hacer un programa en lenguaje C, que lea 3 números. Ordene e imprima de manera descendente.
9. Hacer un programa en lenguaje C que, al recibir como dato el número de días transcurridos del año, imprima a que mes pertenece. Por ejemplo si se captura 32 se imprimirá el mes de febrero.
10. Hacer un programa en lenguaje C que, al recibir como datos la fecha de nacimiento (día, mes y año) de una persona y la fecha actual (día, mes y año), calcule e imprima la edad al día de hoy. Por ejemplo usted tiene 19 años 7 meses y 14 días.
11. Hacer un programa en lenguaje C que permita calcular la cantidad a pagar a un trabajador tomando en cuenta su salario y las horas extra trabajadas. Las horas extra se calculan en función de la siguiente información:

Categoría del trabajador	Horas extra
1	$40.00
2	$50.00
3	$85.00

Cada trabajador puede tener como máximo 30 horas extra, si tiene más, sólo se les pagarán las primeras 30. Los trabajadores con categoría 4 o mayor no pueden recibir este beneficio.

12. Hacer un programa en lenguaje C que, al recibir como dato los tres lados de un triángulo, imprimir si éstos pertenecen al mismo triángulo.
13. Hacer un programa en lenguaje C que, al recibir como dato la edad de una persona, imprima si la persona es un niño (0 a 12 años), adolescente (13 a 17 años), adulto (18 a 65 años) o tercera edad (mayor que 65 años).
14. Hacer un programa en lenguaje C que, al recibir como dato un número entero imprima si es
    a) Cero, positivo o negativo.
    b) Par o impar (cuando sea positivo).
    c) Múltiplo de 7 o no múltiplo de 7 (cuando sea impar).

    Por ejemplo para el número 35, imprimirá: Es un número positivo, es impar y es múltiplo de 7.
15. Hacer un programa en lenguaje C que, calcule el área de un triángulo, un círculo y de un cuadrado. Utilizado switch.
16. Hacer un programa en lenguaje C, que contenga un menú que realice tres conversiones de tiempo.
17. Hacer un programa en lenguaje C que permita generar la tabla de multiplicar de un número entero positivo N, comenzando desde 1.
18. Hacer un programa en lenguaje C que, al recibir como dato un número entero N, calcule e imprima el resultado de la siguiente serie:

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{\mathrm{N}}$$

19. Hacer un programa en lenguaje C que, al recibir como datos N números naturales, determine cuántos de ellos son positivos, negativos o nulos.
20. Escriba un programa en lenguaje C que, al recibir como dato un número entero positivo, escriba una figura como la que se muestra a continuación (ejemplo para N= 5):
    1
    1 2
    1 2 3
    1 2 3 4
    1 2 3 4 5
    1 2 3 4
    1 2 3
    1 2
    1
21. Escribir un ciclo while que visualice todas las potencias de un entero n, menores que un valor especificado max_limite.
22. ¿Qué hace el siguiente ciclo while?. Reescribirlo con sentencias for y do-while.

```
num =10;
while (num <=100)
{
        printf("%d" \n",num);
        num += 10;
}
```

23. Escribir un programa que calcule y visualice el más grande, el más pequeño y la media de N números. El valor N se solicitará al principio del programa y los números serán introducidos por el usuario.
24. Escribir un programa que determine todos los años que son bisiestos en el siglo XXI.
25. Escribir un programa que invierta los dígitos de un entero positivo dado.
26. Considere el siguiente código de programa:

```
for (i = 0; i < n; ++i)
{
                --n;
}
printf("%d ,\n",i);
```

    a) Cuál es la salida si n es 0?
    b) Cuál es la salida si n es 1?
    c) Cuál es la salida si n es 3?

27. Escriba un programa en lenguaje C que, calcule el máximo común divisor (MCD) de dos números naturales N1 y N2. El MCD entre dos números es el natural más grande que divide a ambos.
28. Escriba un programa en lenguaje C que, imprima en forma inversa el adecedario es decir de la z a la a y luego vay eliminando de una letra en una empezando por la z hasta que quede la a.
29. Escriba un programa en lenguaje C que, lea 10 números e imprima la suma de los que sean primos.
30. Escriba un programa en lenguaje C que, calcule el promedio de los cuadrados de los números comprendidos entre 1 y un número N dado. Es decir:

$$promedio = (\ 1^2 + 2^2 + \cdots + n^2)/n$$

5. ARREGLOS Y ARCHIVOS.

5.1 Arreglos unidimensionales.

Un arreglo es una colección finita, homogénea y ordenada de elementos. **Finita**, porque todo arreglo tiene un límite, es decir, se debe determinar cuál es el número máximo de elementos del arreglo. **Homogénea**, porque todos los elementos del arreglo deben ser del mismo tipo. **Ordenada**, porque se puede determinar cuál es el primer elemento, cuál es el segundo, y así sucesivamente. (CAIRÓ BATTISTUTTI, 2006)

Un arreglo es un tipo de dato estructurado que almacena en una sola variable un conjunto limitado de datos o elementos del mismo tipo. Asimismo, es un conjunto de localidades de memoria contiguas donde la dirección más baja corresponde al primer elemento y la dirección más alta al último. Por sí mismo, el nombre del arreglo apunta a la dirección del primer elemento del arreglo. Los datos del arreglo se llaman elementos del arreglo y su posición se numera consecutivamente: 1, 2, 3…n. (CORONA NAKAMURA & ANCONA VALDEZ, 2011)

Un arreglo unidimensional es una colección finita, homogénea y ordenada de datos, en la que se hace referencia a cada elemento del arreglo por medio de un índice. Este último indica la casilla en la que se encuentra el elemento. (CAIRÓ BATTISTUTTI, 2006)

Un arreglo unidimensional también es conocido como vector o lista. Un arreglo en lenguaje C inicia en la posición cero, por lo tanto si el arreglo se llama x, y tiene 6 elementos, sus nombres son x[0], x[1], x[2], x[3], x[4], x[5]. Esto lo puede ver gráficamente en la figura 5.1.

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]
4.2	12.0	3.45	4.32	0.31	51.9

Figura 5.1. Arreglo unidimensional.

El arreglo tiene 6 elementos x[0] contiene 4.2, x[1] contiene 12.0, x[2] contiene 3.45, x[3] contiene4.32, x[4] contiene 0.31 y x[5] contiene 51.9. La figura 5.1 representa realmente una región de la memoria de la computadora, ya que un arreglo se almacena siempre con sus elementos en una secuencia de posiciones de memoria contigua.

Declaración de un arreglo unidimensional.

Al igual que cualquier tipo de variable, los arreglos deben declararse antes de ser utilizados. La sintaxis para declarar un arreglo es la siguiente:

```
tipo_dato  nombre_del_arreglo [número_de_elementos];
```

Ejemplos:

```
int a [10];       /*Declaración de un arreglo de tipo entero de 10 elementos*/
float c [15];     /*Declaración de un arreglo de tipo real de 15 elementos*/
```

Si se va asignar un valor desde teclado, se utiliza la instrucción *scanf* de forma similar a cualquier variable.

```
scanf ( “%d”, &lista [i] );
```

Para imprimir el contenido de una celda, se procede de manera similar a la de cualquier variable, teniendo cuidado de indicar el índice:

```
printf ( “%d”, arreglo [0] );
```

Si se manipularan todos los elementos de un arreglo, es frecuente utilizar una estructura repetitiva; la más usual es el ciclo *for*, ya que los índices del arreglo aumentan de uno en uno. Por ejemplo, cuando desea imprimir el contenido del arreglo o cuando se suman todos los elementos, o también cuando se va a inicializar, como en el ejemplo siguiente:

```
for ( i=0; i<5; i++ )
        ListaCalif [i] = 0;
```

En este caso se inicializó el arreglo con valores de 0; también es posible inicializar con valores distintos desde la declaración, como:

```
int ListaCalif [5] = {10, 5, 8, 9, 4};
```

o

```
int ListaCalif [ ] = {10, 5, 8, 9, 4};
```

Si se omite el tamaño del arreglo éste se definirá por el número de valores de inicialización; inclusive se pueden definir los elementos del arreglo ya declarado.

Para imprimir todos los elementos de un arreglo usando el ciclo *for*:

```
for ( i=0; i<5; i++ )
        printf ( “El valor de la casilla %d es %d:”, i, arreglo [i] );
```

En caso de que el índice *i* esté fuera del límite, el compilador de C no marcará error, pero puede causar un fallo en el programa.

Ejemplos de arreglos unidimensionales.

Programa 1.- Leer y almacenar siete estaturas en un arreglo unidimensional y mostrarlas en forma tabular.

```
#include <stdio.h>
#include <conio.h>
#define TAMANIO 7

float estatura [TAMANIO];
int i;

main()
{
    for ( i=0; i<=6; i++ )
    {
        printf ( "Dame la estatura %d: ", i+1 );
        scanf ( "%f", &estatura [i] );
    }
    printf ( " \n   NUM        ESTATURA  \n" );
    for ( i=0; i< TAMANIO; i++ )
        printf ( "   %d         %f  \n", i+1,estatura [i] );
    getche ();
}
```

En este ejemplo se utiliza una directiva de preprocesador, que es una variable simbólica llamada *TAMANIO* de valor 7, se define un arreglo de siete elementos de número reales y el índice *i* de tipo entero. En la función *main* se utiliza un ciclo *for* para solicitar la estatura *i +1*: se escribió el índice así para imprimir el texto "*Dame la estatura 1:* ", y se almacena en la posición *estatura[0]*. Todo esto se repite hasta que el valor de *i* llega a 6. Una vez almacenadas las estaturas se imprimen, con otro ciclo *for*, en forma de lista. (MÁRQUEZ FRAUSTO, OSORIO ÁNGEL, & OLVERA PÉREZ, 2011)

En la figura 5.2 se puede observar una posible ejecución del programa.

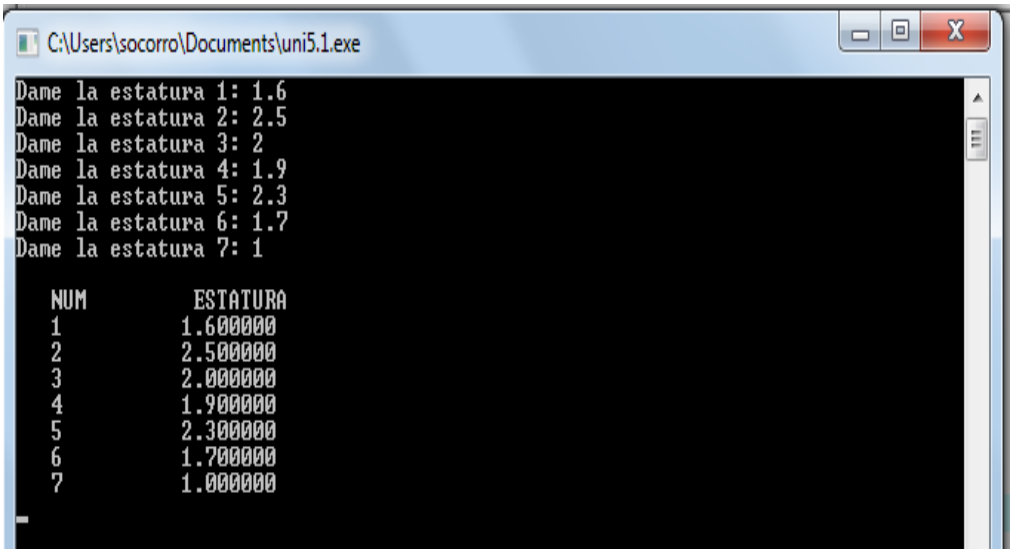


Figura 5.2. Programa que almacena estaturas.

Programa 2.- Imprimir el contenido de las posiciones 0, 3 y 5 del arreglo vec, inicializando el arreglo con los valores 3, 5, 6, 8, 4, 9, 7.

Lo primero es inicializar el arreglo y después imprimir los valores de las posiciones indicadas.

```
#include <stdio.h>
#include <conio.h>


int vec [7] = {3, 5, 6, 8, 4, 9, 7};

main()
{
    printf ( "\n posicion 0: %d ",vec [0] );
    printf ( "\n posicion 3: %d ",vec [3] );
    printf ( "\n posicion 5: %d ",vec [5] );
    getche ();
}
```

En este ejemplo como los valores están asignados al inicio, sólo hay que mostrar el contenido de cada una de las posiciones indicadas.

La ejecución del programa se muestra en la figura 5.3.

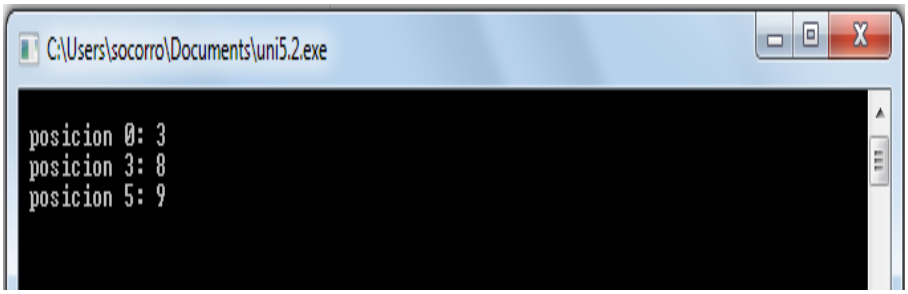


Figura 5.3. Programa que imprime posiciones.

Programa 3.- Buscar un número en un arreglo e indicar en qué posición se encuentra.

Para este programa es necesario pedir el número a buscar, realizar la búsqueda en todo el arreglo y por último imprimir si el número se encontró y la posición.

```
#include <stdio.h>
#include <conio.h>

int vector [5] = {3, 5, 7, 9, 2}, i, n, b=0, p;

main()
{
     printf ( "Teclea el valor a buscar: " );
     scanf ( "%i", &n );
     for ( i=0; i<5; i++ )
        if ( vector [i] == n )
          {
            b = 1;
            p = i;
          }
        if ( b == 1 )
          printf ( "\nEl numero %i si se encuentra en la posicion %i", n, p );
        else
          printf ( "\nEl numero %i no se encuentra", n );
     getche ();
}
```

Primero se solicita un número para buscarlo en el arreglo, éste se almacena en la variable *n*. Se compara si el primer elemento del vector es *n*; si no es así, se revisan los siguientes elementos. Cuando la condición es verdadera, *b* toma el valor de 1, *p* toma el valor de la posición. Al salir del ciclo se revisa si *b* contiene 1; si es cierto, se imprime que sí se encontró y en qué posición. En caso contario *b* conserva el valor de 0 y *p* no toma ningún valor.

La ejecución del programa se puede observar en la figura 5.4

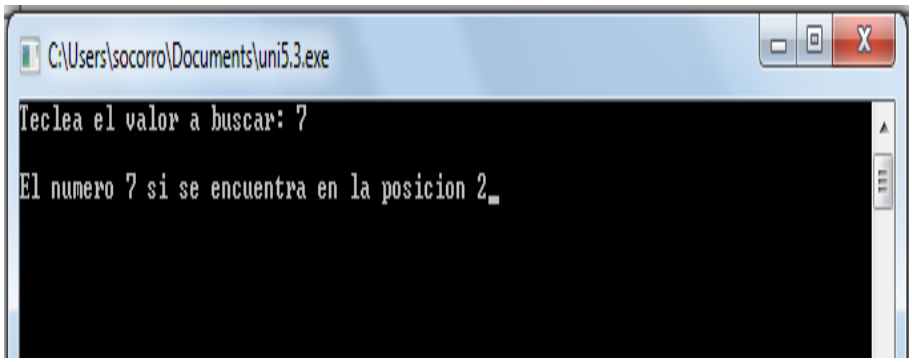


Figura 5.4. Busqueda de datos.

5.2 Arreglos bidimensionales.

Los arreglos bidimensionales son también llamados tablas o matrices. Cada posición de un arreglo bidimensional tiene dos índices: el primero indica el número de renglón y el segundo el número de columna en que se encuentra el elemento. (MÁRQUEZ FRAUSTO, OSORIO ÁNGEL, & OLVERA PÉREZ, 2011)

La forma convencional de declarar un arreglo bidimensional es:

tipo_dato nombre_del_ arreglo [número_de_renglones] [número_de_columnas];

Ejemplo:

int matriz [3] [2];

En un arreglo de números enteros de tres renglones y dos columnas (3 X 2) se podrían ver así los índices:

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

Cuando se hace referencia a la matriz completa, como durante la asignación o la impresión de valores, en general se hace de manera similar como en el arreglo unidimensional, cuidando solamente que se haga referencia a los índices de renglón y columna. La inicialización se realiza de la siguiente manera:

int matriz [3] [2] = {1, 2, 3, 4, 5, 6}; o
int matriz [3] [2] = {

```
        {1, 2}
        {3, 4}
        {5, 6}
};
```

El segundo caso muestra que en realidad un arreglo bidimensional es un arreglo de vectores. Para almacenar un valor desde el teclado:

```
scanf ( “%d”, &matriz [0] [0] );
```

A la acción de manipular todos los elementos de una matriz se le puede llamar recorrido, y éste se lleva a cabo utilizando dos ciclos *for* anidados, ya que es necesario recorrer las columnas para cada una de las filas. (MÁRQUEZ FRAUSTO, OSORIO ÁNGEL, & OLVERA PÉREZ, 2011)

Por ejemplo, para asignar a todos los elementos de la matriz un valor desde el teclado:

```
for ( i = 0; i <= 3; i++ )
for ( j = 0; j <= 2; j++ )
        scanf (“%d”, &matriz [i] [j] );
```

Para cada valor de *i* (que en este caso corresponde a los renglones) se recorrería todos los valores de *j* (que son las columnas), ya que cada renglón 3 columnas en la matriz de ejemplo.

Lo más importante de una matriz son los índices, porque éstos determinan el orden en el que se debe ejecutar el ciclo *for*. Para imprimir un valor:

```
printf ( “%d”, matriz [i] [j] );
```

Para imprimir todos los valores del arreglo en el mismo orden en que se leyeron los datos, según el ejemplo anterior:

```
for ( i = 0; i <= 3; i++ )
for ( j = 0; j <= 2; j++ )
        printf ( “%d”, &matriz [i] [j] );
```

Al invertir los índices se crea el mismo efecto de recorrer la *i*, que son las columnas, más rápidamente que los renglones con *j*.

```
for ( j = 0; j <= 3; j++ )
for ( i = 0; i <= 2; i++ )
        printf ( “%d\t”, &matriz [j] [i] );
```

Para recorrer por columna podría ser así:

```
for ( i = 0; i <= 2; i++ )
for ( j = 0; j <= 3; j++ )
        printf ( “%d”, &matriz [j] [i] );          o        printf ( “%d\n”, &matriz [i] [j] );
```

Para imprimir todos los valores fila por fila:

```
for ( i = 0; i <= 3; i++ )
for ( j = 0; j <= 2; j++ )
        printf ( “%d\t”, &matriz [i] [j] );
```

Y para imprimir los valores columna por columna puede ser así:

```
for ( i = 0; i <= 3; i++ )
for ( j = 0; j <= 2; j++ )
        printf ( “%d”, &matriz [j] [i] );          o también

for ( j = 0; j <= 3; j++ )
for ( i = 0; i <= 2; i++ )
        printf ( “%d”, &matriz [i] [j] );
```

Ejemplos de arreglos bidimensionales.

Programa 1.- Almacenar números entre 1 y 30, generados aleatoriamente, en una tabla de 3 X 2 renglones y columnas.

Para hacer este programa es necesario incluir la función que genera números aleatorios, almacenar los números generados y mostrar la matriz.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>

int tabla [3] [2], i, j;

main()
{
    srand (time(NULL));
```

```
    for ( i = 0; i < 3; i++)
        for ( j = 0; j < 2; j++)
            tabla [i] [j] = rand () % 30 + 1;
    for ( i = 0; i < 3; i++)
    {
        for ( j = 0; j < 2; j++)
            printf ( "\t%d", tabla [i] [j] );
        printf ( "\n" );
    }
    getche ();
}
```

Se declara la matriz indicando el número de renglones y columnas; a continuación con los dos ciclos se recorren las posiciones de la misma almacenando un número aleatorio. La función *rand()* genera números aleatorios entre 0 y 29 y se le suma uno para que sean entre 1 y 30.

Si ejecutamos varias veces nuestro programa, la secuencia de números aleatorios se repite. Para evitar este problema usamos la función *srand()*, a la que se le pasa de parámetro un número que se utilizará como número inicial. En este ejemplo utilizamos el reloj de la computadora con la función *time()*. Sólo debemos llamarla una vez en nuestro programa.

La ejecución de este programa la puede observar en la figura 5.5.

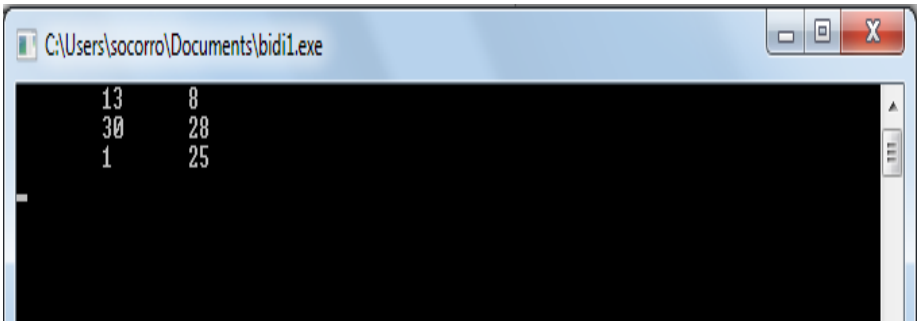


Figura 5.5. Uso de función rand().

Programa 2.- Realizar la suma de dos matrices, con una dimensión de hasta 4 X 4.

Primero se debe declarar el tamaño de las matrices verificando que sean de la misma dimensión, llenar primero una matriz y después la otra, después sumar las dos matrices y guardar el resultado en otra matriz, por último mostrar el resultado de la suma.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

#define col 4
#define ren 4

int mat1 [ren] [col], mat2 [ren] [col], mat3 [ren] [col];
int i, j, col1, ren1, col2, ren2;
int band1;

main()
{
     do
     {
        printf ( "\n\t Matriz 1, dame el numero de renglones y columnas " );
        scanf ( "%d %d", &ren1, &col1 );
        printf ( "\n\t Matriz 2, dame el numero de renglones y columnas " );
        scanf ( "%d %d", &ren2, &col2 );
        if ( ren1 == ren2 && col1 == col2 )
            band1 = 0;
        else
        {
            band1 = 1;
            printf ( "\n\t\t\tNUMERO DE DATOS NO VALIDOS\n" );
        }
     }while ( band1);
     system ("CLS");
     printf ( " Matriz 1 \n" );
     for ( i = 0; i < ren1; i++)
     {
        for ( j = 0; j < col1; j++)
        {

            scanf ( "%d", &mat1 [i] [j]);
            printf ("\t");
        }
        printf ("\n");
     }
     printf ( " Matriz 2 \n" );
     for ( i = 0; i < ren2; i++)
     {
        for ( j = 0; j < col2; j++)
        {
```

```
            printf ("  ");
            scanf ( "%d", &mat2 [i] [j]);
        }
        printf ("\n");
    }
    for ( i = 0; i < ren1; i++)
        for ( j = 0; j < col1; j++)
            mat3 [i] [j] = mat1 [i] [j] + mat2 [i] [j];

    printf ( " RESULTADO \n" );
    for ( i = 0; i < ren1; i++)
    {
        for ( j = 0; j < col1; j++)
        {
            printf ("  ");
            printf ( "%d", mat3 [i] [j]);
        }
        printf ("\n");
    }
    getch ();
}
```

Se solicita el número de renglones y columnas de la primera matriz; después se solicita el número de reglones y columnas para la segunda matriz, se verifica que la dimensión sea igual, de lo contrario pregunta de nuevo las dimensiones como se puede observar en la figura 5.6. Ya que es válida la dimensión se solicitan los datos; para cada una de las matrices.

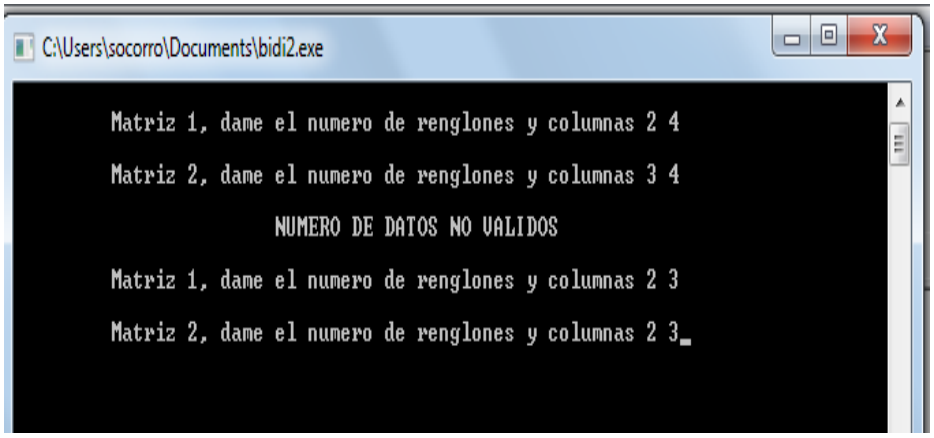


Figura 5.6. Dimensiones no válidas.

A continuación realiza la suma y muestra la matriz resultante conservando la dimensión. La ejecución del programa lo puede observar en la figura 5.7.

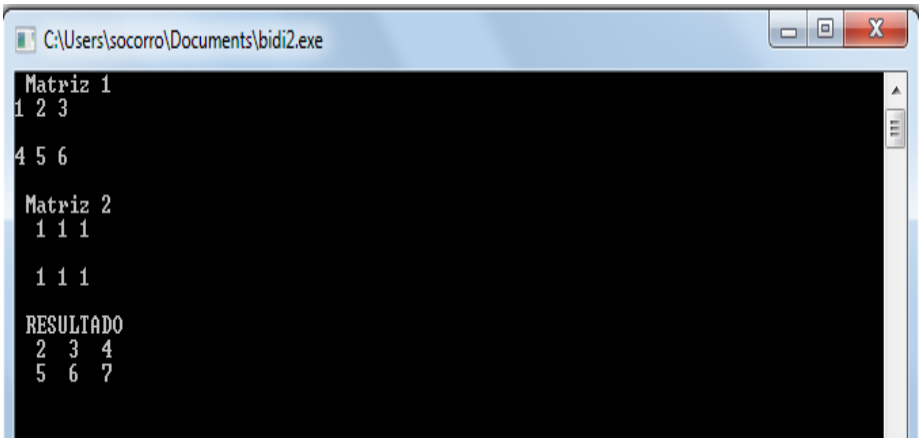


Figura 5.7. Suma de matrices.

Programa 3.- Encontrar el elemento mayor y el menor en una matriz de tamaño 10 X 10 máximo.

Primero se debe declarar una matriz bidimensional, solicitar el número de filas y columnas, llenar la matriz con datos y encontrar el mayor y el menor elemento de la matriz.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

#define MAX 10

int mat [MAX] [MAX], i, j, c, f, may, men;
main()
{
    printf ( "Dame el total de filas " );
    scanf ( "%d", &f );
    printf ( "Dame el total de columnas " );
    scanf ( "%d", &c );
    for ( i = 0; i < f; i++)
       for ( j = 0; j < c; j++)
       {
           printf (" a(%d,%d)= ",i,j);
           scanf ( "%d", &mat[i][j]);
       }
    may = mat[0][0];
    men = mat[0][0];
    for ( i = 0; i < f; i++)
       for ( j = 0; j < c; j++)
       {
```

```
            if( mat[i][j] > may)
               may = mat[i][j];
            else if (mat[i][j] < men)
                men = mat[i][j];
         }
      printf(" El numero mayor es %d", may);
      printf(" El numero menor es %d", men);
      getche();
}
```

En este ejercicio se declara una matriz de *f* (filas) y *c* (columnas); posteriormente se llena utilizando dos ciclos *for* anidados, la variable *i* sirve como subíndice para las filas y la variable *j* para las columnas. A las variables *may* y *men* se les asigna el valor del elemento de la matriz que se encuentra en la primera posición, es decir *mat[0][0]*, y otra vez se utilizan dos ciclos *for* anidados, ahora para ir comparando cada elemento de la matriz con el elemento mayor almacenado en *may* y con el elemento menor almacenado en *men*. Para encontrar el mayor y el menor se utiliza una estructura de control selectiva doble anidada, para ir revisando si el elemento es mayor: *if (mat[i][j] > may)* entonces se almacena este nuevo valor en la variable *may*. Si fuera falso se revisa si es menor con la condición *if(mat[i][j] < men)*; si se cumple se almacena este nuevo valor en la variable *men*.

La ejecución del programa se puede observar en la figura 5.8.

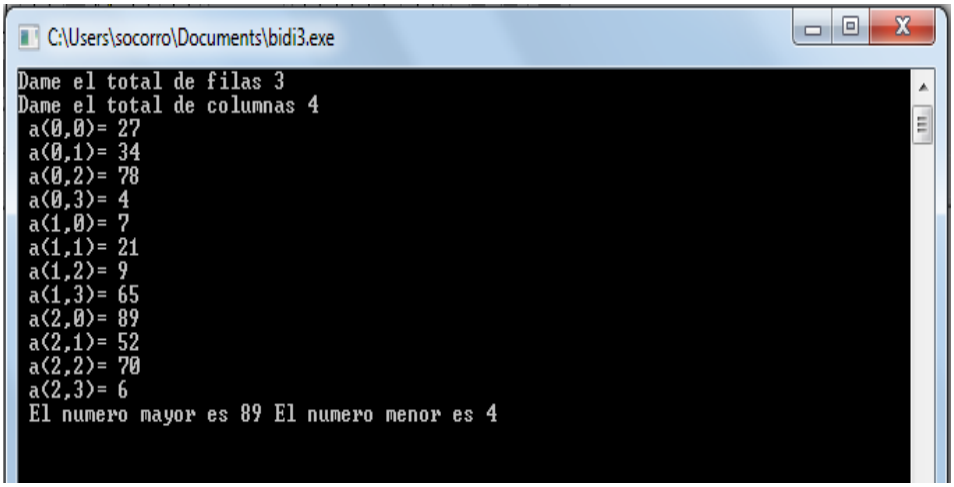


Figura 5.8. Encuentra elemento mayor y menor en una matriz.

5.3 Archivos.

El almacenamiento de los datos dentro de variables y arreglos es temporal; todos esos datos se pierden cuando termina el programa. Los **archivos** se utilizan para retener permanentemente grandes cantidades de datos. Las computadoras almacenan los archivos en dispositivos secundarios de almacenamiento, en especial en dispositivos de disco. (DEITEL & DEITEL, 2004)

Los campos están compuestos por caracteres. Un **campo** es un grupo de caracteres que en forma conjunta representa un mismo significado. Por ejemplo, un campo que consta sólo de letras mayúsculas y minúsculas puede utilizarse para representar el nombre de una persona.

Un **registro** (es decir una estructura en C) está compuesto por varios campos. Por ejemplo, en un sistema de nómina, el registro de un empleado en particular podría estar compuesto por los siguientes campos:

1. Número de seguro social.
2. Nombre.
3. Dirección.
4. Sueldo por hora.
5. Deducciones.
6. Precepciones de un año a la fecha.
7. Monto por concepto de impuestos.

Así, un **registro** es un grupo de campos relacionados. En el ejemplo anterior, cada uno de los campos pertenece al mismo empleado. Por supuesto, una empresa en particular puede tener muchos empleados y tener un registro de nómina por cada empleado. Un **archivo** es un grupo de registros relacionados.

Archivos de texto y método de acceso secuencial.

En los **archivos de texto** los datos se almacenan en formato de texto y ocupan posiciones consecutivas en el dispositivo de almacenamiento secundario. La única forma de acceder a los componentes de un archivo de texto es hacerlo en forma **secuencial**. Es decir, accediendo al primer componente, luego al segundo, y así sucesivamente hasta llegar al último, y por consiguiente al fin del archivo. Un elemento importante cuando se trabaja con archivos de texto es el *área del búfer*, que es el lugar donde los datos se almacenan temporalmente mientras se transfieren de la memoria al dispositivo secundario en que se encuentran o viceversa. (CAIRÓ BATTISTUTTI, 2006)

En el siguiente programa se observará como se abre y cierra un archivo de texto, y la manera en que se almacenan los caracteres. Se utilizarán las funciones **getc** y **fgetc** para lectura de caracteres, así como **putc** y **fputc** para escritura de caracteres.

```
#include <stdio.h>

main()
{
    char p;
    FILE *arch;
    arch = fopen("archivo.txt", "w");
    if (arch != NULL)
    {
       while((p=getchar()) != '\n')
          fputc(p,arch);
       fclose(arch);
    }
    else
       printf ("No se puede abrir el archivo");
}
```

La instrucción **FILE *arch**; indica que **arch** es un apuntador al inicio de la estructura FILE, área del búfer que siempre se escribe con mayúscula.

La instrucción **arch = fopen("archivo.txt", "w")**; permite abrir un archivo llamado **"archivo.txt"** que puede ser una variable de tipo cadena de caracteres, o bien, una constante sin extensión o con extensión txt para realizar escritura en el archivo(**"w"**).

La instrucción **if (arch != NULL)** permite evaluar el contenido del apuntador **arch**. Si éste es igual a **NULL**, implica que el archivo no se pudo abrir, en cuyo caso es conveniente escribir un mensaje para notificar esta situación. Por otra parte, si el contenido del apuntador es distinto de **NULL** entonces se comienza a trabajar sobre el archivo.

Por último la instrucción **fclose(arch)** se utiliza para cerrar el archivo.

Se puede observar la ejecución del programa en la figura 5.9.

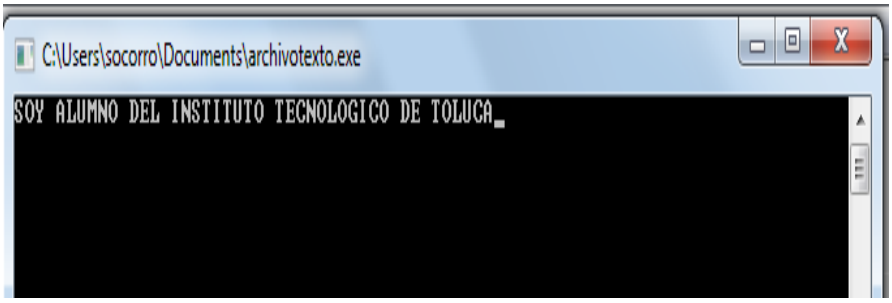


Figura 5.9. Escritura de un archivo de texto.

Los archivos pueden abrirse de diferentes maneras según las necesidades. En la tabla 5.1 puede ver todas las formas de abrir un archivo.

Tabla 5.1. Modo para abrir un archivo.

Modo	**Descripción**
r	Abre un archivo para lectura.
w	Crea un archivo para escritura. Si el archivo ya existe, descarta el contenido actual.
a	Agrega; abre o crea un archivo para escritura al final del archivo.
r+	Abre un archivo para actualización (lectura y escritura).
w+	Crea un archivo para actualización. Si el archivo ya existe, descarta el contenido actual.
a+	Agrega; abre o crea un archivo para actualización; la escritura se hace al final del archivo.

En el siguiente programa podemos observar la forma como se leen caracteres de un archivo. Se introduce **feof()**, que se utiliza para verificar el fin del archivo.

```
#include <stdio.h>
#include <conio.h>

main()
{
    char p;
    FILE *arch;
    if ((arch = fopen("archivo.txt", "r")) != NULL)
    {
       while(!feof(arch))
       {
          p = fgetc(arch);
          putchar(p);
       }
       fclose(arch);
    }
    else
       printf ("No se puede abrir el archivo");
    getch();
}
```

Observe que en la instrucción **if ((arch = fopen("archivo.txt", "r")) != NULL)** se abre el archivo para lectura y al mismo tiempo se verifica si se pudo abrir, quedando agrupadas en una sola instrucción.

En el ciclo **while** se leen caracteres del archivo mientras no se detecta el fin del archivo. En la instrucción **p = fgetc(arch)** lee el carácter del archivo. La instrucción **putchar(p)** despliega el carácter en la pantalla como lo muestra la figura 5.10.

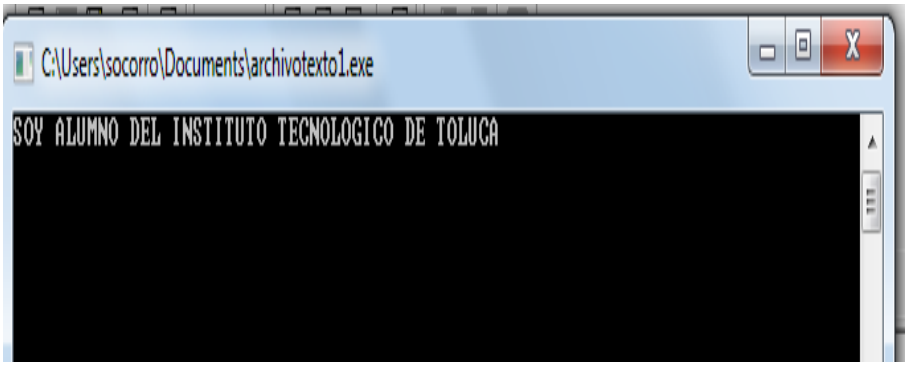


Figura 5.10. Lectura de caracteres de un archivo.

En el siguiente programa se observará la forma como se manejan las cadenas de caracteres en un archivo. Se introducen las funciones **fgets** y **fputs** para lectura y escritura de cadena de caracteres, respectivamente.

```
#include <stdio.h>

main()
{
    char cad[50];
    int res;
    FILE *ar;
    if ((ar = fopen("arc.txt", "w")) != NULL)
      {
        printf("Deseas ingresar una cadena de caracteres? si-1 no-0:");
        scanf("%d",&res);
        while (res)
        {
            fflush(stdin);
            printf("Ingrese la cadena: ");
            gets(cad);
            fputs(cad, ar);
            printf("Deseas ingresar otra cadena de caracteres? si-1 no-0:");
            scanf("%d",&res);
            if (res)
              fputs("\n",ar);
        }
        fclose(ar);
      }
      else
        printf("No se puede abrir el archivo");
}
```

Observe como la instrucción **fputs(cad, ar)** es utilizada para escribir la cadena en el archivo.

Por otro lado, con la instrucción **fputs("\n",ar)** se indica un salto de línea, excepto en la última cadena. Si no se hiciera esta indicación, la función **fputs** pegaría las cadenas y se tendría dificultad para leerlas. La ejecución del programa la puede observar en la figura 5.11.

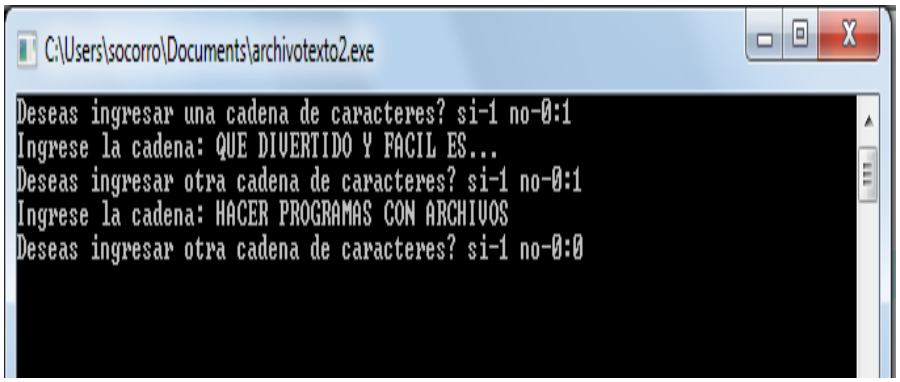


Figura 5.11. Archivo con cadena de caracteres.

En el siguiente programa se puede observar la forma como se leen las cadenas de caracteres de un archivo.

```
#include <stdio.h>
#include <conio.h>

main()
{
    char cad[50];
    FILE *ap;
    if ((ap = fopen("arc.txt", "r")) != NULL)
      {
         while (!feof(ap))
         {
              fgets(cad, 50, ap);
              puts(cad);
         }
         fclose(ap);
      }
      else
         printf("No se puede abrir el archivo");
      getch();
}
```

Observe que la instrucción para leer cadenas requiere de tres argumentos.(**fgets(cad, 50, ap)**). La instrucción **puts(cad)** despliega la cadena en pantalla.

En la figura 5.12 se puede apreciar la ejecución del programa para leer una cadena de caracteres.

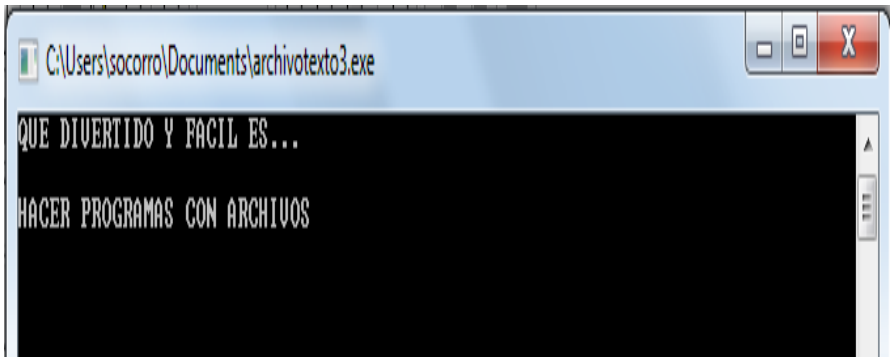


Figura 5.12. Archivo y cadena de caracteres.

En el siguiente programa se crea un archivo de acceso secuencial que puede utilizarse para un sistema de cuentas por cobrar y ayudar a mantener el registro de los montos de las deudas crediticias de sus clientes.

Para cada cliente, el programa solicita un número de cuenta, el nombre del cliente y su saldo (monto que el cliente debe a la empresa). Los datos de cada cliente constituyen un registro del cliente. En este programa el número de cuenta se utiliza como la clave del registro; el archivo se creará y se actualizará de acuerdo con el orden de los números de cuenta. El usuario deberá introducir los registros de acuerdo con el orden de los números de cuenta.

```
#include <stdio.h>

int main()
{
   int cuenta;
   char nombre[30];
   double saldo;
   FILE *arcliente;
   if ((arcliente = fopen("clientes.dat", "w")) == NULL)
       printf("El archivo no pudo abrirse\n");
   else
       {
           printf("introduzca la cuenta, el nombre y el saldo.\n");
           printf("introduzca EOF al final de la entrada.\n");
           printf("? ");
```

```
        scanf("%d%s%lf", &cuenta, nombre, &saldo);
        while(!feof(stdin))
        {
            fprintf(arcliente, "%d %s %.2f\n", cuenta, nombre, saldo);
            printf( "? ");
            scanf("%d%s%lf", &cuenta, nombre, &saldo);
        }
        fclose(arcliente);
    }
    return 0;
}
```

La instrucción **FILE *arcliente**; establece que **arcliente** es un apuntador a una estructura **FILE**. Un programa en C administra cada archivo con una estructura **FILE** diferente.

Es importante señalar que antes de trabajar con un archivo debemos abrirlo y cuando terminamos de trabajar con él debemos cerrarlo por seguridad de la información que se haya almacenado.

Cada archivo abierto debe tener por separado una declaración de tipo FILE que se utiliza para hacer referencia al archivo. La instrucción:

if ((arcliente = fopen("clientes.dat", "w")) == NULL)

nombra al archivo, **"clientes.dat"**, para que el programa lo utilice, y establece u na línea de comunicación con el archivo. A la estructura **FILE** para el archivo abierto con **fopen** se le asigna el apuntador de archivo **arcliente**. La función **fopen** toma dos argumentos: un nombre de archivo y un modo de apertura del archivo. El modo de apertura **"w"** indica que el archivo se abrirá para escritura. Si un archivo no existe y se abre para escritura, **fopen** crea el archivo. Si abre un archivo existente para escritura, el contenido del archivo se descarta sin advertencia alguna. La instrucción **if** se utiliza para determinar si el apuntador de archivo **arcliente** es **NULL** (es decir el archivo no está abierto). Si es **NULL**, el programa imprime un mensaje de error y termina. De lo contrario, el programa procesa la entrada y la escribe en el archivo.

El programa indica al usuario que introduzca los distintos campos de cada registro, o que introduzca el fin de archivo (ctrl + z) cuando la entrada de datos sea completa.

En la instrucción **while(!feof(stdin))** utiliza la función **feof** para determinar si se estableció el indicador de fin de archivo para el archivo al que hace referencia **stdin**. El indicador de fin de archivo informa al programa que ya no existen datos para procesar. El argumento de la función **feof** es un apuntador al archivo que se va a evaluar mediante el indicador de fin de archivo (en este caso, **stdin**). La función devuelve un valor diferente de cero (verdadero) cuando se establece el indicador de fin de archivo; de lo contrario la función

devuelve cero. La instrucción **while**, que incluye la llamada a **feof** en este programa, continua ejecutando el ciclo **while** mientras no se establezca el indicador de fin de archivo.

La instrucción: **fprintf(arcliente, "%d %s %.2f\n", cuenta, nombre, saldo);** escribe los datos en el archivo **clientes.dat.** La función **fprintf** es equivalente a **printf**, excepto que fprintf también recibe como argumento un apuntador para el archivo en el que se escribirán los datos.

Finalmente, después de que el usuario introduce el fin de archivo, el programa cierra el archivo **clientes.dat** con **fclose** y termina. La función **fclose** recibe como argumento el apuntador de archivo (en lugar del nombre del archivo).

La ejecución del programa se puede observar en la figura 5.13.

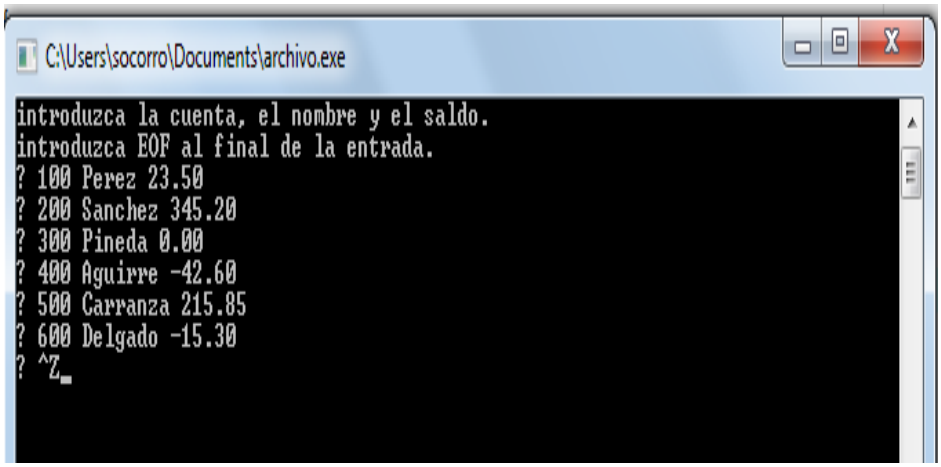


Figura 5.13 Ejecución de programa con archivo y acceso secuencial.

Lectura de datos desde un archivo secuencial.

Los datos se almacenan en archivos para que puedan recuperarse cuando sea necesario procesarlos. En el siguiente programa leeremos los datos que fueron almacenados en el archivo **"clientes.dat"** creado en el programa anterior.

```
#include <stdio.h>
#include <conio.h>

int main()
{
   int cuenta;
   char nombre[30];
   double saldo;
```

```
    FILE *arcliente;
    if ((arcliente = fopen("clientes.dat", "r")) == NULL)
        printf("El archivo no pudo abrirse\n");
    else{
            printf("%-10s%-13s%s\n", "Cuenta", "Nombre", "Saldo");
            fscanf(arcliente,"%d%s%lf", &cuenta, nombre, &saldo);
            while(!feof(arcliente))
            {
              printf("%-10d%-13s%7.2f\n", cuenta, nombre, saldo);
              fscanf(arcliente,"%d%s%lf", &cuenta, nombre, &saldo);
            }
            fclose(arcliente);
          }getch();
          return 0;
}
```

En la instrucción **FILE *arcliente;** indica que **arcliente** es un apuntador a **FILE**.

La instrucción: **if ((arcliente = fopen("clientes.dat", "r")) == NULL)**

Intenta abrir el archivo **“clientes.dat”** para lectura (**“r”**), y determina si el archivo se abrió con éxito (es decir, **fopen** no devuelve **NULL**).

La instrucción: **fscanf(arcliente,"%d%s%lf", &cuenta, nombre, &saldo);** lee un registro desde el archivo. La función **fscanf** es equivalente a la función **scanf**, con la excepción de que **fscanf** recibe como argumento el apuntador al archivo desde el que se leen los datos. Después de la primera ejecución de esta instrucción, **cuenta** tendrá el valor 100, **nombre** tendrá el valor “Perez” y **saldo** tendrá un valor de 23.50. Cada vez que se ejecuta la segunda instrucción **fscanf** (la del ciclo **while**), el programa lee otro registro desde el archivo y, **cuenta**, **nombre** y **saldo** toman nuevos valores. Cuando el programa alcanza el fin de archivo, el archivo se cierra con la instrucción **fclose(arcliente)** y el programa termina.

La ejecución del programa lo puede ver en la figura 5.14.

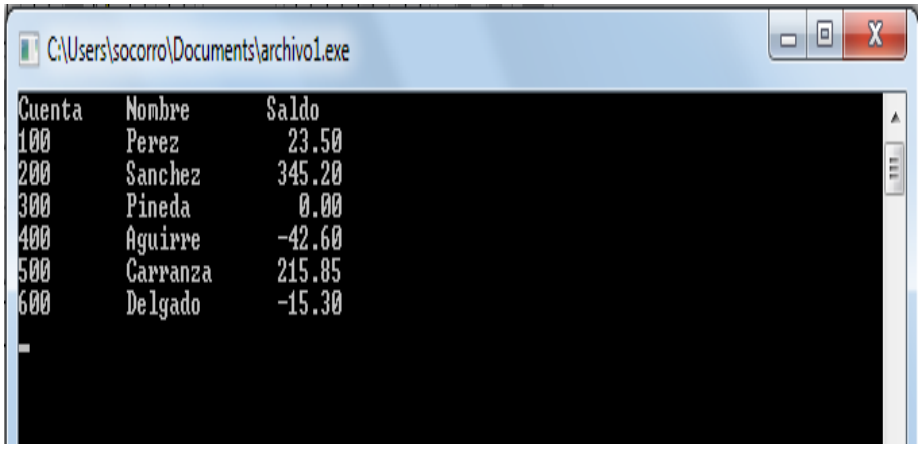


Figura 5.14. Lectura e impresión de un archivo secuencial.

El siguiente programa es un complemento del anterior en el que por medio de un menú se pueden obtener listas de los clientes con saldos en cero, con saldos a favor y clientes con saldos en contra, donde un saldo a favor es un monto negativo y un saldo en contra es un monto positivo

```
#include <stdio.h>
#include <conio.h>

int main()
{
   int opc;
   int cuenta;
   char nombre[30];
   double saldo;
   FILE *arcliente;
   if ((arcliente = fopen("clientes.dat", "r")) == NULL)
       printf("El archivo no pudo abrirse\n");
   else
       {
         printf("Introduce la opcion deseada\n"
           "1 - Lista las cuentas con saldo cero\n"
           "2 - Lista las cuentas con saldo a favor\n"
           "3 - Lista las cuentas con saldo en contra\n"
           "4 - Fin del programa\n?");
         scanf( "%d", &opc);
         while(opc != 4)
         {
            fscanf(arcliente, "%d%s%lf", &cuenta, nombre, &saldo);
            switch(opc)
            {
               case 1: printf("\n Cuentas con saldo cero:\n");
```

```
                while (!feof(arcliente))
                {
                   if (saldo == 0)
                      printf( "%-10d%-13s%7.2f\n", cuenta, nombre, saldo);
                   fscanf( arcliente, "%d%s%lf", &cuenta, nombre, &saldo);
                }
                break;
            case 2: printf("\n Cuentas con saldos a favor:\n");
                while (!feof(arcliente))
                {
                   if (saldo < 0)
                      printf( "%-10d%-13s%7.2f\n", cuenta, nombre, saldo);
                   fscanf( arcliente, "%d%s%lf", &cuenta, nombre, &saldo);
                }
                break;
             case 3: printf("\n Cuentas con saldo en contra:\n");
                while (!feof(arcliente))
                {
                   if (saldo > 0)
                      printf( "%-10d%-13s%7.2f\n", cuenta, nombre, saldo);
                   fscanf( arcliente, "%d%s%lf", &cuenta, nombre, &saldo);
                }
                break;
            }
            rewind(arcliente); /*devuelve el apuntador al principio del archivo*/
            printf("\n?");
            scanf("%d", &opc);
         }
         printf("Fin de la ejecucion.\n");
         getch();
         fclose(arcliente);
      }
      return 0;
}
```

En la figura 5.15 aparece una muestra de la salida correspondiente a la ejecución del programa.

```
C:\Users\socorro\Documents\archivo2.exe
Introduce la opcion deseada
1 - Lista las cuentas con saldo cero
2 - Lista las cuentas con saldo a favor
3 - Lista las cuentas con saldo en contra
4 - Fin del programa
?1

 Cuentas con saldo cero:
300       Pineda          0.00

?2

 Cuentas con saldos a favor:
400       Aguirre       -42.60
600       Delgado       -15.30

?3

 Cuentas con saldo en contra:
100       Perez          23.50
200       Sanchez       345.20
500       Carranza      215.85

?4
Fin de la ejecucion.
```

Figura 5.15. Salida de ejemplo del programa.

Observe que los datos en este tipo de archivo secuencial no pueden modificarse sin el riesgo de destruir otros datos del archivo. Por ejemplo, si tuviéramos que cambiar el nombre **"Pineda"** por **"Zubizarreta"**, el nombre viejo no puede simplemente sobrescribirse. El registro para **Pineda** se escribió en el archivo como

300 Pineda 0.00

Si sobrescribimos el registro comenzando en la misma posición del archivo, utilizando el nuevo nombre, el registro sería

300 Zubizarreta 0.00

El nuevo registro es mayor (tiene más caracteres) que el registro original. Los caracteres que se encuentran más allá de la **"e"** en **"Zubizarreta"** sobrescribirán el principio del siguiente registro secuencial en el archivo. El problema es que los campos(y, por lo tanto, los registros) en el *modelo de entrada/salida con formato* de **fprintf** y **fscanf**, pueden cambiar de tamaño. Por ejemplo, 7, 14, -117, 2024 y 27383 son enteros almacenados internamente en el mismo número de bytes, pero son campos de diferente tamaño cuando se despliegan en pantalla o se escriben dentro de un archivo de texto.

Por lo tanto, en general, el acceso secuencial con **fprintf** y **fsacnf** no se utiliza para actualizar datos en la misma posición. En lugar de eso, usualmente se sobrescribe el archivo completo. Para realizar el cambio de nombre anterior, los registros que se

encuentran antes de **300 Pineda 0.00**, en el archivo de acceso secuencial, se copian a un nuevo archivo. Esto requiere procesar cada registro del archivo para actualizar un registro.

Archivos de acceso directo.

Los archivos de acceso directo (o aleatorios) almacenan los datos en bloques de longitud fija. Esta característica permite tener acceso directamente a un bloque en particular, siempre que se conozca la posición en que se encuentra dicho bloque, sin necesidad de recorrer todo el archivo. Un bloque tiene siempre la misma longitud en términos de bytes y generalmente representa una estructura de datos tipo registro. Otra característica de los archivos de acceso directo es que se pueden modificar con facilidad.

En el siguiente programa se mostrará como abrir, cerrar y almacenar bloques (estructuras) en un archivo de acceso directo.

Cada bloque representa una estructura que es utilizada para almacenar información sobre alumnos de una escuela, los campos de la estructura son:

- Número de control.
- Nombre del alumno.
- Carrera.
- Calificación.

```
#include <stdio.h>

typedef struct                    /*Declaración de la estructura*/
{
       int control;
       char nom[30];
       char carrera[20];
       int calif;
}alumno;

main()
{
     alumno alu;
     int res;
     FILE *arch;
     if ((arch = fopen ("alum.dat", "w")) != NULL)
     {
        printf ("Deseas ingresar informacion de algun alumno? Si-1  No-2  ");
```

```
        scanf("%d", &res);
        while (res == 1)
        {
            printf("Numero de control: ");
            scanf("%d", &alu.control);
            printf("Nombre: ");
            fflush(stdin);
            gets(alu.nom);
            printf("Carrera: ");
            fflush(stdin);
            gets(alu.carrera);
            printf("Calificacion: ");
            scanf("%d", &alu.calif);
            fwrite(&alu, sizeof(alumno), 1, arch);
            printf ("Deseas ingresar informacion de algun alumno? Si-1  No-2  ");
            scanf("%d", &res);
        }
    }
    else
        printf("\nEl archivo no se puede abrir");
    fclose(arch);
}
```

El conjunto de instrucciones para abrir y cerrar archivos de acceso directo es idéntico a los archivos de acceso secuencial.

La función **fwrite()** se utiliza para escribir un bloque en el archivo.

Como se puede apreciar la instrucción **fwrite(&alu, sizeof(alumno), 1, arch)**; tiene cuatro argumentos:

- El primer argumento (**&alu**) indica la variable tipo estructura que se desea almacenar.
- El segundo argumento (**sizeof(alumno)**) indica el tamaño de esa variable en términos de bytes.
- El tercer argumento (**1**) indica el número de variables que se leerán o almacenaran en el dispositivo secundario.
- El cuarto argumento (**arch**) representa el apuntador al inicio de la estructura FILE.

La ejecución de este programa la puede observar en la figura 5.16 que se presenta a continuación.

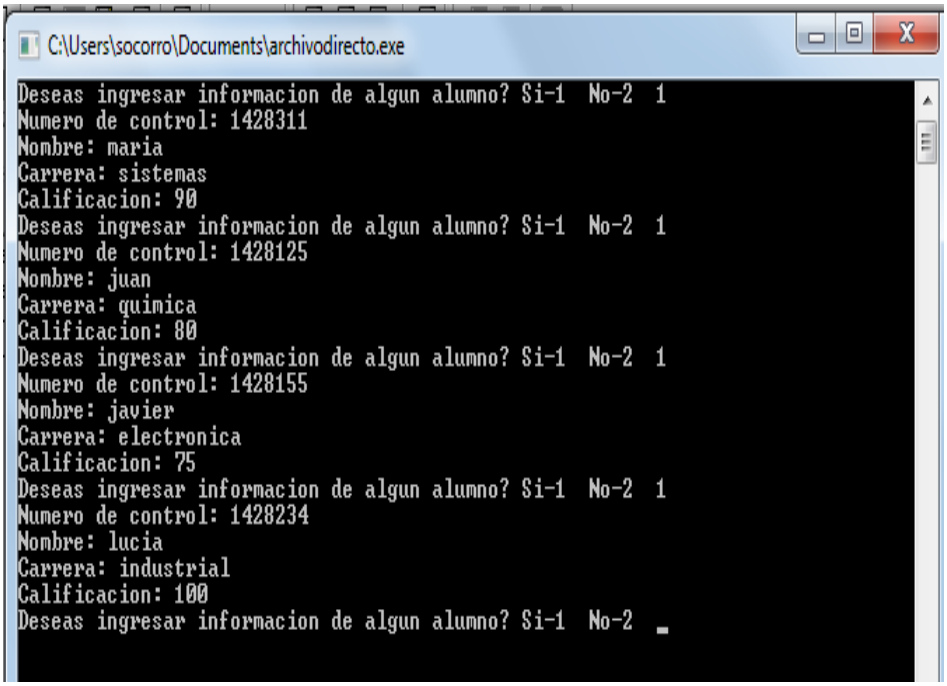


Figura 5.16. Lectura de archivo con acceso directo.

El siguiente programa recibirá como dato el archivo **"alum.dat"** se creó en el programa anterior. Se leerán los registros del archivo e imprimirá en pantalla la información de cada registro (estructura).

```
#include <stdio.h>
#include <conio.h>

typedef struct
{
        int control;
        char nom[30];
        char carrera[20];
        int calif;
}alumno;

main()
{
      alumno alu;
      int reg = 0;
      FILE *arch;
      if ((arch = fopen ("alum.dat", "r")) != NULL)
      {
          fread(&alu, sizeof(alumno), 1, arch);
          while (!feof(arch))
          {
```

```
            reg++;
            printf ("\nRegistro %d\n",reg);
            printf("\nNumero de control: %d",alu.control);
            printf("\tCarrera: %s",alu.carrera);
            printf("\tCalificacion: %d\t", alu.calif);
            printf("Nombre: ");
            puts(alu.nom);
            fread(&alu, sizeof(alumno), 1, arch);
        }
    }
    else
        printf("\nEl archivo no se puede abrir");
    getch();
    fclose(arch);
}
```

La función **fread()** es utilizada para leer bloques de un archivo, ésta tiene los mismos argumentos que la función fwrite(). Es importante que cuando se lean registros de una archivo utilizando una estructura repetitiva (como en este caso el while), se debe hacer una primera lectura antes de ingresar al ciclo y luego las siguientes dentro del ciclo, pero como última instrucción del mismo, con la finalidad de no imprimir dos veces el último registro.

La ejecución del programa se puede observar en la figura 5.17, que se presenta enseguida.

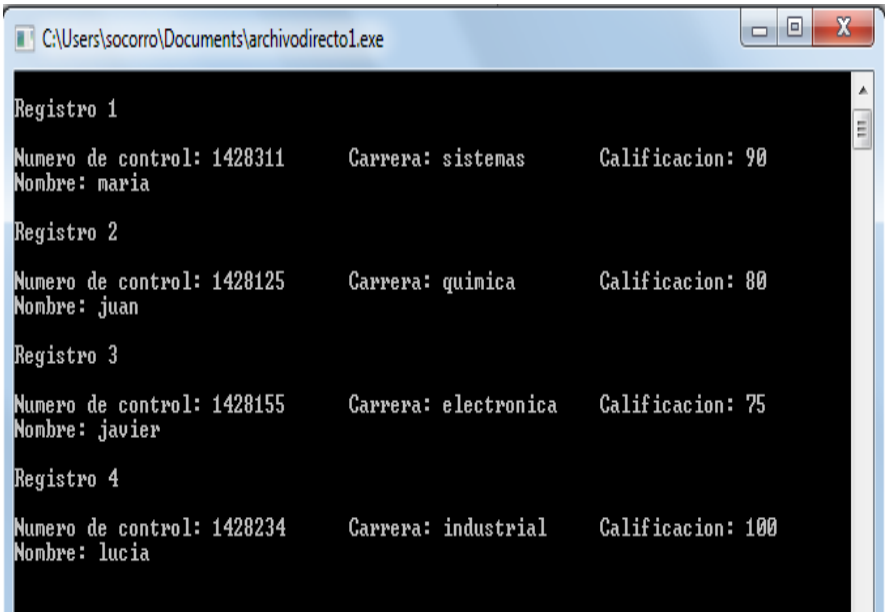


Figura 5.17. Impresión de archivo con acceso directo.

El siguiente programa realizará la modificación de un registro al archivo **“alum.dat”** mediante el número de registro del alumno al cual se desea modificar su calificación.

```
#include <stdio.h>


typedef struct
{
       int control;
       char nom[30];
       char carrera[20];
       int calif;
}alumno;

main()
{
     alumno alu;
     FILE *arch;
     int reg,res;
     if ((arch = fopen ("alum.dat", "r+")) != NULL)
     {
              printf ("Desea modificar un registro? Si-1  No-2  ");
              scanf("%d", &res);
              while (res == 1)
              {
                printf("Ingrese el numero de registro a modificar ");
                scanf("%d", &reg);
                fseek(arch, (reg-1) *sizeof(alumno),0);
                fread(&alu, sizeof(alumno), 1, arch);
                printf("ingrese la calificacion correcta: ");
                scanf("%d", &alu.calif);
                fseek(arch, (reg-1) *sizeof(alumno),0);
                fwrite(&alu, sizeof(alumno), 1, arch);
                printf ("Desea modificar otro registro? Si-1  No-2  ");
                scanf("%d", &res);
              }
     }
     else
        printf("\nEl archivo no se puede abrir");
     fclose(arch);
}
```

La función **fseek()** sitúa el apuntador en una posición aleatoria, dependiendo del desplazamiento y el origen relativo que se pasan como argumentos.

El lenguaje C almacena el primer registro en la posición cero. Por lo tanto, para modificar un registro debe buscarlo en la posición n-1.

La instrucción **fseek(arch, (reg-1) *sizeof(alumno),0)**; tiene tres argumentos:

- El primer argumento (**arch**) indica que el apuntador se debe posicionar al inicio del FILE.
- El segundo argumento (**reg-1 *sizeof(alumno)**) indica el número de bloques que debe moverse en términos de bytes para llegar al registro deseado.
- El tercer argumento (**0**) muestra a partir de qué posición se debe mover el bloque de bytes. El cero indica inicio del archivo, El 1 que se debe mover a partir de la posición que se encuentra actualmente y el 2 par indicar que se debe mover a partir del fin del archivo.

La función fseek() se utiliza dos veces, la segunda es para posicionar nuevamente en el lugar correcto para escribir el registro modificado. De lo contrario escribiremos el registro actualizado en la siguiente posición.

La ejecución del programa la puede observar en la figura 5.18.

Figura 5.18. Ejecución del programa que realiza modificaciones.

Si ejecutamos nuevamente el programa anterior después de las modificaciones la salida se vería como lo muestra la figura 5.19. En la que puede apreciar que se modificaron los 2 y 4.

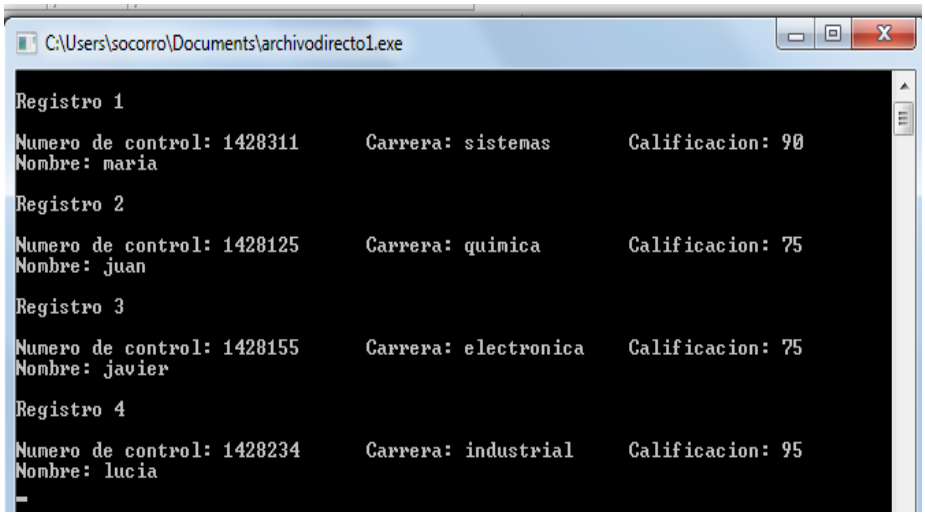


Figura 5.19. Archivo “alum.dat” modificado.

5.4 Elaboración de programas.

1. Almacene en un vector de tamaño 10 números reales. Calcule el promedio e indique cuántos elementos del vector son mayores que el promedio y genere otro arreglo con los menores o iguales.
2. Almacene en un vector n números (máximo 10), valide si el tamaño es par e invertir los elementos a la mitad del arreglo. Imprimir el vector original y el invertido.
3. Almacene en dos vectores números reales a y b de 10 elementos cada uno, a partir de ellos genere un tercer vector con el siguiente criterio: sumar el primer elemento de a más el último elemento de b y luego el segundo elemento de a con el noveno elemento de b y así sucesivamente hasta llegar al décimo elemento de a más el primer elemento de b. Imprimir las sumas almacenadas en el vector c.
4. Almacene 2 vectores a y b de longitudes n y m, el primero ordenado en forma ascendente y el segundo ordenado de manera descendente, crear un nuevo vector c de n + m elementos intercalando los elemento de a y b de modo que c quede ordenado en forma ascendente.
5. Almacene en un arreglo n números enteros y determine ¿Cuántas veces se repite cada uno de ellos?
6. Almacene en un vector 10 números y cambie algún número por otro del mismo arreglo, realice el cambio y muestre el arreglo modificado.
7. Almacene en un arreglo, n elementos (máximo 30) e imprima la suma de:
   a) Números pares.
   b) Números impares.
   c) Total de los elementos del arreglo.
8. Genere aleatoriamente un vector de tamaño 20 con números entre 0 y 150 y genere otros tres vectores con los siguientes criterios:

a) Si los números están comprendidos entre 0 y 50 se guardaran en el vector1.
b) Si los números están comprendidos entre 51 y 100 se guardaran en el vector 2.
c) Si los números son mayores a 100 se guardaran en el vector 3.

Al final imprimir los cuatro vectores.

9. Almacene en un arreglo, 20 elementos de tipo entero e imprimir:
   a) La suma de los elementos que ocupan las posiciones pares.
   b) El mayor de los elementos que ocupan posiciones impares.
   c) La posición del mayor número par.
10. Almacene en un vector n elementos de tipo real (máximo 25) e imprima los siguientes valores:
    a) Máximo.
    b) Mínimo.
    c) La media de los que estén en posiciones pares.
    d) La varianza.
11. Calcular la matriz traspuesta.
12. Calcular la raíz cuadrada de una matriz.
13. Calcular la inversa de una matriz cuadrada de 3 X 3.
14. Almacene los elementos de una matriz de m X n (máximo 10 X 10) e imprimir cuántos números son ceros, cuántos negativos, cuántos positivos. Imprimir además la suma de los negativos y la suma de los positivos.
15. Leer los elementos de una matriz de m X n (máximo 10 X 10) y multiplicarla por un escalar.
16. Almacene los elementos de una matriz m X n (máximo 10 X 10) e imprimir:
    a) Suma de los elementos.
    b) Promedio de los elementos.
    c) Elemento que más se repite (moda).
17. Leer los elementos de una matriz cuadrada (máximo 15 X 15) e imprimir la suma de todos los elementos que no son de la “periferia” de la matriz.
18. Genere una matriz de 4 X 4 con las primeras 16 letras del abecedario en minúsculas.
19. Almacene los elementos de una matriz n X m (10 X 10 como máximo) y posteriormente almacene toda la matriz en un vector. Imprimir el vector resultante.
20. Leer los elementos de una matriz cuadrada de tamaño n (10 X10 máximo), calcular la sumatoria de los elementos de las diagonales (principal e inversa) y guardar estos elementos en dos vectores (DP y DI).
21. Genere una matriz espiral (en forma de caracol) de n X n.
22. Leer los elementos de una matriz n X m (máximo 10 X 10) y generar un vector que almacene el máximo elemento de cada columna de la matriz. Imprimir dicho vector.
23. Escriba un programa que, al recibir como datos el archivo arch.txt compuesto por cadena de caracteres, calcule el número de cada una de las vocales que se encuentran en el archivo.

24. Escriba un programa que, al recibir como datos una cadena de caracteres y el archivo de texto arch1.txt, compuesto también por cadenas de caracteres, determine cuantas veces se encuentra la primera cadena de caracteres en el archivo.

6. FUNCIONES.

6.1 Introducción.

La programación modular es una técnica que consiste en separar un problema en las diferentes tareas que se requieren resolver, dando origen a la creación de módulos (pequeños programas a los que llamaremos funciones), donde cada módulo o función se diseña, se codifica y se procesa de manera independiente. (MÁRQUEZ FRAUSTO, OSORIO ÁNGEL, & OLVERA PÉREZ, 2011)

En el lenguaje de programación C la solución de un problema se expresa por medio de un programa; la solución de un subproblema, por medio de una función. El uso de funciones tiene múltiples ventajas: facilita la lectura y escritura de los programas, permite el trabajo en paralelo –diferentes programadores se pueden encargar de diferentes funciones-, facilitan la asignación de responsabilidades, permiten que el código de la función se escriba solamente una vez y se utilice tantas veces como sea necesario, facilitan el mantenimiento de los programas, etc. (CAIRÓ BATTISTUTTI, 2006)

Un programa en C está constituido por un programa principal y un conjunto de funciones. El programa principal consta generalmente de pocas líneas, las cuales pueden ser llamadas a funciones. La llamada a una función indica al procesador que debe continuar con el procesamiento de la función. Una vez que ésta termine, el control regresa al punto de partida en el programa principal. La función se escribe de forma similar al programa principal, pero con diferencias principalmente en el encabezado de la misma. Una función resuelve un subproblema de forma independiente y se ejecuta sólo cuando recibe una llamada desde el programa principal o desde otras funciones. El lenguaje de programación C permite que una función pueda incorporar llamadas a otras funciones.

La comunicación entre las funciones y el programa principal, al igual que entre las mismas funciones, se lleva a cabo por medio de **parámetros por valor**, **parámetros por referencia** y **variable globales**. Estas últimas son menos utilizadas por razones de eficiencia y seguridad en la escritura de programas.

6.2 Funciones estándar.

Todas las versiones del lenguaje C ofrecen una biblioteca estándar de funciones que proporcionan soporte para operaciones utilizadas con más frecuencia. Estas funciones permiten utilizar una operación con sólo una llamada a la función (sin necesidad de escribir su código fuente). (JOYANES AGUILAR, 2005)

Las **funciones estándar o predefinidas**, como así se denominan las funciones pertenecientes a la biblioteca estándar, se dividen en grupos; todas las funciones que pertenecen al mismo grupo se declaran en el mismo *archivo de cabecera*. (JOYANES AGUILAR, 2005)
En este capítulo sólo veremos las funciones más comunes. En la tabla 6.1 podemos observar las funciones matemáticas.

Tabla 6.1. Funciones matemáticas (librería *math.h*).

Nombre	**Descripción**	**Tipo de argumentos**	**Tipo de valor de regreso**
abs(x)*	Valor absoluto de un int	int	int
acos(x)	Arco coseno de *x*	double	double**
asin(x)	Arco seno de *x*	double	double**
atan(x)	Arco tangente de *x*	double	double**
ceil(x)	Redondea hacia el número inmediato superior	double	double
cos(x)	Coseno de *x* en radianes	double	double
exp(x)	Exponencial de *x*, e^x	double	double
fabs(x)	Valor absoluto de un double	double	double
floor(x)	Redondeo hacia el número inmediato inferior	double	double
fmod(x,y)	Residuo de *x*/*y* como número de punto flotante	double	double
log(x)	Logaritmo natural de *x*	double	double
log10(x)	Logaritmo en base 10 de *x*	double	double
pow(x,y)	Potencia x^y	double	double
sin(x)	Seno de *x* en radianes	double	double
sqrt(x)	Raíz cuadrada de *x*	double	double
tan(x)	Tangente de *x* en radianes	double	double

*La función está incluida en la librería *stdlib.h*.

Programa 1. Leer el ángulo en grados y mediante un menú de 3 opciones calcular: s) seno, c) coseno y t) tangente.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

#define PI 3.1416

main()
{
    char op;
    float ang, resp;
    printf("Menu funciones trigonometricas\n\n");
```

```
    printf(" s)seno\n");
    printf(" c)coseno\n");
    printf(" t)tangente\n");
    printf("\nQue opcion deseas? ");
    scanf("%s", &op);
    printf("Escribe el angulo en grados: ");
    scanf("%f", &ang);
    switch(op)
    {
        case 'S': case 's': resp = sin(ang*PI/180);
                            printf ("seno = %f",resp);
                            break;
        case 'C': case 'c': resp = cos(ang*PI/180);
                            printf ("coseno = %f",resp);
                            break;
        case 'T': case 't': resp = tan(ang*PI/180);
                            printf ("tangente = %f",resp);
                            break;
            default: printf(" Opción no valida ");
    }
    getch();
}
```

Una posible ejecución del programa se puede observar en la figura 6.1.

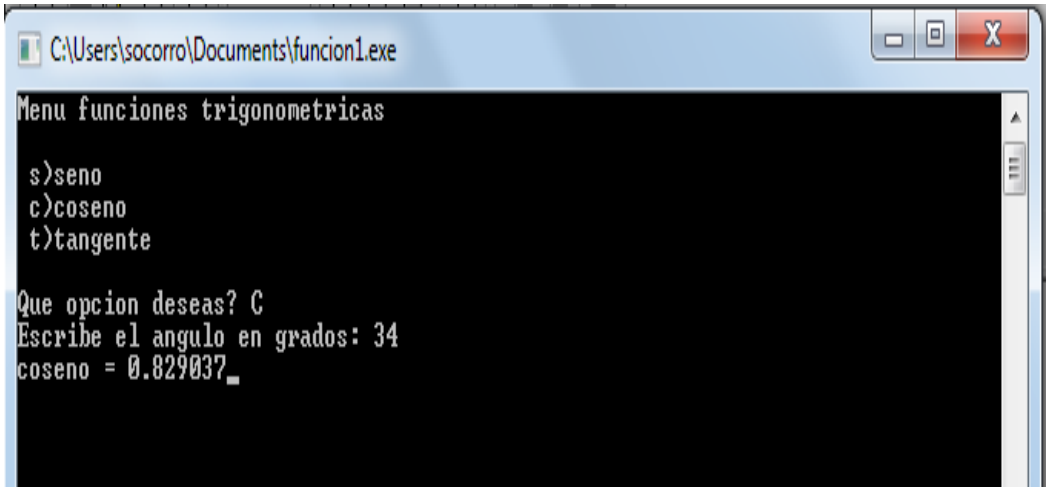


Figura 6.1. Librería math.h.

Fundamentos de cadenas y caracteres.

Los caracteres son los bloques de construcción fundamentales para los programas fuente. Cada programa está compuesto por una secuencia de caracteres que, cuando se agrupan apropiadamente, la computadora los interpreta como un conjunto de instrucciones utilizadas para llevar a cabo una tarea. Un programa puede contener *constantes de carácter*. Una constante de carácter es un valor **int** representado por un carácter entre comillas sencillas. El valor de una constante de carácter es el valor entero del carácter en el *conjunto de caracteres* de la máquina. Por ejemplo, **‘ z ’** representa el valor entero de **z**, y **‘ \n ’** representa el valor entero de una nueva línea. (DEITEL & DEITEL, 2004)

Una cadena es un conjunto de caracteres tratados como una sola unidad. Una cadena puede incluir letras, dígitos y varios *caracteres especiales* como +, -, *, /, y $. En C, las *literales de cadena*, o *constantes de cadena*, se escriben dentro de comillas dobles de la siguiente manera:

“Juan P. Pérez”	(un nombre)
“98256 de Eje Central”	(la dirección de una calle)
“México, Distrito Federal”	(una ciudad y un estado)
“(55)54 32 20 70”	(un número telefónico)

En C, una cadena es un arreglo de caracteres, los cuales terminan con el *carácter nulo* (‘\0’). Se accede a una cadena mediante un apuntador a su primer carácter. El valor de una cadena es la dirección del primer carácter. Así, en C, es apropiado decir que *una cadena es un apuntador*, de hecho, un apuntador al primer carácter de la cadena. En este sentido, las cadenas son como los arreglos, debido a que un arreglo también es un apuntador a su primer elemento. (DEITEL & DEITEL, 2004)

Un arreglo de caracteres o una variable de tipo **char *** puede inicializarse con una cadena en la definición. Las definiciones

```
char color[ ] = “azul”;
const char *ptrColor = “azul”;
```

inicializan una variable con la cadena **“azul”**. La primera definición crea un arreglo de 5 elementos, **color**, que contiene los caracteres **‘a’**, **‘z’**, **‘u’**, **‘l’** y **‘\0’**. La segunda definición crea una variable apuntador, **ptrColor**, que apunta a la cadena **“azul”** en algún lugar de la memoria.

Funciones para manejo de caracteres y cadenas.

Las funciones para manejo de caracteres y cadenas se encuentran en las librerías *string.h*, *ctype.h* y *stdlib.h*. Varias de las macros y funciones declaradas en <ctype>

realizan pruebas y conversiones de caracteres. En la tabla 6.2 podemos ver las funciones incluidas en la biblioteca *string.h*.

Tabla 6.2. Operaciones con cadenas de caracteres (*string.h*).

Nombre	**Descripción**
strcat(cad1,cad2)	Concatena *cad2* al final de *cad1* (incluyendo el carácter nulo); el carácter inicial de *cad2* sobrescribe el carácter nulo al final de *cad1*. Devuelve el valor de *cad1*.
strcmp(cad1,cad2)	Compara los elementos de dos cadenas *cad1* y *cad2*. Si son iguales, devuelve 0. Si el de *cad1* es mayor que *cad2*, devuelve un valor mayor que cero; en caso contrario, devuelve un valor menor que cero.
strcpy(cad1,cad2)	Copia la cadena *cad2*, incluyendo el nulo, en el arreglo *cad1*. Devuelve el valor de *cad1*.
strlen(cad)	Devuelve el número de caracteres de la cadena *cad*, sin incluir el nulo de terminación.
strncat(cad1,cad2,n)	Concatena *n* caracteres de *cad2* a *cad1*. Devuelve el valor de *cad1*.
strncmp(cad1,cad2,n)	Es igual que *strcmp* pero con *n* caracteres, compara los *n* primeros caracteres de una cadena.
strncpy(cad1,cad2,n)	Copia los primeros *n* caracteres de *cad2* a *cad1*. Devuelve el valor de *cad1*.
strchr(cad,c)	Localiza la primera ocurrencia del carácter *c* en la cadena *cad*; si no se encuentra, devuelve un puntero nulo.
strstr(cad1,cad2)	Busca en *cad1* la subcadena *cad2*, si la encuentra devuelve un puntero hacia la primera ocurrencia de *cad2* en *cad1*, si no regresa *null*.
strupr(cad)	Convierte a mayúsculas la cadena *cad*.
strlwr(cad)	Convierte a minúsculas la cadena *cad*.

Programa 2. Leer el password, si es correcto imprimir “Correcto, puedes continuar” y si no imprimir “Password incorrecto”.

```
#include <string.h>
#include <stdio.h>
#include <conio.h>

main()
{
    char cad[80];
    puts ("Escribe tu password: "); gets (cad);
    if ((strcmp(cad, "acceso1234")== 0 || strcmp(cad, "ACCESO1234") == 0))
      puts("Correcto, puedes continuar");
    else puts(" Password incorrecto");
    getch();
}
```

En la figura 6.2 se muestra la ejecución del programa.

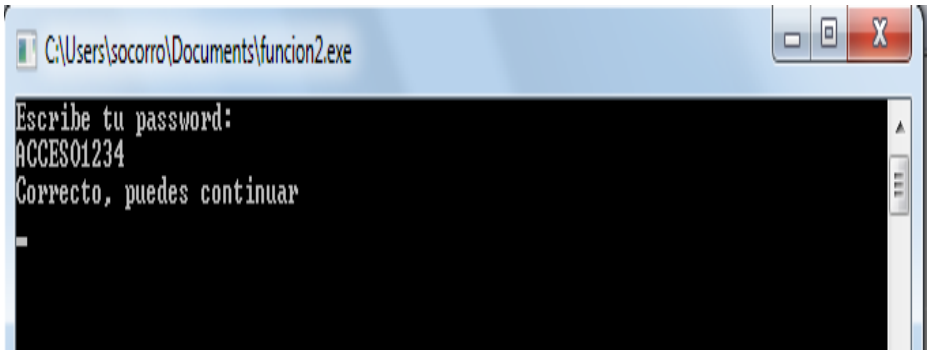


Figura 6.2. Librería string.h.

Tabla 6.3. Pruebas y conversiones de caracteres (*ctype.h*).

isalpha(c)	Comprueba si un carácter es alfabético.
isdigit(c)	Comprueba si un carácter es un dígito decimal.
islower(c)	Comprueba si un carácter es de tipo minúscula. Devuelve un valor distinto de cero si *c* es cualquiera de las letras minúsculas.
ispunct(c)	Comprueba si un carácter es signo de puntuación.
isspace(c)	Comprueba si un carácter es un espacio.
isupper(c)	Comprueba si un carácter es de tipo mayúscula. Devuelve un valor distinto de cero si *c* es cualquiera de las letras mayúsculas.
toascii(c)	Convierte *c* a su correspondiente código ASCII; regresa un valor dentro del rango entre 0 y 127.
tolower(c)	Devuelve la correspondiente letra minúscula si existe y si *isupper(c)* es distinto de cero, en caso contrario, devuelve *c*.
toupper(c)	Devuelve la correspondiente letra mayúscula si existe y si *islower(c)* es distinto de cero, en caso contrario, devuelve *c*.

Programa 3. El siguiente programa utiliza las funciones *isdigit*, *isalpha*, *islower*, *isupper*, *toupper*, *tolower*, *isspace* y *ispunct*.

```
#include <stdio.h>
#include <ctype.h>
#include <conio.h>
```

```
main()
{
    printf("%s\n%s%s\n%s%s\n\n",
    "De acuerdo con isdigit: ",
    isdigit('8')? "8 es un " : "8 no es un ", "digito",
    isdigit('#')? " # es un ": "# no es un ", "digito");

    printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
    "De acuerdo con isalpha: ",
    isalpha('A')? "A es una " : "A no es una ", "letra",
    isalpha('b')? "b es una " : "b no es una ", "letra",
    isalpha('&')? "& es una " : "& no es una ", "letra",
    isalpha('4')? "4 es una " : "4 no es una ", "letra");

    printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
    "De acuerdo con islower: ",
    islower('p')? "p es una " : "p no es una ","letra minuscula",
    islower('P')? "P es una " : "P no es una ","letra minuscula",
    islower('5')? "5 es una " : "5 no es una ","letra minuscula",
    islower('!')? "! es una " : "! no es una ","letra minuscula");

    printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
    "De acuerdo con isupper: ",
    isupper('D')? "D es una " : "D no es una ","letra mayuscula",
    isupper('d')? "d es una " : "d no es una ","letra mayuscula",
    isupper('8')? "8 es una " : "8 no es una ","letra mayuscula",
    isupper('$')? "$ es una " : "$ no es una ","letra mayuscula");

    printf("%s%c\n%s%c\n%s%c\n%s%c\n\n",
    "u convertida a mayuscula es ", toupper('u'),
    "7 convertida a mayuscula es ", toupper('7'),
    "$ convertida a mayuscula es ", toupper('$'),
    "L convertida a minuscula es ", tolower('L'));

    printf("%s\n%s%s%s\n%s%s%s\n%s%s\n\n",
    "De acuerdo con isspace: ",
    "Nueva linea", isspace('\n') ? " es un " : " no es un ",
    "caracter espacio en blanco",
    "Tabulador horizontal ", isspace('\t') ? " es un " : "  no es un ",
    "caracter espacio en blanco",
    isspace('%') ? "% es un " : "% no es un ",
    "caracter espacio en blanco");

    printf("%s\n%s%s\n%s%s\n%s%s\n",
```

```
    "De acuerdo con ispunct: ",
    ispunct(';')? "; es un " : "; no es un ","caracter de puntuacion",
    ispunct('Y')? "Y es un " : "Y no es un ","caracter de puntuacion",
    ispunct('#')? "# es un " : "# no es un ","caracter de puntuacion");
    getch();
}
```

La ejecución del programa la puede observar en la figura 6.3.

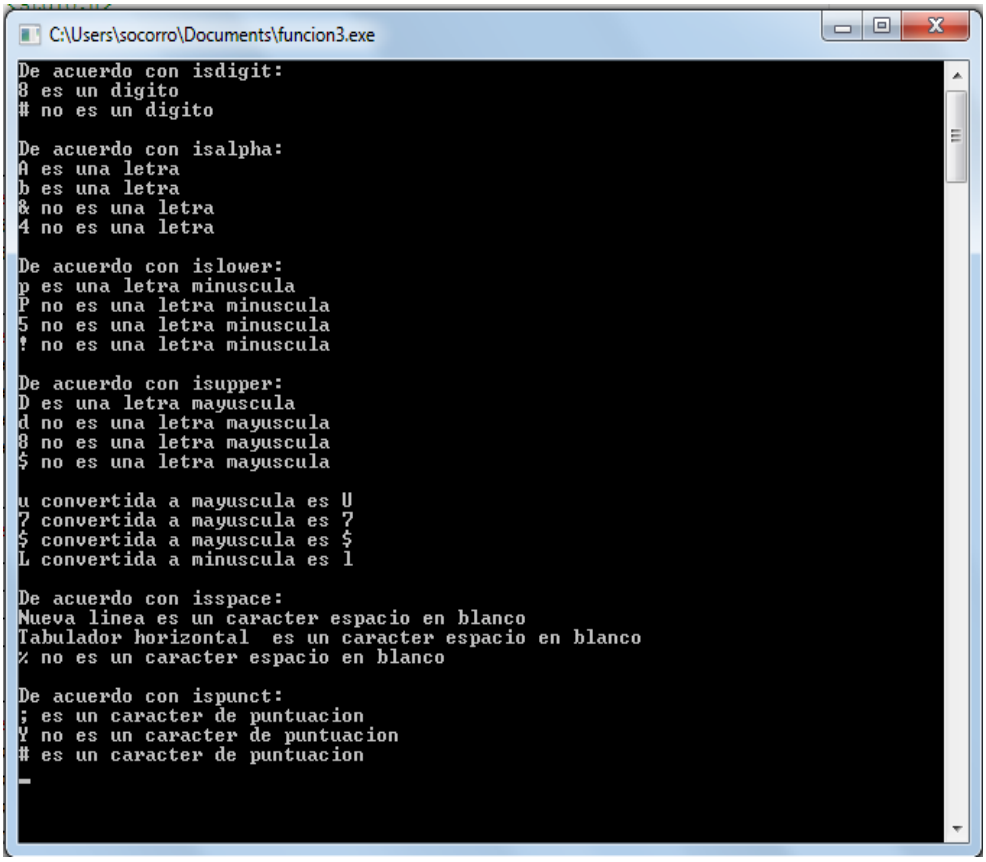


Figura 6.3. Librería ctype.h.

En la tabla 6.4 puede observar las funciones que se utilizan para operaciones con cadenas de caracteres *stdlib.h.*

Tabla 6.4. Funciones para la conversión de tipos (stdlib.h).

Nombre	**Descripción**	**Tipo de valor de regreso**
atof(cad)	Convierte los caracteres de la cadena *cad* a la representación interna de tipo *double* y devuelve ese valor.	double
atoi(cad)	Convierte los caracteres de la cadena *cad* a la representación interna de tipo *int* y devuelve ese valor.	int

Programa 4. El siguiente programa utiliza las funciones de la tabla 6.4.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

main()
{
    double d;
    int i;
    d = atof("99.0");
    i = atoi("2593");
    printf ("Funcion atof\n\n");
    printf("%s%.3f\n%s%.3f\n\n", "La cadena convertida a double es ",d,
    "El valor convertido dividido entre 2 es ", d/2.0);
    printf ("Funcion atoi\n");
    printf("%s%d\n%s%d\n\n", "La cadena \"2593\" convertida a int es ",i,
    "El valor convertido menos 593 es ", i-593);
    getch();
}
```

La ejecución de este programa lo puede ver en la figura 6.4

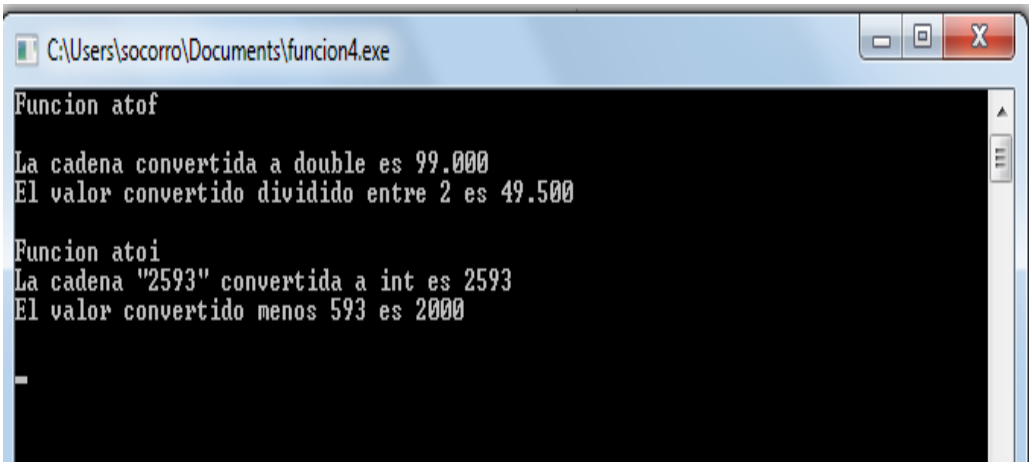


Figura 6.4. Librería stdlib.h.

6.3 Entrada y salida de datos.

Los programas interactúan con el exterior, a través de datos de entrada o datos de salida. La biblioteca C proporciona facilidades para entrada y salida, para lo que todo programa deberá tener el archivo de cabecera *stdio.h*. En C la entrada y la salida se lee y escribe de

los dispositivos estándar de entrada y salida, se denominan *stdin* y *stdout* respectivamente. La salida normalmente es a pantalla del ordenador, y la entrada se capta del teclado. (JOYANES AGUILAR, 2005)

En el archivo *stdio.h* están definidas macros, constantes, variables y funciones que permiten intercambiar datos con el exterior.

La entrada y salida con formato se realiza por medio de las funciones *scanf()* y *printf()*. Las *f* de *scanf()* y *printf()* significan "con formato". (CORONA NAKAMURA & ANCONA VALDEZ, 2011)

Entrada de datos (*scanf()*).

La entrada de datos a un programa puede tener diversas fuentes, teclado, archivos en disco, etc. La entrada que consideraremos ahora es a través del teclado, asociado al archivo estándar de entrada **stdin**. La función más utilizada, por su versatilidad, para entrada formateada es *scanf()*.

La sintaxis es: *scanf ("cadena de control", , argumento1, argumento2…);*

La *cadena de control* contiene uno o varios *especificadores* o *códigos de formato* que harán referencia al tipo de dato de los argumentos y, si se desea, su anchura. Los códigos estarán precedidos por el carácter *%* seguido de un *carácter de conversión*. La *cadena de control* deberá escribirse entre comillas (" ").

La lista de *argumentos* serán las variables; *scanf* necesita saber la posición de memoria de la computadora en que se encuentra la variable para poder almacenar la información obtenida, para ello se utilizará el símbolo *ampersand* (*&*), que se colocará antes del nombre de cada variable. El *&* sirve para apuntar a la dirección de memoria de dicha variable.

Ejemplos:

```
char x;
scanf( "%c", &c);  //se captura un carácter y se almacena en la variable x
```

No es necesario el & cuando el dato que se va a capturar es una cadena de caracteres, ya que con el nombre de la cadena (en general, de un arreglo) sin índices se identifica la dirección del primer elemento. (CORONA NAKAMURA & ANCONA VALDEZ, 2011)

```
char cadena[30];
scanf("%s", cadena);        o        scanf("%s", &cadena);
```

El *scanf ("%f %d %c", &x, &y, &z)*; almacena en *x*, *y* y *z*, respectivamente, los datos introducidos de cualquiera de las siguientes formas:

- Oprimir la tecla <enter> después de introducir cada dato.
- Oprimir la tecla <TAB> después de introducir cada dato.
- Oprimir un espacio en blanco después de introducir cada dato.
- Alternar cualquiera de las opciones anteriores entre cada dato.

Hay que tener precaución con el uso de los espacios en blanco. Una sentencia como scanf ("%s", cadena); no volverá hasta que no se teclee un carácter no blanco, puesto que el espacio que hay después de %s hace que se lean y descarten espacios, tabuladores y saltos de línea. (CORONA NAKAMURA & ANCONA VALDEZ, 2011)

Otra forma es delimitar la entrada de cada variable, haciendo uso de la coma, por ejemplo:

scanf("%d,%d", &x, &y); //la entrada podría ser 18,45

En la tabla 6.5 podemos observar el formato de las variables de acuerdo con su tipo.

Tabla 6.5. Formato de escritura de las variables.

Formato	**Descripción**
%u	Escribe enteros sin signo de 2 bytes (unsigned int).
%d %i	Escribe enteros de 2 bytes (int).
%ld	Imprime enteros de largo alcance (long).
%f	Escribe reales de 4 bytes (float).
%lf	Escribe reales de doble precisión, 8 bytes (double).
%e	Imprime en forma exponencial.
%g	Imprime en %f o %e en función del tamaño del número.
%c	Escribe un carácter de un byte (char).
%s	Escribe una cadena de caracteres, que termina en '\0'.
%p	Escribe la dirección de un puntero.

Salida de datos (*printf()*).

La salida de datos de un programa se puede dirigir a diversos dispositivos, pantalla, impresora, archivos. La salida que se tratará a continuación será a pantalla. La función printf() es similar a la función scanf(), pero se usa para la salida; permite escribir textos y datos en la pantalla con determinado formato.

La sintaxis es: *printf ("cadena de control", argumento1, argumento2…);*

La *cadena de control* debe escribirse entre comillas (" ") y puede tener al menos uno de los tres elementos siguientes:

1. Los caracteres que se imprimirán en pantalla (los cuales se visualizarán idénticos).
2. Secuencia de escape.
3. Uno o varios códigos de formato (indica de qué tipo de dato se trata), los cuales determinan cómo se verán en pantalla los argumentos.

En la tabla 6.6 podemos observar las secuencias de escape y en la tabla 6.5 los códigos de formato.

Tabla 6.6 Secuencias de escape.

Secuencia de escape	**Significado**
\a	Alerta
\b	Espacio atrás
\f	Salto de página
\n	Salto de línea
\r	Retorno de carro
\t	Tabulación horizontal
\v	Tabulación vertical
\\	Barra invertida
\’	Comilla simple
\”	Comillas dobles
\ooo	Visualiza un carácter cuyo código ASCII es OOO en octal
\xhhh	Visualiza un carácter cuyo código ASCII es HHH en hexadecimal

Un *código de formato* se inicia con un carácter *%* y termina con un carácter de conversión; por ejemplo, en el formato *%s* la letra ese (*s*) es el carácter de conversión para imprimir cadenas. Debe haber el mismo número de *códigos de formato* que de *argumentos* y cada uno de éstos debe tener un orden. En caso contrario el resultado será incorrecto.

Los *argumentos* pueden ser variables, constantes o en general expresiones de salida.

Ejemplos: Para imprimir las letras XYZ en la pantalla podría hacerse de las siguientes maneras:

printf (“XYZ”); //La lista de argumentos es opcional.
printf (“*%s*”, ”XYZ”); // El formato *%s* imprime el texto “XYZ” como *cadena de caracteres*.
printf (“*%c%c%c*”,’X’, ’Y’, ’Z’); //Los apóstrofos que encierran cada letra se emplean para designar constantes de caracteres. El formato *%c* imprime el valor como un carácter.

Los códigos de formato pueden control aspectos de la apariencia de los datos tales como la longitud, el número de decimales y la justificación a la derecha o izquierda.

Sintaxis % [signo] [longitud] [.precisión] [l/L] Carácter de conversión

Los valores entre [ ] son opcionales, donde:

- *signo*: Indica si el valor aparecerá a la izquierda (signo menos), o a la derecha (por defecto).
- *longitud*: Longitud máxima del valor que aparecerá en la pantalla.
- *precisión*: Indica el número máximo de decimales que tendrá el valor.
- *l/L*: Se utiliza *l* cuando se trata de una variable de tipo *long*, y *L* cuando es de tipo *double*.

El *código de formato %4d* indica que se imprimirá un número entero en cuatro posiciones o lugares, justificando a la derecha. El relleno es blanco.

Ejemplo de salida 1: printf (“Numero%4dEntero”,96);

Veremos dos espacios en blanco entre “Número” y el 96 como lo muestra la figura 6.5.

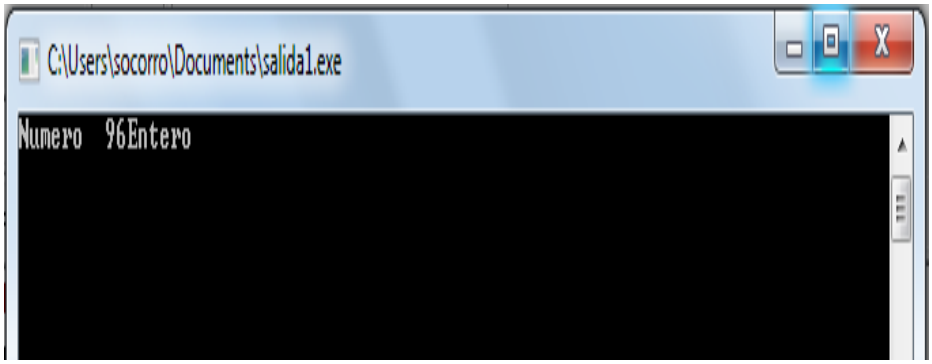


Figura 6.5. Salida con formato y alineación a la derecha.

Para rellenar con ceros en lugar de espacios en blanco, colocamos un 0 antes del indicador de tamaño:

Ejemplo de salida 2: printf (“Número%04dEntero”,96);

La ejecución de la instrucción se vería como se muestra en la figura 6.6.

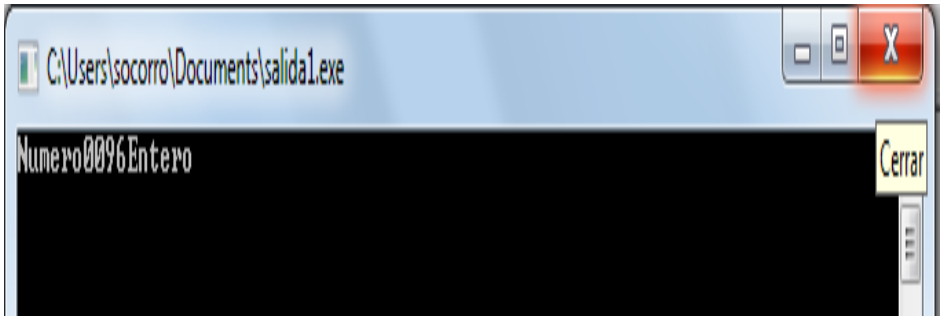


Figura 6.6. Salida con formato y relleno.

Si se desea salida formateada justificada a la izquierda se debe colocar un guión después del signo %:

Ejemplo de salida 3: printf (“Número%-4dEntero”,96);

Los espacios estarán a la derecha y se visualizaría como lo muestra la figura 6.7.

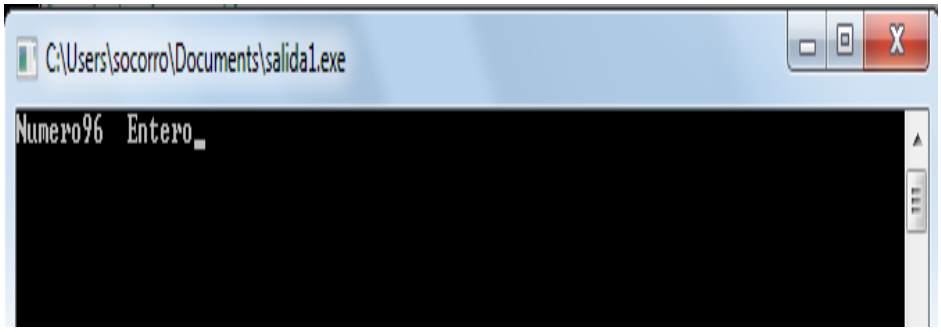


Figura 6.7. Salida con formato y alineación a la izquierda.

Si una cadena o número es mayor que el ancho especificado, éstos se imprimirán completos.

La sintaxis del formato de un número con parte decimal es: *%l.df*

Donde *l* es la longitud total del campo (incluyendo el punto decimal) y *d,* el número de decimales.

El especificador de formato %9.3f nos dice que la variable ocupará nueve lugares: cinco para la parte entera, uno para el punto decimal y tres para los decimales.

Cuando se usa el punto decimal en una cadena el valor que le sigue al punto indica el máximo ancho de la cadena. El formato %3.5s, indica que la cadena tendrá al menos tres caracteres y un máximo de cinco. Si la cadena sobrepasa el tamaño máximo, se trunca por la derecha.

Ejemplo de salida 4: printf(“Cadena%3.5sGrande”,”1234567”);

En la pantalla veremos como se muestra en la figura 6.8.

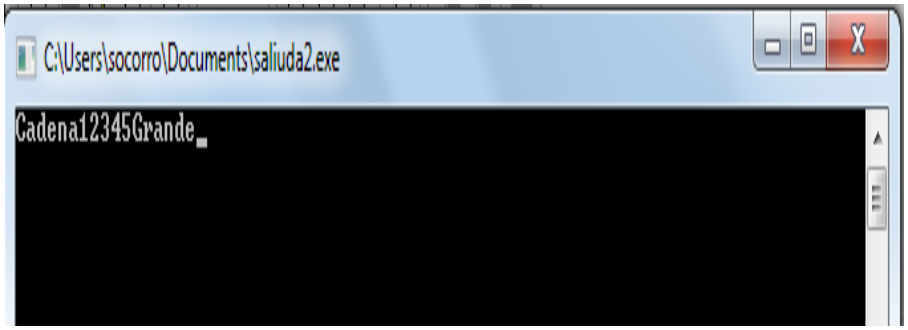


Figura 6.8. Salida con formato decimal.

Entrada y salida sin formato.

Las funciones principales que realizan la entrada y salida sin formato las puede visualizar en la tabla 6.7.

Tabla 6.7 Funciones de entrada y salida sin formato.

Función	Descripción
getche()	Lee un carácter del teclado, no hay que esperar hasta que se pulse la tecla <enter>. Visualiza el eco del carácter.
getchar()	Lee un carácter del teclado, espera hasta que se pulsa la tecla <enter>.
getch()	Lee un carácter del teclado, no hay que esperar que se pulse la tecla <enter>. No visualiza el eco del carácter.
putchar()	Imprime un carácter en la pantalla en la posición actual del cursor.
gets()	Lee una cadena de caracteres introducida por el teclado.
puts()	Imprime una cadena en la pantalla, seguida de un carácter de salto de línea.

Ejemplo de un programa con funciones de entrada y salida sin formato.

```
#include <stdio.h>
#include <conio.h>
main()
{
    char c;
    puts ("Dame un caracter: "); fflush(stdin);
    c = getche();
    putchar(c);
    puts ("Dame otro caracter: "); fflush(stdout);
    c = getchar();
    putchar (c);
    puts ("Dame otro caracter: "); fflush(stdin);
    c = getch();
    putchar(c);
    getch();
}
```

La ejecución del programa lo puede ver en la figura 6.9.

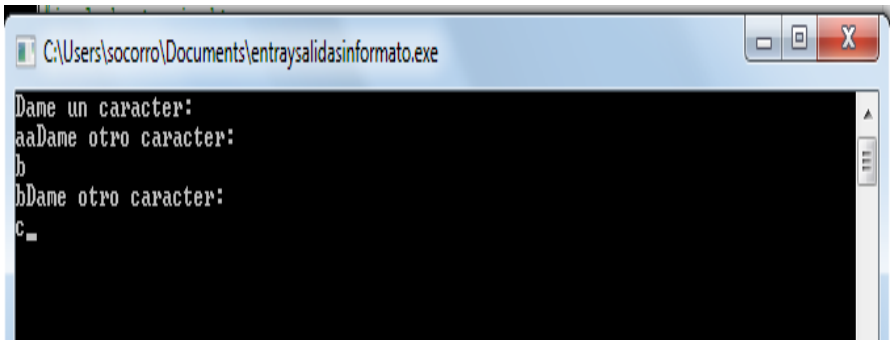


Figura 6.9. Salida sin formato.

6.4 Funciones definidas por el usuario.

En el lenguaje C se pueden distinguir dos tipos de funciones: las predefinidas y las definidas por el programador. Las funciones predefinidas se encuentran en las bibliotecas estándar de C (como se vio en el subtema 6.2). Con respecto a las definidas por el programador, éste las diseña según sus necesidades. En este subtema se explicará la forma de crear sus propias funciones.

Función *main()*.

Es la función principal y puede contener de pocas a muchas líneas; su papel es coordinar a las otras funciones mediante llamadas o invocaciones. (JOYANES AGUILAR, 2005)

Una función es un subprograma que realiza una tarea específica que puede o no recibir valores (parámetros). En C podemos devolver cualquier tipo de datos escalares (puntero, tipo numérico y el tipo carácter o en su caso regresar un valor nulo que llamaremos nada o ninguno). Asimismo, no se pueden devolver arreglos ni estructuras. El uso de las funciones es una práctica común y recomendable ya que permite dividir el código, simplificando así el desarrollo y la depuración del mismo. Para utilizar funciones en un programa es necesario declararlas previamente. (CORONA NAKAMURA & ANCONA VALDEZ, 2011)

Variables.

Variable local. Es la declarada en una determinada función, sólo se encuentra disponible durante el funcionamiento de la misma, es decir está en memoria cuando dicha función está activa.

Variable global. Variable declarada fuera de cualquier función y que puede ser utilizada por las funciones que se encuentran después de dicha declaración; por lo tanto si la

declaración junto a las librerías, la podrá utilizar todo el programa. Esta característica es propia del lenguaje C.

Llamada o invocación de una función.

Para llamar una función es necesario es necesario escribir el nombre de la función y enseguida un par de paréntesis, que estarán vacíos si no llevan parámetros y tendrán información en caso de ser funciones con paso de parámetros.

Para ejecutar un programa que contiene funciones, el compilador inicia con la primera instrucción del programa principal (*main*) y continúa ejecutándolo según la estructura de control del programa; cuando encuentra la llamada a una función éste se traslada y ejecuta el cuerpo de la función; cuando termina el compilador regresa a la siguiente instrucción que llamó a la función y continua de la misma manera.

Ubicación de la función.

Las funciones pueden ir antes o después del programa principal, pero si se ubican después es necesario escribir el prototipo de la función (encabezado) antes del programa principal. Una función puede que regrese o no regrese nada (void). En el primer caso se comporta como una variable, ya que la podemos imprimir, asignar a otra variable, comparar, utilizar como parámetro, etc. En el segundo caso, el resultado lo tenemos que imprimir dentro de la función (sólo las variables globales se imprimirán fuera).Existen dos tipos de funciones: Sin paso de parámetros y con paso de parámetros. (CORONA NAKAMURA & ANCONA VALDEZ, 2011)

Sintaxis de funciones sin parámetros y sin prototipo.

```
# librerías
Variables globales;
Constantes

void funcion_A ( )
{
        Variables locales;
        Instrucción 1;
        Instrucción 2;
}
void funcion_B ( )
{
        Variables locales;
        Instrucción 1;
        Instrucción 2;
}
```

```
main ( )
{
        Variables locales;
        Cuerpo del programa principal
        ..................................
        funcion_A( );
        funcion_B ( );
        Instrucción 3;
        ..................................
}
```

Prototipo de una función.

Un prototipo es el encabezado de una función, esto es, la primera línea de la función. Si utiliza prototipo las funciones pueden estar en cualquier lugar y en cualquier orden y pueden ser llamadas desde cualquier punto del programa principal o de otras funciones.

Un prototipo de función le da información importante al compilador. En el prototipo se indica el tipo de dato que regresa la función y el número, tipo y orden de los parámetros que recibe la misma. El compilador utiliza los prototipos para verificar las llamadas a funciones. Por ejemplo:

float promedio(int, float, int); o bien float promedio(int x, float y, int z);

Este prototipo indica que la función de nombre promedio retorna como resultado un valor de tipo real. Además informa que la función debe ser llamada con tres parámetros del tipo entero, real y entero. Como se puede observar el nombre de los parámetros es opcional.

Sintaxis de funciones sin parámetros y con prototipo.

```
# librerías
Variables globales;
Constantes

funcion_A ( );                  //prototipo de la función A
funcion_B ( );                  //prototipo de la función B

main ( )
{
    Variables locales;
        Cuerpo del programa principal
        ..................................
        funcion_A( );
```

```
        funcion_B ( );
        Instrucción 3;
        ..................................
        ..................................
}

void funcion_A ( )
{
        Variables locales;
        Instrucción 1;
        Instrucción 2;
        Instrucción 3;
}
void funcion_B ( )
{
        Variables locales;
        Instrucción 1;
        Instrucción 2;
}
```

La desventaja de las funciones sin prototipos, es que en el momento de llamarlas o invocarlas tienen que estar declaradas antes.

Un prototipo sin paso de parámetros es el encabezado de la función y a diferencia de la declaración de una función, ésta lleva punto y coma. La ventaja es que le avisa al compilador que existe una función que regresa un determinado tipo de dato y puede ser utilizada en cualquier momento, por lo que la declaración de las funciones no tiene que ser antes de la llamada sino después de la función principal.

Programa 1. Hacer un programa con un menú de tres opciones:

a) Hipotenusa de un triángulo rectángulo.
b) Área de un triángulo.
c) El tercer ángulo de un triángulo.

Utilizando funciones sin parámetros y sin prototipo.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

void hipotenusa ()
{
    int cat_op, cat_ady;
    float hipo;
```

```
    printf ("Introduce la longitud del cateto adyacente: ");
    scanf("%d",&cat_ady);
    printf ("Introduce la longitud del cateto opuesto: ");
    scanf("%d",&cat_op);
    hipo = sqrt( cat_op * cat_op + cat_ady * cat_ady);
    printf (" la hipotenusa del triangulo mide: %f",hipo);
}
void area()
{
    int base, h; float a;
    printf ("Introduce la base y la altura: ");
    scanf("%d%d",&base,&h);
    a = base * h/2;
    printf (" El area del triangulo es : %f",a);
}
void angulo()
{
    float ang1, ang2, ang3;
    printf ("Escriba el valor de los dos angulos: ");
    scanf("%f%f", &ang1, &ang2);
    ang3 = 180 - (ang1 +ang2);
    printf ("El tercer angulo mide: %f", ang3);
}
main ()
{
    char opc;
    printf ("a) Hipotenusa de un triangulo rectangulo \n ");
    printf ("b) Area de un triangulo  \n ");
    printf ("c) Tercer angulo de un triangulo \n \n");
    printf ("Elige una opcion \n\n ");
    scanf("%c", &opc);
    switch(opc)
    {
            case 'a': hipotenusa();
                    break;
            case 'b': area();
                    break;
            case 'c': angulo();
                    break;
    }
    getch(); return 0;
}
```

La ejecución del programa 1 lo puede observar en la figura 6.10.

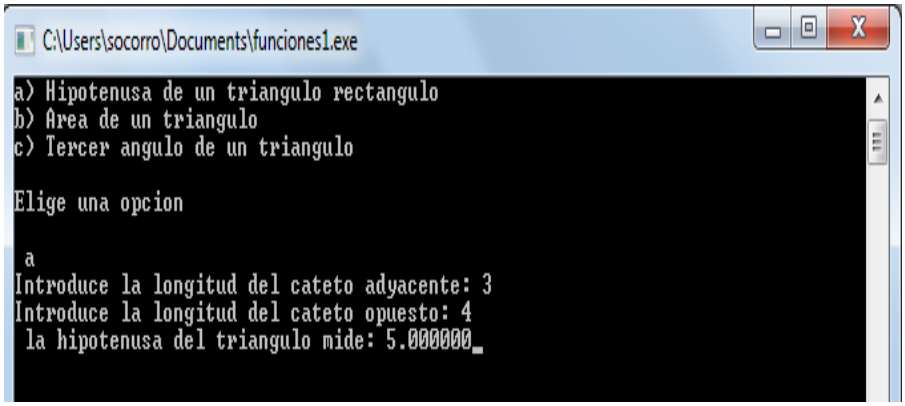


Figura 6.10. Ejecución de programa con funciones.

Programa 2. Hacer una función que lea 2 valores y regrese la suma de éstos.

```
#include <stdio.h>
#include <conio.h>

float suma ()
{
    float n1, n2, result;
    printf ("Introduce el primer numero: ");
    scanf("%f",&n1);
    printf ("Introduce el segundo numero: ");
    scanf("%f",&n2);
    result = n1 + n2;
    return result;
}

main()
{
    float res;
    res = suma();
    printf ("El resultado es %.2f", res);
    getch();
}
```

La ejecución del programa 2, la puede observar en la figura 6.11.

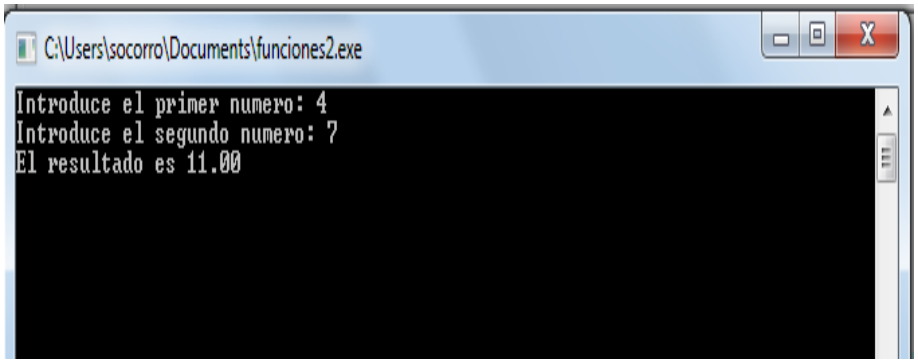


Figura 6.11. Función que realiza suma.

Programa 3. Hacer un programa que, tenga una función que calcule el área de un trapecio. Utilizando prototipo.

```
#include <stdio.h>
#include <conio.h>

float area();
float sumabases();

main()
{
    printf(" El area es %.2f", area());
    getch();
}
float area()
{
     float h;
     printf ("Escribe la altura ");
     scanf("%f",&h);
     return sumabases()* h/2;
}

float sumabases()
{
     float bm, BM;
     printf ("Introduce la base menor ");
     scanf("%f",&bm);
     printf ("Introduce la base mayor ");
     scanf("%f",&BM);
     return (bm + BM);
}
```

La ejecución del programa 3 con funciones y prototipo lo puede observar en la figura 6.12.

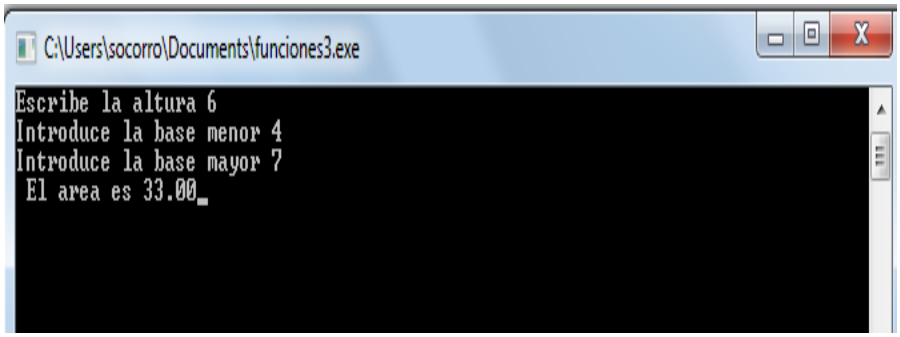


Figura 6.12. Ejecución de programa con funciones y prototipo.

Programa 4. Cal cular el cuadrado de los 10 primeros números naturales.

```
#include <stdio.h>
#include <conio.h>

int cuadrado();

int i;

main()
{
     int  cuad;
     for (i=1; i<=10; i++)
     {
         cuad = cuadrado();
         printf("\n El cuadrado de %d es: %d", i, cuad);
     }getch();
}

int cuadrado()
{
   return (i*i);
}
```

La ejecución del programa 4 con función y prototipo puede observarla en la figura 6.13.

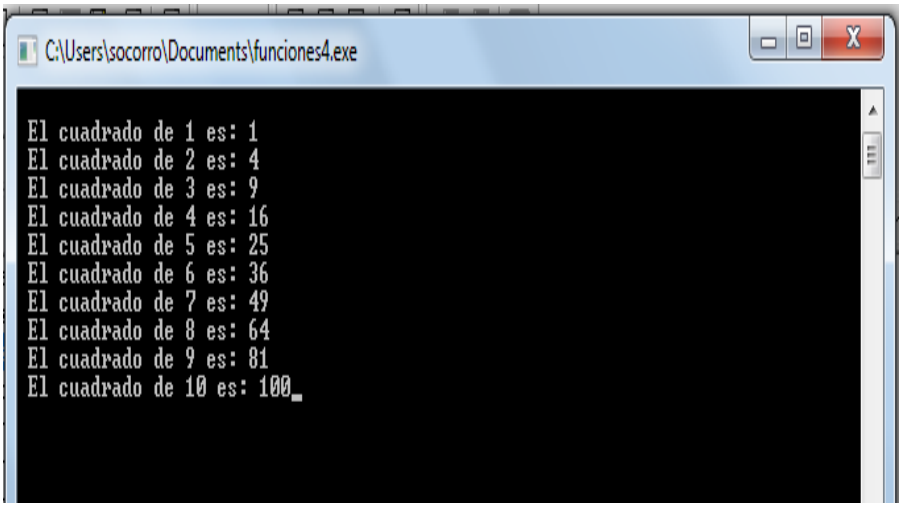


Figura 6.13. Función que calcula el cuadrado.

6.5 Pase por valor.

Funciones con paso de parámetros.

Hasta ahora, el diseño de nuestras funciones ha hecho que *main* simplemente las invoque, ya que con sólo llamarlas piden directamente al usuario los valores de entrada requeridos para trabajar. Pero existen otro tipo de funciones donde hay una comunicación más estrecha en el contexto de un programa modular: las funciones con paso de parámetros. (MÁRQUEZ FRAUSTO, OSORIO ÁNGEL, & OLVERA PÉREZ, 2011)

Una función con paso de parámetros es aquella que además de ser invocada requiere información por parte del subprograma que la llama. Esta información se refiere a los datos de entrada que se necesitan para que la función trabaje. (MÁRQUEZ FRAUSTO, OSORIO ÁNGEL, & OLVERA PÉREZ, 2011)

Hay dos formas de pasar datos a las funciones: paso por valor y por referencia. Estas funciones pueden tener uno o varios parámetros que pueden entregarse sólo por valor, sólo por referencia o de ambas formas. Los parámetros pueden ser de un mismo tipo de datos o de tipos de datos diferentes, según se requieran en el diseño de la función: por ejemplo, una función puede recibir únicamente datos enteros, si así se requiere, recibir datos de tipo entero, real o carácter.

Los **parámetros por valor** permiten pasar datos entre el programa principal y las funciones, e incluso entre las mismas funciones. En el parámetro se escribe una copia de la variable original. Si el parámetro sufre una alteración en la función que lo recibe, la variable original no se ve afectada. (CAIRÓ BATTISTUTTI, 2006)
La sintaxis para funciones con parámetros de valor es la siguiente:

```
tipo_dato identificador_funcion (tipo_dato_parámetro)
{
        declaración de variables locales;
        cuerpo de la función;
        return (valor);
}
```

Donde:

Tipo_dato: es el tipo de dato de la función, es decir el valor que regresa la función identificador_funcion es el nombre de la función.

Tipo_dato_parámetro: Variables (parámetros) antecedidas por su tipo de dato donde la función recibe los valores que requiere para trabajar por parte de la función que la invoca. Puede ser uno o varios.

Programa1. Hacer una función que reciba 2 valores (parámetros), realice una suma y regrese el resultado de la suma al programa principal.

```
#include <stdio.h>
#include <conio.h>

float suma1 (float a, float b)
{
    return a+b;
}
main()
{
    float num1, num2, res;
    printf("Introduce el primer valor ");
    scanf("%f", &num1);
    printf("Introduce el segundo valor ");
    scanf("%f", &num2);
    res =suma1(num1,num2);
    printf("El resultado de la suma es %.2f",res);
    getch();
}
```

Observe que en la función *suma1* se declaran dos parámetros de tipo real (*a y b*) los cuales son una especie de “molde” donde la función recibe los valores que necesita para trabajar.

En la función *main* se solicitan los datos al usuario y se almacenan en *num1* y *num2* respectivamente, y realiza la llamada a la función *suma1* con los valores almacenados en

num1 y *num2* (los contenidos de *num1* y *num2*, entran a la función a través de los parámetros *a* y *b* en el orden en que se enviaron, es decir, el valor de *num1* entra a través de *a* y *num2* entra a través de *b*). Se almacena la suma en *res*; *suma1* devuelve el valor de la variable *res* a *main*. Finalmente *main* imprime el resultado.

La ejecución del programa 1 lo puede ver en la figura 6.14.

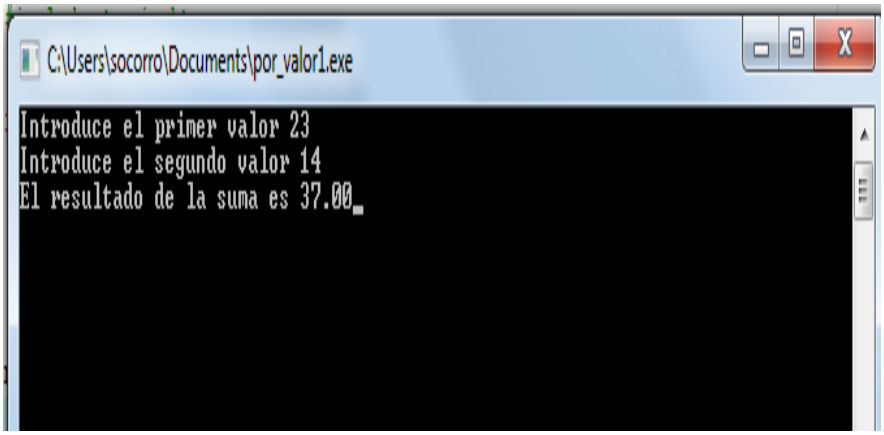


Figura 6.14. Función con parámetros.

Programa 2. El mismo programa anterior pero con algunas modificaciones en el *main* para que el programa calcule la suma de cuatro números.

```
#include <stdio.h>
#include <conio.h>

float suma1 (float a, float b)
{
    return a+b;
}

main()
{
    float num1, num2, num3, num4, res;
    printf("Introduce el primer valor ");
    scanf("%f", &num1);
    printf("Introduce el segundo valor ");
    scanf("%f", &num2);
    printf("Introduce el tercer valor ");
    scanf("%f", &num3);
    printf("Introduce el cuarto valor ");
    scanf("%f", &num4);

    res =suma1 (suma1(num1,num2),suma1(num3,num4));
```

```
    printf("El resultado de la suma es %.2f",res);
    getch();
}
```

La función *main* solicita cuatro valores y una vez que los tiene almacenados hace una llamada a *suma1*, pero en esta ocasión los argumentos de la primera llamada a *suma1* son llamadas a la misma función, es decir, en una instrucción se invoca o llama en tres ocasiones a *suma1*.

La primera en ejecutarse es la llamada a *suma1 (num1, num2)*, la cual devuelve un valor; la segunda función en ejecutarse es *suma1 (num3, num4)* y regresa un segundo valor. Estos dos valores son tomados como argumentos de la función *suma1 (suma1 (num1, num2), suma1 (num3, num 4))*, y todo se almacena en *res* para después imprimirse. Es importante mencionar que las llamadas a las funciones se van ejecutando de adentro hacia afuera.

La ejecución del programa 2 puede verla en la figura 6.15.

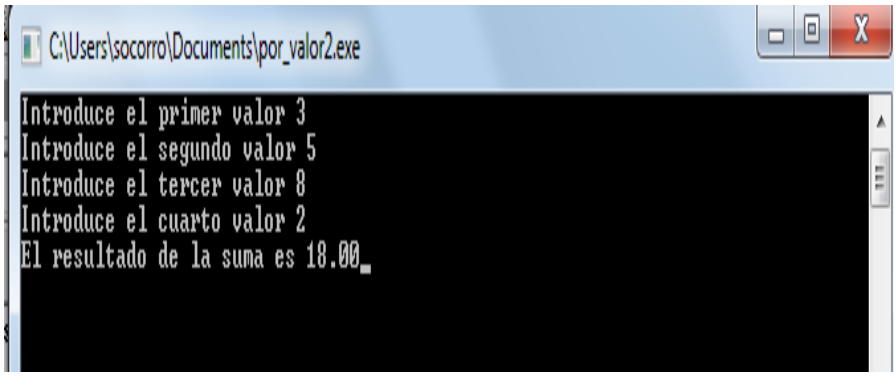


Figura 6.15. Función suma de cuatro números.

Programa 3. Hacer un programa con dos funciones: una que calcule el cuadrado de un número y otra que calcule el cubo. Ambas deben devolver el resultado.

```
#include <stdio.h>
#include <stdlib.h>

int cuadrado (int x)
{
   return (x * x);
}

int cubo (int y)
{
   return cuadrado(y) * y;
}
```

```
main()
{
    int opc, n;
    do{
      printf("Elige una opcion\n 1) CUADRADO\n 2) CUBO\n 3)SALIDA DEL
PROGRAMA\n");
      scanf("%i", &opc);
      switch(opc)
      {
          case 1: printf("Escribe el numero a elevar ");
                scanf("%i", &n);
                printf("%d al cuadrado es %d\n\n", n, cuadrado(n));
                system ("pause");
                break;
          case 2: printf("Escribe el numero a elevar ");
                scanf("%i", &n);
                printf("%d al cubo es %d\n\n", n, cubo(n));
                system ("pause");
                break;
          case 3: break;
          default: printf ("ERROR\n\n");
                 system ("pause");
      }
    }while (opc != 3);

}
```

Si en el *main* se selecciona la opción 1, se ejecutará *cuadrado*, la cual recibe un valor, lo multiplica por sí mismo y devuelve el resultado a *main*. En caso de que sea seleccionada la opción 2, se llamará a *cubo*, ésta a su vez llama a *cuadrado*, el cual recibe el resultado, lo multiplica por el parámetro y devuelve el *cubo* a *main*. La ejecución del programa termina hasta que selecciona la opción 3.

La ejecución del programa 3 lo puede ver en la figura 6.16.

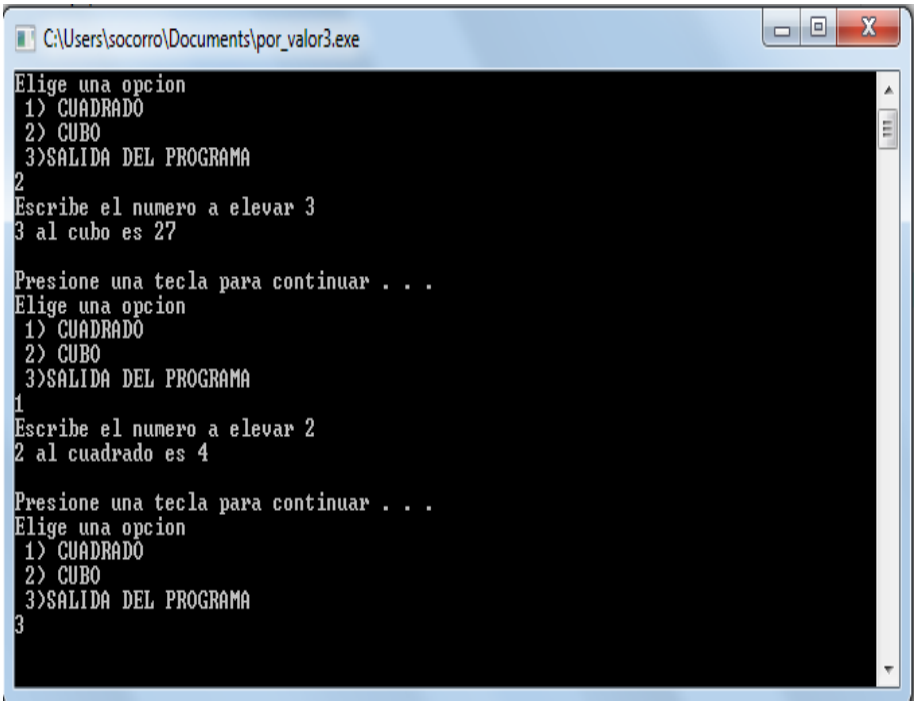


Figura 6.16. Programa con dos funciones.

6.6 Pase 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**. (JOYANES AGUILAR, 2005)

Los **parámetros por referencia** también permiten la comunicación entre el programa principal y las funciones, o entre las mismas funciones. Sin embargo, en este caso, en lugar de escribir una copia de la variable en el parámetro se escribe la dirección de la misma. Si el parámetro sufre una alteración en la función que lo recibe, la variable original también se ve afectada. En **C**, las llamadas *por referencia* se realizan mediante **apuntadores**. Un apuntador es una variable que contiene la dirección de otra variable y se representa por medio de los operadores de dirección (**&**) e indirección (*****). (CAIRÓ BATTISTUTTI, 2006)

Programa 1. Realizar una función que intercambie dos valores entre sí, utilizando parámetros por referencia.

```
#include <stdio.h>
#include <stdlib.h>

void intercambio (int *x, int *y)
```

```
{
   int aux;
   aux = *x;
   *x = *y;
   *y = aux;
}

main()
{
     int n1, n2;
     printf("Introduce el valor de n1 ");
     scanf("%d", &n1);
     printf("Introduce el valor de n2 ");
     scanf("%d", &n2);
     intercambio(&n1, &n2);
     printf("\nLos nuevos valores son n1 = %d y n2 = %d\n\n", n1,n2);
     system ("pause");
}
```

La función *main* solicita dos valores y los recibe en *n1* y *n2*. Suponga que el usuario decide teclear 9 y 10, *main* pasa como argumentos a *n1* y *n2* a la función de intercambio, la cual al ejecutarse recibe el 9 a través de *x* y el 10 a través de *y*, y los intercambia. Al terminar de ejecutarse la función intercambio, *main* imprime los valores de *n1* y *n2*. La ejecución en pantalla se vería como lo muestra la figura 6.17.

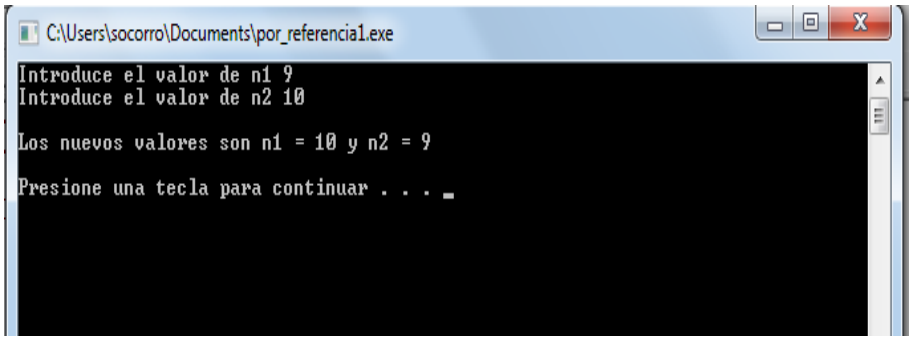


Figura 6.17. Parámetros por referencia.

Programa 2. Hacer un programa que calcule el cubo de un número utilizando una función y parámetro de referencia.

```
#include <stdio.h>
#include <stdlib.h>

void cubo (int *num);
```

```
int main()
{
    int n;
    printf ("Introduce un numero: ");
    scanf("%d",&n);
    cubo (&n);
    printf ("El cubo es %d\n", n);
    system("pause");
    return 0;
}

void cubo(int *num)
{
     *num = *num * *num * *num;
}
```

La ejecución del programa 2 la puede observar en la figura 6.18.

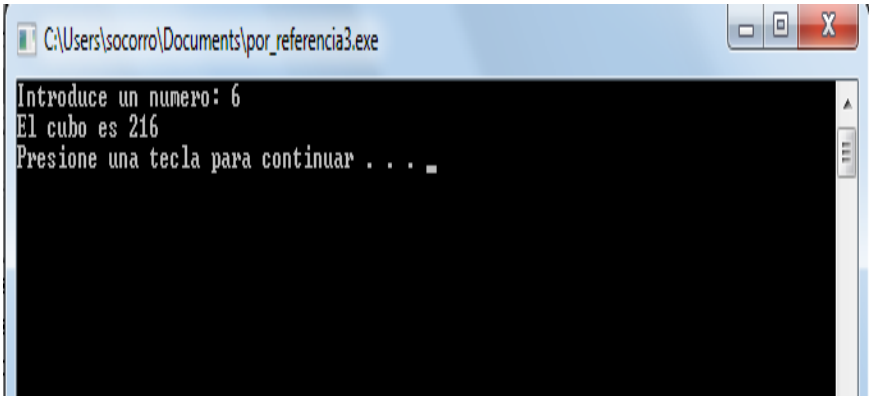


Figura 6.18. Cálculo del cubo.

Programa 3. Hacer un programa que convierta una cadena de caracteres en minúsculas a mayúsculas.

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>

void mayusculas (char *cad);

int main()
{
    char cadena[]= "instituto tecnologico de toluca";
    printf("la cadena antes de la conversion es: %s\n\n", cadena);
```

```
    mayusculas(cadena);
    printf("la cadena despues de la conversion es: %s\n\n", cadena);
    system ("pause");
    return 0;
}
void mayusculas (char *cad)
{
     while (*cad != '\0')
     {
          if (islower( *cad))
          {
             *cad = toupper(*cad);
          }
          ++cad;
     }
}
```

La ejecución del programa la puede ver en la figura 6.19.

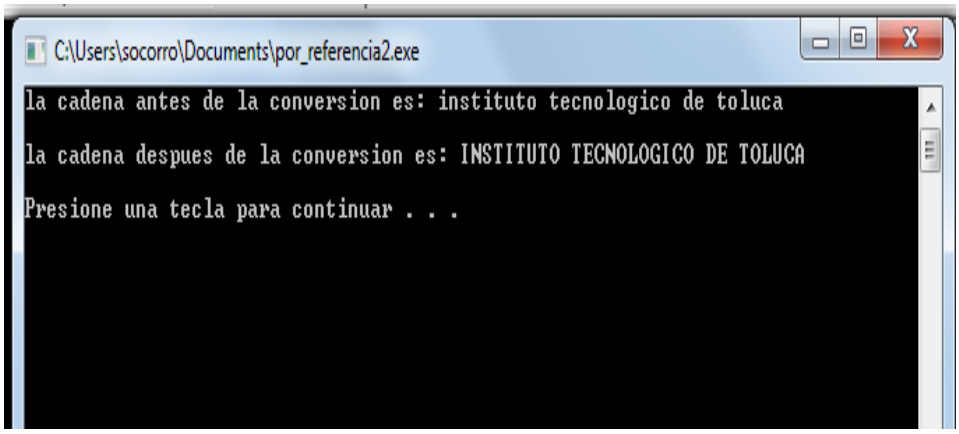


Figura 6.19. Conversión de minúsculas a mayúsculas.

Programa 4. Hacer un programa que, al recibir N número enteros, determine cuántos de éstos son pares y cuántos impares. Utilizando parámetros de valor y de referencia.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void par_e_impar (int NUM, int *P, int *I);

main()
{
```

```
    int i, N, NUM, PAR=0, IMP=0;
    printf("Ingresa el numero de datos: ");
    scanf("%d", &N);
    for (i=1; i <= N; i++)
    {
        printf("Ingresa el numero %d: ",i);
        scanf("%d", &NUM);
        par_e_impar(NUM, &PAR, &IMP);
    }
    printf("\nNumero de pares: %d\n", PAR);
    printf("\nNumero de impares: %d\n", IMP);
    system ("pause");
}
void par_e_impar (int NUM, int *P, int *I)
{
    double res;
    res = pow(-1,NUM);
    if (res > 0)
      *P += 1;
    else   *I += 1;
}
```

La ejecución del programa 4 la puede ver en la figura 6.20.

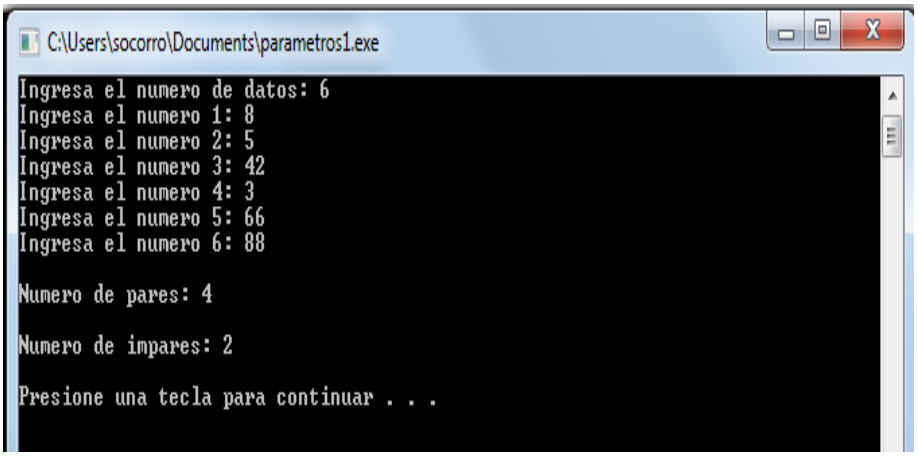


Figura 6.20. Cálculo de pares e impares.

6.7 Punteros.

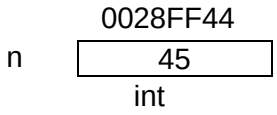
El puntero, no es más que una herramienta muy potente que puede utilizar en sus programas para hacerlos más eficientes y flexibles. Los punteros son, sin género de

dudas, una de las razones fundamentales para que el lenguaje C sea tan potente y tan utilizado.
Una **variable puntero** (o puntero, como se llama normalmente) es una variable que contiene direcciones de otras variables. Todas las variables vistas hasta el momento contienen valores de datos, por el contrario las variables punteros contienen valores que son direcciones de memoria donde se almacenan datos. (JOYANES AGUILAR, 2005)

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

int n = 45;

Asocia al nombre *n*, el tipo *int* y la dirección de alguna posición de memoria donde se almacena el valor de *n* (45).



La caja del ejemplo 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 de la caja. Si el valor de la variable se conoce, se representa en el interior de la caja.
Al valor de la variable se accede por medio de su nombre, a la dirección de la variable se accede mediante el operador de dirección &. Por ejemplo, se puede imprimir la dirección de n con la sentencia:

printf (“%p”, &n);

El operador de dirección “&” se aplica sobre el nombre de la variable para obtener sus direcciones.

El siguiente programa imprime el valor de una variable y su dirección.

```
#include <stdio.h>
#include <stdlib.h>

main()
{
    int n = 45;
    printf("n = %d\n", n);              /* visualiza el valor de n */
    printf ("&n = %p\n", &n);           /* visualiza la dirección de n */
    system ("pause");
}
```

En la figura 6.21 puede observar la ejecución del programa.

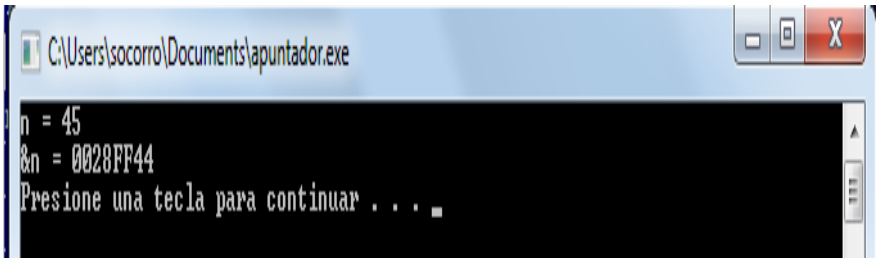


Figura 6.21. Impresión del valor y dirección de una variable.

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*. (JOYANES AGUILAR, 2005)

Una variable que se declara en C tiene una dirección asociada con ella. Un *puntero* o *apuntador* es una dirección de memoria.

La sintaxis para declarar un puntero es :

```
Tipo_apuntador  *  identificador_variable
```

Ejemplos:

```
int *p_entero;        // puntero a un tipo de dato entero (int)
char *c1;             // puntero a un tipo de dato carácter (char)
float *p_real;        //puntero a un tipo de dato real (float)
```

Se pueden asignar a un puntero direcciones de variables a través del operador unario & o direcciones almacenadas en otros punteros.

Por ejemplo:

```
int a = 37;
int *p, *q;
p = &a;        // Se le asigna a p la dirección de la variable a
q = p;         // Se le asigna a q la dirección almacenada en p (la misma de a)
```

Esto gráficamente quedaría como se muestra en la figura 6.22.

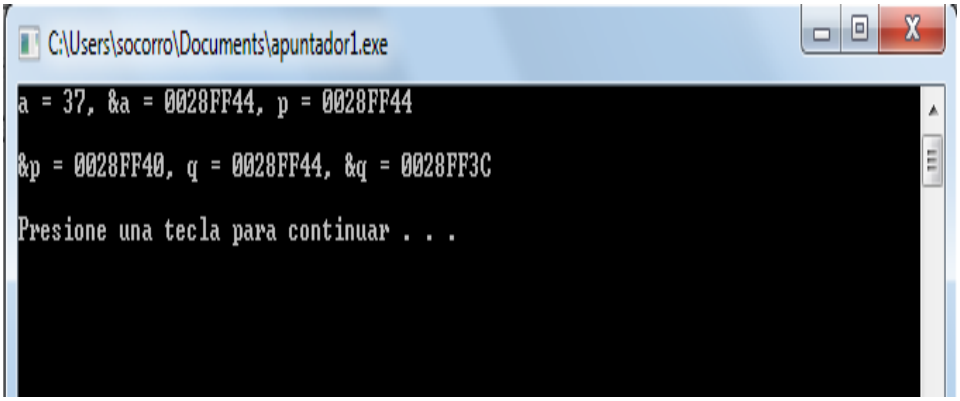


Figura 6.22. Asignaciones de punteros.

Para la obtención del valor almacenado en el espacio de memoria donde apunta un puntero utilizamos el operador unario *, aplicado al puntero que contiene la dirección del valor.

Para comprender mejor el concepto de punteros, observemos los siguientes programas con su explicación correspondiente. Luego de cada programa se presenta la ejecución del mismo para que pueda observar los valores que van tomando las variables.

Programa 1. Uso de punteros, variables y valores.

```
#include <stdio.h>
#include <conio.h>

main()
{
    int X = 3, Y = 7, Z[5] = { 2, 4, 6, 8, 10};
    printf ("DATOS ORIGINALES\n");
    printf("\nX=%d\tY=%d\nZ[0]=%d\tZ[1]=%d\tZ[2]=%d\tZ[3]=%d\tZ[4]=%d\n", X,Y,
    Z[0], Z[1], Z[2], Z[3], Z[4]);
    int *IX;            /* IX representa un puntero a un entero. */
    IX = &X;              /* IX apunta a X tiene la dirección de X. */
    Y = *IX;            /* Y toma el valor de X, ahora vale 3.*/
    *IX = 1;            /* X se modifica ahora vale 1. */
    printf("\nDATOS MODIFICADOS\n");
    printf("\nX=%d\tY=%d\nZ[0]=%d\tZ[1]=%d\tZ[2]=%d\tZ[3]=%d\tZ[4]=%d\n", X,Y,
    Z[0], Z[1], Z[2], Z[3], Z[4]);
    getch ();
}
```

Al ejecutar el programa podemos ver los datos modificados de X y de Y, como lo muestra la figura 6.23.

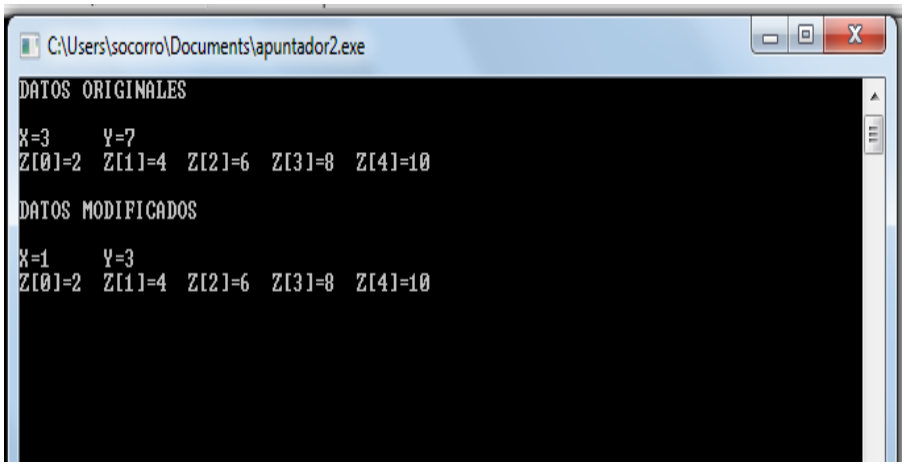


Figura 6.23. Uso de punteros, variables y valor.

Ahora agregamos las siguientes instrucciones:

```
IX = &Z[2];
Y = *IX;
*IX = 15;
```

En la primera instrucción IX apunta al tercer elemento del arreglo Z. En la segunda instrucción Y toma el valor de Z[2], ahora vale 6, y con la última instrucción Z[2] se modifica, ahora vale 15.

Las modificaciones a Y y Z[2] se verían como lo muestra la figura 6.24.

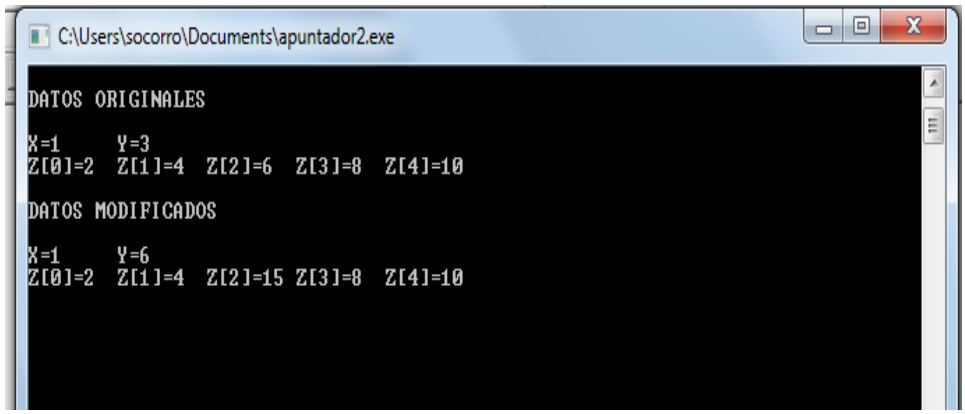


Figura 6.24. Modificaciones al programa anterior.

Nuevamente modificamos el programa 1 con las siguientes instrucciones:

```
X = *IX + 5;
*IX = *IX - 5;
```

De acuerdo con la primera línea X se modifica, ahora vale Z[2] + 5 = 20 (ya que *IX contiene el valor de Z[2]). También Z[2] se modifica, ahora vale 10 (*IX = *IX -5), como lo puede observar en la figura 6.25.

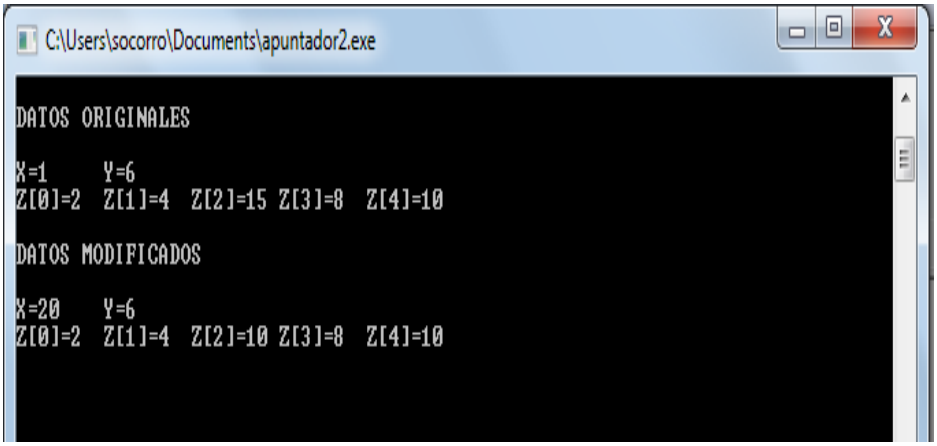


Figura 6.25. Continuación de uso de punteros.

Ahora agregaré dos instrucciones que se pueden utilizar para incrementar en 1.Ambas instrucciones se pueden utilizar para hacer exactamente lo mismo.

```
++*IX;
*IX += 1;
```

En la primera línea Z[2] se modifica, se incrementa en 1, ahora Z[2] vale 11. En la segunda línea se vuelve a modificar Z[2], incrementando su valor en 1, quedando su nuevo valor en 12, como lo puede observar en la figura 6.26.

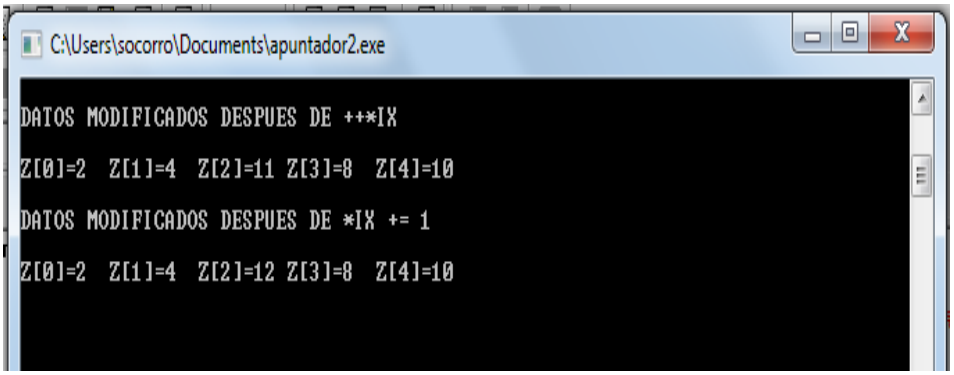


Figura 6.26. Incrementando el puntero.

Ahora desplazaremos el puntero una posición en el arreglo Z y haremos una nueva asignación.

```
X = *(IX + 1);
Y = *IX;
```

Con la instrucción X = *(IX + 1), X se modifica. El puntero IX se desplaza una posición y accede temporalmente a Z[3], por lo tanto X toma este valor (8). Observe que IX no se reasigna. En la segunda línea Y se modifica, toma el valor de Z[2], es decir 12.

En la figura 6.27 puede ver las modificaciones.

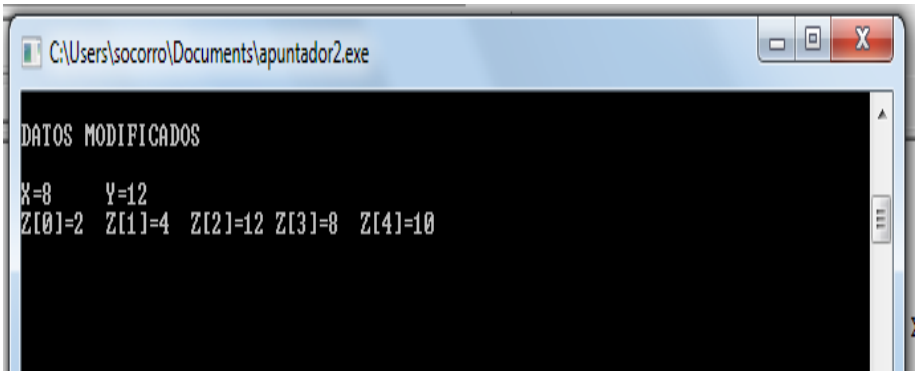


Figura 6.27. Modificando el puntero.

Otra forma de mover el puntero es con las siguientes instrucciones:

```
IX = IX + 1;
Y = *IX;
```

Con la primera instrucción IX se desplaza una posición, pero a diferencia del caso anterior, ahora se reasigna. IX apunta ahora al cuarto elemento de Z es decir a Z[3]. En la segunda instrucción Y se modifica tomando ahora el valor de Z[3], es decir 8, como lo puede observar en la figura 6.28.

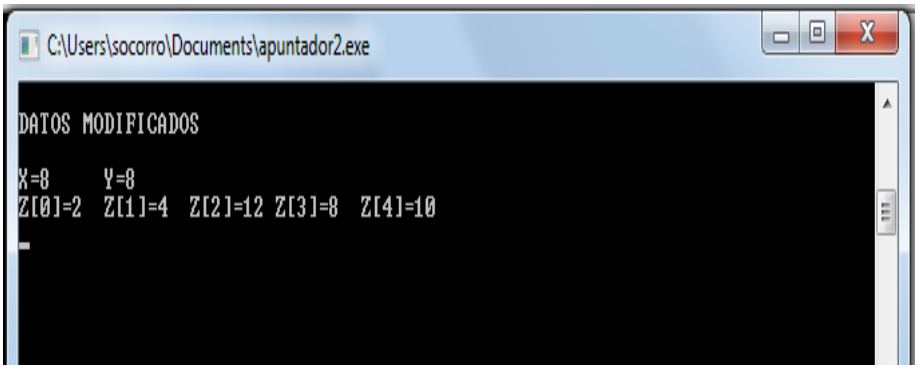


Figura 6.28. Desplazamiento del puntero.

Arreglos y funciones.

El lenguaje de programación C utiliza parámetros de referencia para pasar los arreglos a las funciones. Cualquier modificación que se realice a los arreglos en las funciones afecta su valor original. En la llamada a la función sólo se debe incluir el *nombre del arreglo*, que es un puntero. No se deben incluir los corchetes porque ocasionan un error de sintaxis. (CAIRÓ BATTISTUTTI, 2006)

Programa 2. Hacer un programa que lea dos vectores y realice la suma en un tercer vector. Utilizar los vectores como parámetros.

```
#include <stdio.h>
#include <conio.h>

void llenar_vector (int *x, int tam);
void imprime_vector (int *x, int tam);
void suma_vector (int x[20], int y[], int z[20], int t);

main()
{
     int t, A[20], B[20], C[20];
     printf("Total de elementos: ");
     scanf("%d", &t);
     printf("Escribe los elementos del vector A: \n");
     llenar_vector (A, t);
     printf("Escribe los elementos del vector B: \n");
     llenar_vector (B, t);
     suma_vector(A, B, C, t);
     printf("\nEl vector A es: \n");
     imprime_vector(A, t);
     printf("El vector B es: \n");
     imprime_vector(B, t);
     printf("La suma es : \n");
     imprime_vector(C, t);
     getch();
}
void llenar_vector (int x[], int tam)
{
    int i;
    for (i=0; i < tam; i++)
    {
        printf("Elemento %d:", i+1);
        scanf("%d", &x[i]);
    }
```

```
}
void imprime_vector (int x[], int tam)
{
    int i;
    for (i=0; i < tam; i++)
    {
      /* printf("Elemento %d: ", i+1);*/
       printf("%d\n", x[i]);
    }
}
void suma_vector (int x[20], int y[], int z[20], int t)
{
    int i;
    for (i=0; i < t; i++)
    {
       z[i]=x[i]+y[i];
    }
}
```

Como puede observar en los prototipos de las funciones, los parámetros para indicar que lo que recibe es un arreglo, se puede escribir x[] o *x.

La ejecución del programa lo puede ver en la figura 6.29

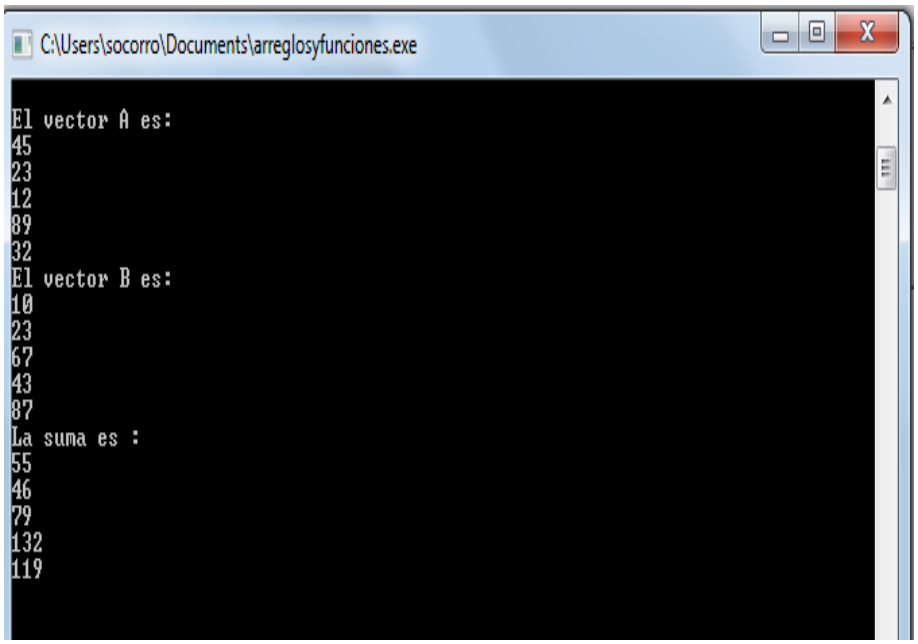


Figura 6.29. Vectores y funciones.

Programa 3. Hacer un programa que, al recibir un arreglo unidimensional de tipo entero que contiene calificaciones de exámenes de alumnos, calcule lo siguiente:

a) La media aritmética. Se calcula sumando los elementos del arreglo y dividiendo entre el número de elementos.
b) La varianza. Suma de los cuadrados de las desviaciones de la media, entre el número de elementos.
c) La desviación estándar. Se calcula como la raíz cuadrada de la varianza.
d) La moda. Se calcula obteniendo el número con mayor frecuencia.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
const int MAX =50;

void lectura(int *A, int T);
float media (int *A, int T);
float varianza (int *A, int T, float M);
float desviacion (float V);
void frecuencia (int *A, int P , int *B);
int moda (int *A, int T);

main()
{
    int TAM, MOD, alu[MAX], FRE[11] ={0};
    float MED, VAR, DES;
    do
    {
        printf("Ingrese el tamaño del arreglo ");
        scanf("%d", &TAM);
    }
    while (TAM > MAX || TAM < 1);
    lectura (alu, TAM);
    MED = media (alu, TAM);
    VAR = varianza (alu, TAM, MED);
    DES = desviacion (VAR);
    frecuencia (alu, TAM, FRE);
    MOD = moda (FRE, 11);
    printf("\nMedia:     %.2f", MED);
    printf("\nVarianza:     %.2f", VAR);
    printf("\nDesviacion:     %.2f", DES);
    printf("\nModa:       %d", MOD);
    getch();
```

```
}
void lectura (int A[], int T)
{
    int i;
    for (i=0; i < T; i++)
    {
        printf("Ingrese el elemento %d:", i+1);
        scanf("%d", &A[i]);
    }
}

float media (int A[], int T)
{
    int i;
    float sum =0.0;
    for (i=0; i < T; i++)
        sum += A[i];
    return (sum/T);
}
float varianza (int A[], int T, float M)
{
    int i;
    float sum = 0.0;
    for (i=0; i < T; i++)
        sum += pow((A[i] - M),2);
    return (sum/T);
}
float desviacion (float V)
{
     return (sqrt(V));
}
void frecuencia (int *A, int P , int *B)
{
    int i;
    for (i=0; i < P; i++)
        B[A[i]]++;
}
int moda (int A[], int T)
{
   int i, MOD=0, VAL = A[0];
   for (i=0; i < T; i++)
       if (MOD < A[i])
       {
          MOD = i;
```

```
        VAL = A[i];
      }
   return (MOD);
}
```

La ejecución del programa la puede observar en la figura 6.30.

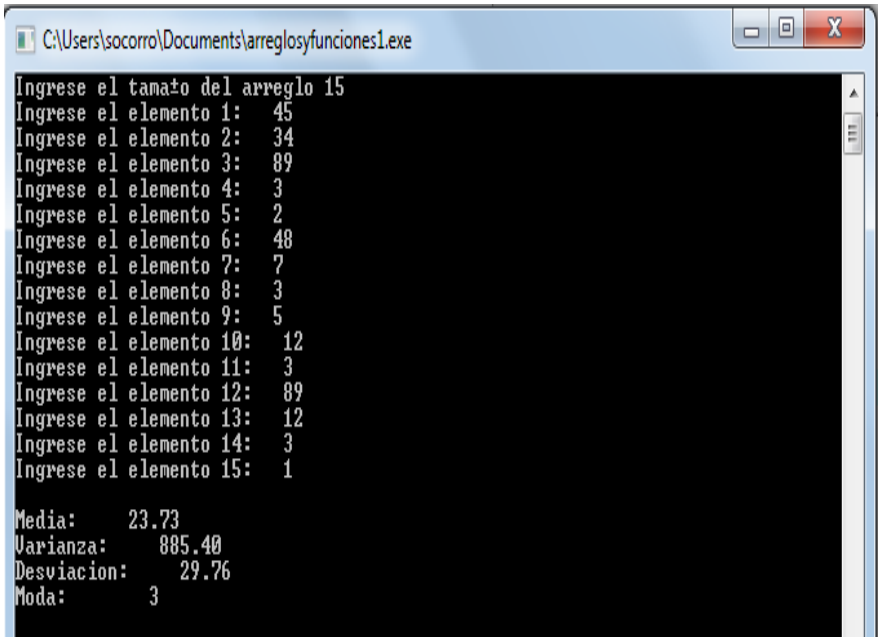


Figura 6.30. Vectores, funciones y parámetros.

Programa 4. Llenar un arreglo unidimensional con N elementos (máximo 50) y ordenarlo en forma ascendente.

```
#include <stdio.h>
#include <conio.h>

const int MAX =50;

void lectura (int *, int);
void ordena_burbuja (int *, int);
void imprime (int *, int);

main()
{
     int tam, vec[MAX];
     do
     {
        printf ("Ingrese el tamaño del arreglo ");
        scanf("%d",&tam);
```

```
    }while (tam > MAX || tam < 1);
    lectura (vec, tam);
    ordena_burbuja (vec, tam);
    imprime (vec, tam);
    getch();
}

void lectura (int vect[], int t)
{
  int i;
  for (i=0; i < t; i++)
    {
        printf("Ingrese el elemento %d:   ", i+1);
        scanf("%d", &vect[i]);
    }
}
void imprime (int vect[], int t)
{
  int i;
  for (i=0; i < t; i++)
        printf("\nvect[%d]: %d", i, vect[i]);
}
void ordena_burbuja(int vect[], int t)
{
    int i, j, temp;
    for (j=0; j < t-1; j++)
        for (i=0; i < t; i++)
            if (vect[i] > vect[i+1])
              {
                temp = vect[i];
                vect[i] = vect[i+1];
                vect[i+1] = temp;
              }
}
```

La ejecución del programa la puede observar en la figura 6.31.

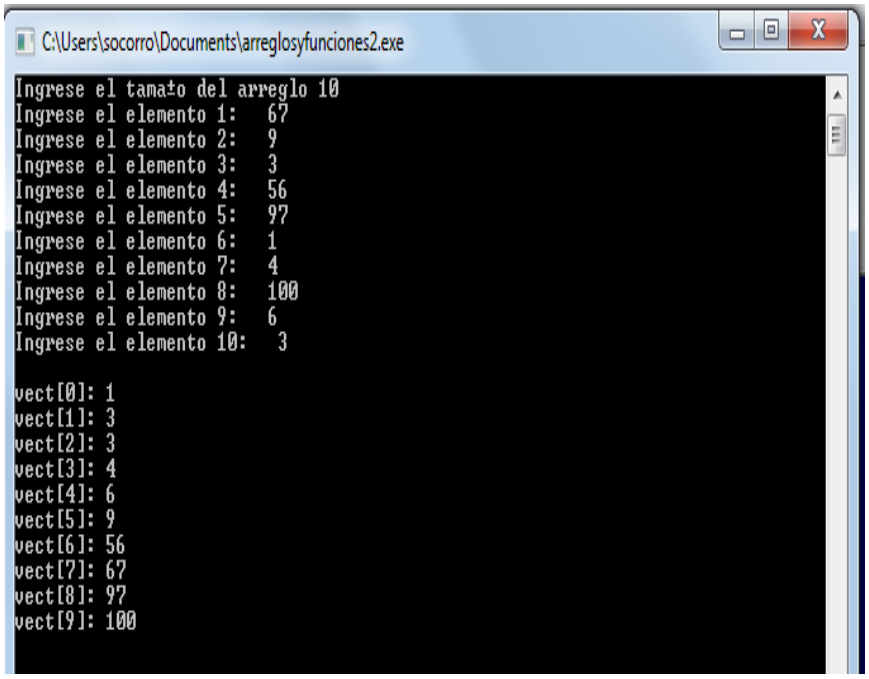


Figura 6.31. Método burbuja.

Programa 5. Hacer un programa que intercambie la N columnas de un arreglo bidimensional. Los elementos de la primera columna se intercambian con los de la última, los de la segunda con los de la penúltima, y así sucesivamente.

```
#include <stdio.h>
#include <conio.h>

const int MAX =50;

void lectura (float [][MAX], int, int);
void intercambio (float [][MAX], int, int);
void imprime (float [][MAX], int, int);

main()
{
    int f, c;
    float matriz[MAX][MAX];
    do
    {
        printf ("Ingrese el numero de filas ");
        scanf("%d",&f);
    }while (f > MAX || f < 1);
    do
    {
        printf ("Ingrese el numero de columnas ");
```

```
            scanf("%d",&c);
      }while (c > MAX || c < 1);
      lectura (matriz, f, c);
      printf("\n");
      printf ("Matriz original\n");
      imprime (matriz, f, c);
      intercambio (matriz, f, c);
      printf("\n");
      printf ("Matriz modificada\n");
      imprime (matriz, f, c);
      getch();
}
void lectura (float mat1[][MAX], int f, int c)
{
     int i, j;
     for (i=0; i < f; i++)
       for (j=0; j < c; j++)
       {
         printf("Ingrese el elemento %d %d:  ", i+1,j+1);
         scanf("%f", &mat1[i][j]);
       }
}
void intercambio (float mat1[][MAX], int f, int c)
{
   int i, j;
   float aux;
     for (i=0; i < (c/2); i++)
       for (j=0; j < f; j++)
       {
           aux = mat1[j][i];
           mat1[j][i] = mat1[j][c-i-1];
           mat1[j][c-i-1] = aux;
       }
}
void imprime (float mat1[][MAX], int f, int c)
{
   int i,j;
   for (i=0; i < f; i++)
   {
       for (j=0; j < c; j++)
         printf("%4.2f\t", mat1[i][j]);
       printf("\n");
   }
}
```

La ejecución del programa 5 los puede observar en las figuras 6.32 y 6.33.

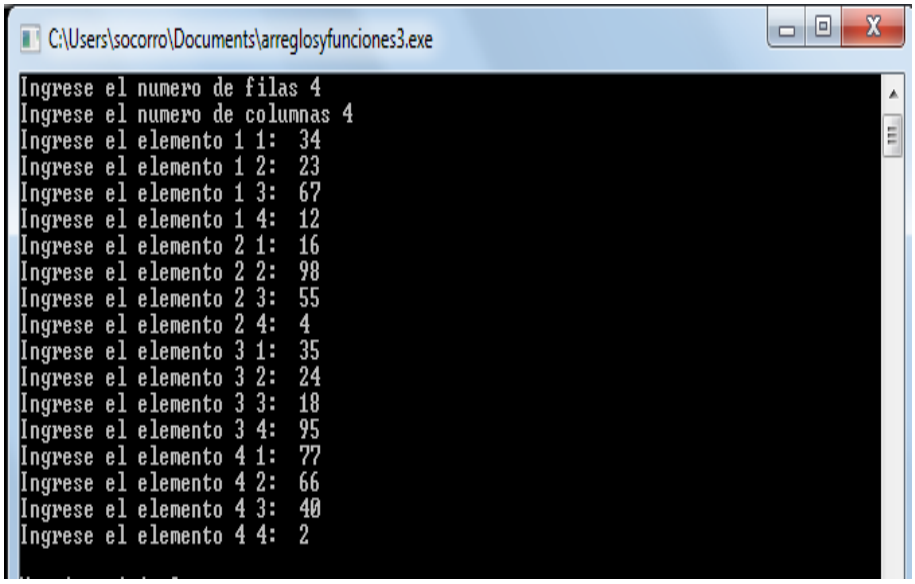


Figura 6.32. Matriz, funciones y parámetros.

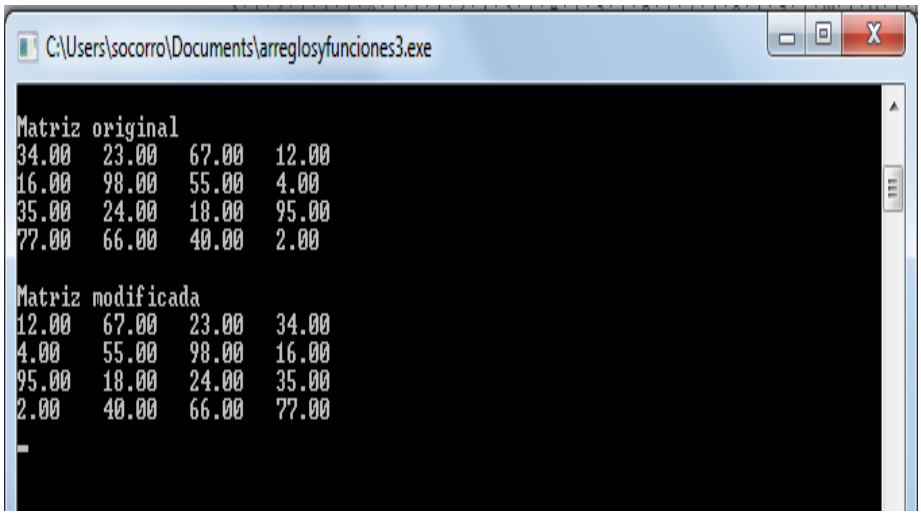


Figura 6.33. Continuación de la ejecución del programa.

Programa 6. Escribe un programa que, al recibir como dato un arreglo bidimensional cuadrado, determine si el mismo es simétrico. Se considera simétrico si cumple la condición mat[i][j] = mat[j][i].

```
#include <stdio.h>
#include <conio.h>

const int MAX =50;
```

```
void lectura (int [][MAX], int);
int simetrico (int [][MAX], int);

main()
{
     int matriz[MAX][MAX], n, r;
     do
     {
          printf ("Ingrese el tamaño del arreglo ");
          scanf("%d",&n);
     }while (n > MAX || n < 1);
     lectura (matriz, n);
     r = simetrico(matriz, n);
     if (r)
          printf("\nEl arreglo bidimensional es simetrico\n");
     else
           printf("\nEl arreglo bidimensional no es simetrico\n");
     getch();
}
void lectura (int mat[][MAX], int t)
{
    int i, j;
    for (i=0; i < t; i++)
      for (j=0; j < t; j++)
      {
        printf("Fila:%d\tColumna:%d  ", i+1,j+1);
        scanf("%f", &mat[i][j]);
      }
}
int simetrico (int mat[][MAX], int t)
{
   int i = 0, j,f = 1;
   while ((i < t) && f)
   {
        j = 0;
        while ((j < i) && f)
             if (mat[i][j] == mat[j][i])
                j++;
             else
                 f = 0;
        i++;
   }
   return (f);
}
```

La ejecución del programa 6 la puede observar en la figura 6.34

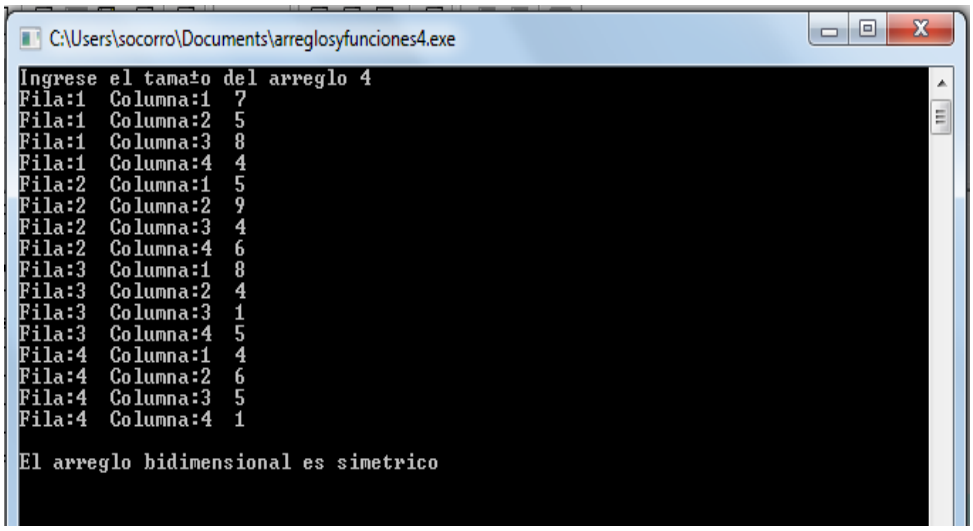


Figura 6.34. Determinación de matriz simétrica.

6.8 Elaboración de programas.

Funciones estándar.

1. Hacer un programa que tenga una función que reciba como entrada una cadena y devuelva el número de vocales, de consonantes y de dígitos de la cadena.
2. La función atoi() transforma una cadena formada por dígitos decimales en el equivalente número entero. Escribir una función que transforme una cadena formada por dígitos hexadecimales en un entero largo.
3. Hacer un programa que lea una cadena utilizando getchar() y cuente el número de ocasiones que aparece la letra ‘a’ en la cadena.
4. Hacer un programa que reciba como entrada un arreglo de caracteres y muestre la longitud de la cadena leída, carácter a carácter, con strlen() y con sizeof(). Finalmente imprimir la dirección de memoria del arreglo.
5. Leer una cadena de caracteres e imprimir las consonantes en mayúsculas y sus vocales en minúsculas.
6. Leer una cadena e imprimir la cantidad de palabras que tiene una frase.
7. Leer una cadena e imprimir la palabra de tamaño mayor.
8. Leer una cadena y sustituir los espacios en blanco por &.
9. Leer una cadena que representa un número romano menor o igual a 3000 y convertirlo a su equivalente decimal. Validar si es un número romano o no.
10. Leer el nombre y fecha de nacimiento de una persona y obtener su RFC.
11. Leer una secuencia de letras y mostrar la suma de sus códigos ASCII.

Funciones definidas por el usuario.

1. Hacer un programa que, tenga una función que calcule el promedio individual de n materias y devuelva ese valor a la función main() para que calcule el promedio grupal de n alumnos.
2. Un individuo invierte en un banco un capital específico y quiere saber cuánto obtendrá al cabo de cierto tiempo, si el dinero se coloca a una determinada tasa de interés mensual. Escriba el programa correspondiente.
3. Escriba un programa que lea un número entero N y calcule el resultado de la siguiente serie:
$$1 * \frac{1}{2} / \frac{1}{3} * \frac{1}{4} / \ldots (*, /) \frac{1}{N}$$
4. Escriba un programa que al recibir como dato un número entero N, obtenga el resultado de la siguiente serie:
$$1^1 - 2^2 + 3^3 - \cdots \pm N^N$$
5. Hacer un programa que lea una cantidad en grados Centígrados y la convierta a grados Fahrenheit.
6. Hacer un programa con un menú con opciones a cinco funciones que calcule diferentes volúmenes.

7. Hacer una función que calcule la potencia de X número a la N.

Funciones con parámetros de valor y referencia.

1. Hacer una función que reciba un número y calcule el factorial de dicho número. Utilice un parámetro de valor y uno de referencia.
2. Hacer una función que reciba un valor entero como parámetro de valor e imprima la tabla de multiplicar de dicho número.
3. Escriba un programa con una función que realice las siguientes operaciones con un número: raíz cuadrada, cuadrado y cubo. Mostrar el resultado en forma de tabla, con los números del 1 al 10. Utilizar parámetros de valor y referencia.
4. Realizar cuatro funciones para calcular el cuadrado de un número: una de ellas sin parámetros, otra que devuelva el resultado a la función principal, otra con parámetros por valor y que devuelva el resultado a la función principal, y finalmente otra con parámetros: uno de valor para el número y otro por referencia para el resultado.
5. Realizar una función que dibuje en la pantalla un marco con asteriscos. Utilizar parámetro de valor.
6. Realizar una función que reciba como parámetro de valor un número entero de cuatro dígitos, lo imprima en forma inversa como se muestra a continuación:
   Ejemplo: 4527
   7 2 5 4
7. Realice una función que reciba como parámetros dos números *x* y *n*, calcule la suma de la progresión geométrica siguiente:
$$1 + x + x^2 + x^3 + \cdots + x^n$$

8. Se dice que un número N es primo si los únicos enteros positivos que lo dividen son exactamente 1 y N. Escriba una función que al recibir como parámetro un número entero positivo, determine si éste es un número primo.
9. Escriba una función que tome como parámetros las longitudes de los tres lados de un triángulo (a, b, c) y devuelva el área del triángulo.

$$Area = \sqrt{p(p-a)(p-b)(p-c)}$$

Donde $p = \frac{a+b+c}{2}$

10. Dado el valor de un ángulo escribir una función que muestre el valor de tres funciones trigonométricas correspondientes al mismo. Utilizar parámetro de valor.

Funciones con arreglos.

1. Realizar una función que lea hasta 10 valores para un arreglo, enviándolo como parámetro, y calcule en otra función el promedio, el valor mayor, el valor menor y los muestre en la función principal.
2. Realice un programa que sume dos vectores de 20 elementos como máximo y almacene el resultado en un tercer vector utilizando funciones con paso de parámetros.
3. Realice un programa que sume dos matrices cuadradas de máximo 20 elementos y almacene el resultado en una tercera matriz utilizando funciones con paso de parámetros.
4. Leer un vector de enteros y un número entero x e imprimir la cantidad de:
   a) Veces que x aparece en el arreglo.
   b) Elementos menores a x en el arreglo.
   c) Elementos mayores a x en el arreglo.
   Utilizar funciones con paso de parámetros.
5. Leer un arreglo de enteros e imprimir la cantidad de valores del arreglo que son números primos. Utilizar una función que reciba un número entero y que decida si el número es primo. Utilizar parámetros.
6. Leer n datos para un vector de números reales, utilizando funciones para calcular la cantidad de: valores impares, valores pares y veces que se repite el valor máximo del arreglo. Utilizar parámetros.
7. Escriba una función que al recibir como parámetro una matriz, calcule su transpuesta.
8. Escriba una función que reciba una matriz cuadrada como parámetro y coloque un 1 en las diagonales principales. El resto se debe completar con ceros.
9. Escriba una función que reciba una matriz entera como parámetro, recorra este arreglo en forma de espiral.
10. Escriba una función que reciba una matriz entera como parámetro, recorra este arreglo columna a columna.

Bibliografía

CAIRÓ BATTISTUTTI, O. (2006). *FUNDAMENTOS DE PROGRAMACIÓN. PIENSA EN C.* MÉXICO: PEARSON-PRENTICE HALL.

CORONA NAKAMURA, M. A., & ANCONA VALDEZ, M. Á. (2011). *DISEÑO DE ALGORITMOS Y SU CODIFICACIÓN EN LENGUAJE C.* MÉXICO: MC GRAW-HILL/INTERAMERICANA EDITORES, S. A. DE C. V.

DEITEL, H. M., & DEITEL, P. J. (2004). *COMO PROGRAMAR EN C/C++ Y JAVA.* MÉXICO: PEARSON EDUCACIÓN DE MÉXICO.

FERNANDEZ, I. B. (2012). *INFORMÁTICA I.* SANTILLANA.

GONZÁLEZ, F. P. (2002). *DOMINE MICROSOFT OFFICE XP PROFESIONAL.* ESPAÑA: ALFAOMEGA.

GRUPO NORIEGA EDITORES. (2000). *MICROSOFT EXCEL 2000.* MÉXICO: LIMUSA S.A. DE C.V.

HENNEFELD, J. (1992). *TURBO PASCAL CON APLICACIONES 4.0-6.0.* MÉXICO: GRUPO EDITORIAL IBEROAMÉRICA.

JOYANES AGUILAR, L. (2005). *PROGRAMACIÓN EN C. METODOLOGÍA, ALGORITMOS Y ESTRUCTURA DE DATOS.* ESPAÑA: MC GRAW-HILL/INTERAMERICANA DE ESPAÑA, S. A. U.

MÁRQUEZ FRAUSTO, T. G., OSORIO ÁNGEL, S., & OLVERA PÉREZ, E. N. (2011). *INTRODUCCIÓN A LA PROGRAMACIÓN ESTRUCTURADA EN C.* MÉXICO: PEARSON.

NORTON, P. (1994). *INTRODUCCIÓN A LA COMPUTACIÓN.* MÉXICO: MC GRAW-HILL/INTERAMERICANA DE MÉXICO.

ORTIZ, M. (7 de JULIO de 2011). *EXCEL TOTAL*. Recuperado el 8 de FEBRERO de 2014, de www.exceltotal.com