

Programa de la asignatura:

Programación

U2

Introducción al lenguaje C



DCSBA



BIOTECNOLOGÍA



Índice

Presentación de la unidad	2
Propósitos	3
Competencia específica	3
2.1. Componentes de un programa y lenguajes de programación	4
2.1.1. Instrucciones.....	6
2.1.2. Comentarios	6
2.1.3. Estructura general de un programa.....	7
2.2. Tipos de datos	8
2.3. Variables, constantes y arreglos	10
2.3.1. Identificadores	11
2.3.2. Declaración e inicialización de variables	11
2.3.3. Declaración e inicialización de arreglos	12
2.3.4. Declaración de constantes	13
2.4. Expresiones matemáticas	13
2.4.1. Tipos de operadores.....	14
2.4.2. Evaluación de expresiones.....	16
2.5. Bibliotecas y funciones	18
2.5.1. Funciones matemáticas.....	19
2.5.2. Funciones de entrada y salida.....	20
2.6. Codificación de algoritmos	22
Actividades	29
Autorreflexiones.....	29
Cierre de la unidad	29
Fuentes de consulta	30



Presentación de la unidad

En la unidad anterior se presentó un conjunto de instrucciones que una ardilla (ficticia) podría realizar y se mostró cómo, a través de éstas, la ardilla podía resolver varios problemas siempre y cuando se definiera un algoritmo. Ahora veamos qué relación hay entre el mundo de la ardilla y las computadoras: la ardilla representa la computadora, que sólo puede ejecutar un conjunto de instrucciones definidas; además, al igual que la ardilla, la computadora por sí misma es incapaz de resolver ningún problema, para hacerlo necesita un programa que pueda seguir (ejecutar) para obtener el resultado deseado, así como la ardilla necesitaba un algoritmo que le indicara cómo realizar una tarea específica.

Por lo anterior, en esta unidad se mostrará la forma en que se crean programas, existen muchos lenguajes de programación, junto con tu figura académica decidirán en cual se trabajara durante la materia. En este documento los ejemplos se mostrarán en el lenguaje de programación C, desarrollado por Dennis M. Ritchie en el año de 1972, en los laboratorios Bell de la AT&T, y que posteriormente fue estandarizado por un comité del ANSI (por sus siglas en inglés *American National Standard Institute*) dando lugar al ANSI C, a través del cual se garantiza que cualquier programa creado bajo este estándar pueda ser ejecutado en cualquier computadora (Kernighan & Ritchie, 1991).

A lo largo de la unidad se introducen los elementos básicos del lenguaje de programación C, que en concepto son comunes a todos los lenguajes: tipos de datos, variables y constantes, expresiones aritméticas, funciones matemáticas y funciones de entrada y salida, con el fin de empezar a crear programas secuenciales que resuelvan problemas simples. Como primer tema analizaremos la estructura del programa “hola mundo”, pues como ya es costumbre en programación es el primer programa que se realiza cuando se aprende un nuevo lenguaje de programación. A partir de este ejemplo se revisa la estructura de un programa y los elementos que lo integran.



Propósitos



En esta unidad:

- Distinguirás la estructura de un programa.
- Declararás las variables y constantes que se utilizan en un algoritmo.
- Codificarás expresiones matemáticas.
- Utilizarás funciones de la biblioteca estándar para codificar instrucciones de entrada y salida y funciones matemáticas.
- Utilizarás un entorno de trabajo para escribir, compilar y ejecutar programas.

Competencia específica



Utilizar el lenguaje de programación C para resolver problemas a través de la implementación de algoritmos secuenciales.



2.1. Componentes de un programa y lenguajes de programación

En general, “un *programa* es una secuencia de instrucciones mediante las cuales se ejecutan diferentes acciones de acuerdo con los datos que se estén procesando (López, 2005)”.

En la unidad 1 se explicó que un programa sólo puede ser ejecutado por una computadora solamente si está escrito en lenguaje de máquina, pero existen lenguajes de programación, que son fáciles de entender para los seres humanos, mediante los cuales se pueden escribir programas más legibles conocidos como programas fuentes –en C los programas fuente tiene extensión .c–, que son traducidos a lenguaje de máquina mediante compiladores o traductores. En el caso de C es un lenguaje compilado, por lo que se genera un programa ejecutable con extensión .exe, que puede ser ejecutado cada vez que se desee sin necesidad de volver a compilar el programa fuente, a menos que se realice algún cambio.

De acuerdo con su creador un programa fuente en C, cualquiera que sea su tamaño, consta de funciones y variables. Una función contiene un conjunto de instrucciones, también llamadas preposiciones o enunciados, que especifican las operaciones que la computadora debe realizar; en tanto, las variables son los espacios de memoria donde se almacenan los valores utilizados en dichas operaciones (Kernighan & Ritchie, 1991, pág. 6).

Como se ha mencionado, el primer programa que se debe escribir cuando se aprende un nuevo lenguaje es “hola mundo”, así que para no perder la tradición a continuación se muestra cómo se implementa este programa en lenguaje C.

```
/*Directivas de preprocesador*/
#include<stdio.h>

/* Definición de función Principal
*/
main( )
{
    printf(“Hola mundo... \n”);
}
```

Programa 2.1: hola.c

El programa hola.c solo tiene una función: main; generalmente se puede dar cualquier nombre a las funciones que se definan en un programa, sin embargo, main es una función especial que siempre debe aparecer en algún lugar del programa, ya que es el punto desde el cual se inicia la ejecución (equivale a la instrucción de inicio de un algoritmo). Los paréntesis que aparecen después de la



palabra `main` indican que se trata de una función; las instrucciones que forman parte de ésta, llamadas cuerpo de la función, van encerradas entre llaves “{ }”, señalando el inicio y fin de la misma, respectivamente.

Las instrucciones que comienzan con “`/*`” y terminan con “`*/`”, se llaman comentarios e indican que todo lo que está escrito entre esos símbolos no son instrucciones que la computadora debe ejecutar sino información de interés para el programador; por ejemplo, la primera línea de código:

```
/*Directivas de preprocesador*/
```

Otro tipo de instrucciones especiales son las directivas del preprocesador, que son instrucciones que se realizan antes de la compilación del programa, un ejemplo es:

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

Se distinguen porque inician con el carácter gato “`#`”, en este caso esta instrucción le indica al compilador que debe incluir la información que está definida en el archivo de biblioteca `stdio.h`, en el cual se encuentran todas las funciones de salida y entrada, como `printf`.

Los compiladores de lenguaje C ofrecen distintas directivas, sin embargo, las que utilizaremos son:

- `#define`, que sirve para definir de constantes y/o macros
- `#include`, que se utiliza para incluir otros archivos

En el cuerpo de la función `main`, del programa 2.1, sólo aparece una instrucción que es la invocación a la función `printf` con el argumento “`Hola mundo... \n`”.

```
printf(“Hola mundo... \n”);
```

Invocamos o llamamos una función cuando requerimos que se ejecute con un conjunto de datos específicos, a los cuales llamamos argumentos. Una función se invoca o llama al nombrarla, es decir, escribiendo su nombre seguido de la lista de argumentos, separados por comas y encerrados entre paréntesis. En otras palabras, los argumentos son los valores que una función necesita para realizar la tarea que tiene encomendada, por ejemplo, la función `printf` tiene como fin imprimir la cadena de caracteres que recibe como parámetro, en este caso particular imprimirá la frase “`Hola mundo...`” seguida de un salto de línea, que es lo que representa la secuencia de caracteres “`\n`”.



2.1.1. Instrucciones

Una *instrucción o enunciado* en cualquier lenguaje se puede definir como una expresión que tiene alguna consecuencia, generalmente la consecuencia se ve reflejada en el cambio del valor que está almacenado en las variables. De acuerdo con su estructura sintáctica se pueden clasificar en dos tipos: *simples y compuestas*.

Las *instrucciones simples* se distinguen porque ejecutan una sola acción

Como ejemplo de instrucciones simples tenemos la declaración de variables, la llamada de funciones y la asignación.

```
int x;  
x = 2*y;  
printf("Hola");
```

En cambio, las *instrucciones compuestas* son aquellas que engloban más instrucciones o generan más de una acción. Un ejemplo de este tipo de instrucciones es el cuerpo de la función `main`, del programa 2.1.

2.1.2. Comentarios

Los comentarios son textos que sirven como información al programador y no son procesados por el compilador, es decir, no son instrucciones que debe realizar la computadora y por lo tanto no son traducidos a lenguaje de máquina. Para que un texto sea comentario en el lenguaje C y varios otros lenguajes debe estar entre los símbolos `/*` (marca el comienzo) y `*/` (marca el final de comentario). Pueden ir en cualquier parte del programa.

Un buen programador debe comentar sus programas para que otras personas puedan entender la lógica del programa, e incluso, los comentarios también le sirven al programador cuando en un tiempo futuro requiera realizar cambios. También es recomendable incluir al inicio del programa: el nombre del programa, el nombre del programador, una breve descripción de la tarea que realiza, las fechas de creación y de la última modificación, todo esto encerrado entre comentarios. Por ejemplo, al inicio del programa 2.1 sería conveniente incluir el siguiente comentario:

```
/*  
Programa: hola.c  
Programador: Pedro Pérez  
Descripción: El programa imprime la cadena "hola mundo"  
Fecha de creación: Agosto, 2010  
Última modificación: Septiembre, 2010
```



```
*/
```

Los comentarios también se pueden incluir al lado de una instrucción para describir de qué se trata, por ejemplo:

```
printf("Hola mundo"); /* Imprime el mensaje "Hola mundo" en la pantalla*/
```

2.1.3. Estructura general de un programa

Si observas el programa *hola.c*, la primera instrucción en el programa es la directiva al preprocesador, seguida de la función `main` y las instrucciones que la integran, esta estructura corresponde a los programas más simples, pero es similar a la de cualquier programa en C. La forma general de cualquier programa en C es la siguiente:

```
<instrucciones para el preprocesador>
<declaraciones globales>
<tipo_devuelto> main(<lista de parámetros>)
{
    <lista de instrucciones>
}
```

Y como se ha mencionado, se pueden incluir comentarios en cualquier parte del código. A continuación, se presenta un programa que calcula el área de una circunferencia dada la medida de su radio.

```
/* Directivas del procesador (bibliotecas) */
#include<stdio.h>

/* Declaraciones globales */
const float PI = 3.1416;

/* Función Principal */
main( )
{
    /* Declaraciones de variables locales la función main*/
    float radio, area
    printf(" Calcula el area de una circunferencia\n");
    printf(" Ingresa el radio: ");
    scanf("%f", &radio);
    area = PI * radio * radio; printf(" Area = %.2f",area);
    getchar();
}
```




```
        getchar();  
    }
```

Programa 2.2: areaCircunferencia.c

Por el momento, basta con que observes la estructura del programa, que se hace evidente con los comentarios, el significado de cada una de las líneas de código se irá definiendo en las siguientes secciones. Para cerrar esta sección en la siguiente figura se muestra la ejecución del programa con la entrada radio=3.

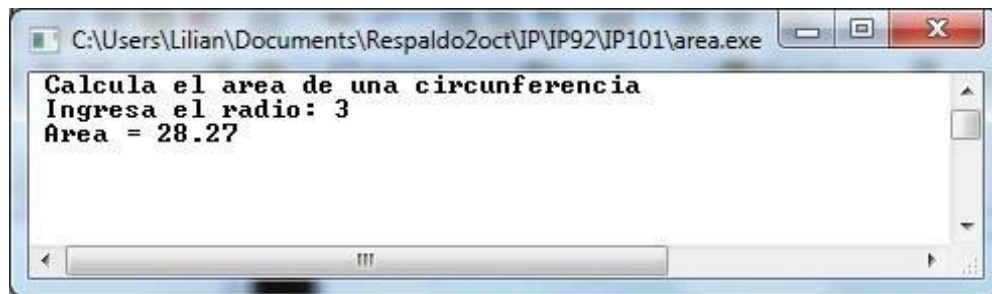


Figura 2.1: Ejecución del programa areaCircunferencia.c utilizando Dev-C++

2.2. Tipos de datos

En la Unidad 1 se mencionó que todos los datos que son procesados por una computadora se almacenan en la memoria principal y, no importa de qué tipo sean, se representan por medio de unos y ceros, sin embargo, no todos se pueden representar con el mismo número de bits¹, esto sí depende del tipo que se trate. Por ejemplo, los caracteres como: 'a', '@', 'Z', entre otros, se representan utilizando 8 bits, en cambio para representar un número decimal se utilizan, al menos, 32 bits; por lo que cuando se crea un espacio de memoria es necesario indicar qué tipo de dato se desea almacenar ahí, para que se reserve la cantidad de celdas de memoria que se necesitan y las operaciones que se pueden realizar con ellos.

En general, los tipos de datos que existen, independientemente de un lenguaje de programación, se pueden clasificar de la siguiente forma:

¹ Recuerda que un bit se definió como la unidad mínima de información, esto es, 1 ó 0.



Simples	TM Numéricos	<i>Enteros</i>
		<i>Decimales</i>
	TM Lógicos (verdadero o falso)	
	TM Alfanuméricos (caracteres)	
Estructurados	TM Arreglos	Unidimensionales
		Multidimensionales
	TM Estructuras o registros	

Tabla 2.1: Tipos de datos

En la siguiente tabla se muestran los tipos de datos simples en C:

Tipo	Descripción	Ejemplo
char	Su tamaño equivale a ocho bits, capaz de contener un carácter del conjunto de caracteres ASCII.	'a', 'C', '3'
int	Un entero, normalmente ocupa mínimo 16 bits, pero su tamaño es según la máquina en que se ejecute.	1024, -258
Float	Número de punto flotante de precisión normal.	10.5, -11.6
Double	Punto flotante de doble precisión	0.00045, -0.55236

Tabla 2.2: Tipos básicos en C

Es posible aumentar o disminuir la capacidad de representación de los números utilizando los modificadores long, short y unsigned. De esta manera las combinaciones que se ajustan al estándar ANSI, junto con sus rangos mínimos, son:



Tipo	Bits	Rango
Char	8	-127 a 127
unsigned char	8	0 a 255
signed char	8	-127 a 127
Int 16		-32767 a 32767
unsigned int	16	0 a 65535
signed int	16	-32767 a 32767
short int	16	-32767 a 32767
unsigned short int	16	0 a 65535
signed short int	16	-32767 a 32767
long int	32	-2147483647 a 2147483647
signed long int	32	-2147483647 a 2147483647
unsigned long int	32	0 a 4294967295 seis
Float	32	dígitos de precisión diez
Double	64	dígitos de precisión diez
long double	64	dígitos de precisión

Tabla 2.3: Tipos básicos y modificadores

Una vez que conocemos lo tipos de datos que se pueden representar, es conveniente saber cómo se reservan espacios de memoria donde son almacenados, esto es justo lo que se explica en el siguiente tema.

2.3. Variables, constantes y arreglos

Las variables y las constantes en los lenguajes de programación se utilizan para almacenar valores, la diferencia que existe entre ellas es que el valor almacenado en una variable puede ser modificado en cualquier instante del programa en tanto que las constantes no pueden modificarse.

Formalmente, una variable es un espacio de memoria que tiene asignado un nombre (también llamado identificador) y se utiliza para almacenar un valor que puede ser modificado durante la ejecución de un programa, a este valor que se encuentra almacenado en una variable en un momento dado se le llama estado de la variable. Por lo contrario, una constante es un dato cuyo valor se establecen en tiempo de compilación y no pueden cambiar durante la ejecución del programa. Existen dos tipos de constantes: literales y simbólicas. Las segundas, al igual que las variables, también tienen asignado un nombre.

A lo largo de esta sección descubrirás cómo puedes construir variables y constantes, así que lo primero será listar las reglas que debes seguir para nombrarlas.



2.3.1. Identificadores

Hay una serie de restricciones en cuanto a los nombres o identificadores, ya sea de variables, constantes o funciones. Éstas son:

- a) Los identificadores se integran por letras y dígitos, pero es necesario que el nombre siempre
- b) comience con una letra, por ejemplo: enteroA, arco3, S184.
- c) No puede contener caracteres especiales, por ejemplo, acentos (á,í), la letra eñe (Ñ), gato (#), guion (-). El carácter de subrayado “_” es el único carácter especial que puede utilizarse, generalmente se usa para darle una mejor legibilidad al nombre de una variable. Por ejemplo: entero_A, area_Circulo, i_elemento.
- d) El lenguaje C distingue entre letras mayúsculas y minúsculas, esto significa que los siguientes identificadores son distintos: area, Area, AREA.
- e) No pueden contener espacios en blanco.
- f) No pueden ser palabras reservadas.

Aunque no es una regla, se recomienda que los identificadores sean significativos, es decir, que el nombre indique qué dato se está almacenando ahí.

Ejemplo 2.1:

- 1) Se requiere una variable para almacenar el radio de un círculo
Las opciones sugeridas son: radio, radioCirculo, radio_circulo.
- 2) Se requiere un identificador para una variable que almacene el promedio de ventas anuales de una tienda departamental.
Opciones sugeridas: promedioVentas, prom_Ventas, promAnual.
- 3) Almacenar el número telefónico de una persona.
Opciones sugeridas: tel, telefono.

2.3.2. Declaración e inicialización de variables

La *declaración de una variable* es una instrucción que proporciona información de la variable al compilador, para que éste reserve en memoria el espacio adecuado y la referencia (identificador) para tener acceso a ella. Las declaraciones en general tienen la siguiente sintaxis:



```
<tipo><identificador>;
```

Donde **<tipo>** se refiere un tipo básico y el **<identificador>** se refiere al nombre con el cual se identificará el espacio de memoria reservado, puede ser cualquier nombre siempre y cuando se respeten las reglas vistas previamente. Veamos los siguientes ejemplos en C:

```
int edad;  
/* declara una variable de tipo entero con el identificador edad*/  
  
float area;  
/* declara una variable de tipo float a la cual identificará por area*/
```

También es posible declarar una lista de variables de un mismo tipo separando los nombres con comas “,”.

```
<tipo><identificador1>, ... , <identificadorN>;
```

Podemos ver un ejemplo de este tipo de declaración en el programa que calcula el área de una circunferencia (programa 2.2), cuando declaramos las variables locales a la función **main**

```
float radio, area;
```

Siempre es posible asignarle un valor inicial a las variables cuando las declaramos, a esta operación se conoce como inicialización. La sintaxis es:

```
<tipo><identificador>=<valor>;
```

Del mismo modo, podemos inicializar varias variables:

```
<tipo><identificador1> = <valor1>, ... , <identificadorN> = <valor2>;
```

Para ilustrar esto, se presentan las siguientes instrucciones:

```
int edad = 18;  
float radio = 3.0, area = 0.0;
```

2.3.3. Declaración e inicialización de arreglos

Un arreglo es una estructura de datos, o más técnicamente, un espacio de memoria que permite almacenar una colección de elementos, todos del mismo tipo. Conviene imaginar un arreglo como una secuencia contigua de celdas (espacios de memoria), o casillas, en cada una de las cuales se puede guardar un elemento de la colección.



Los lenguajes de programación permiten que el programador declare arreglos de cualquier tipo y prácticamente de cualquier tamaño. Problemas que requieren guardar una colección de datos como resultados de un examen, una base de teléfonos ocupa este tipo de estructuras.

Veamos como ejemplo la declaración del arreglo unidimensional planteado para las calificaciones del problema1. Inicializando sus elementos en la declaración queda como:

```
int lista[10] = {9,10,8,5,9,6,7,9,4,8};
```

Para referirse a un elemento del arreglo es necesario indicar el nombre del arreglo seguido del índice o índices correspondientes al elemento que deseamos acceder. Para ello se debe seguir la siguiente sintaxis.

Cada elemento del arreglo se puede tratar igual que a cualquier otra variable, es decir, podemos asignarle un valor, incluir en una expresión algebraica o lógica, imprimir en pantalla su valor, asignarle desde el teclado un valor, etc.

2.3.4. Declaración de constantes

Las constantes son expresiones con un significado invariable. Una de las formas que existe en C para declarar una constante simbólica es utilizando la directiva de preprocesador **#define**, la cual asocia un identificador a un valor constante, sin reservar espacio en memoria, por lo que no podemos decir que se declara, sólo se *define*. La sintaxis general es la siguiente:

```
#define<identificador><valor_constante>
```

Por ejemplo:

```
#define PI 3.1416
```

Se recomienda escribir el nombre de una constante con letras mayúsculas para diferenciarlas de las variables, pero las reglas son exactamente las mismas que para los identificadores de las variables.

2.4. Expresiones matemáticas

Una *expresión matemática* puede ser un número, una variable, una constante o la combinación de operadores y todas las anteriores. Toda expresión al ser evaluada produce un valor. Se dividen en dos tipos de acuerdo con el tipo de datos que devuelven cuando son evaluadas: *expresiones aritméticas* cuando el resultado de la evaluación es un número y *expresiones booleanas* cuando el resultado de la evaluación es un valor booleano (verdadero o falso). En este punto es importante destacar que el



modo en que el lenguaje C maneja los valores booleanos es por medio de valores enteros: *cero* equivale a *falso* y cualquier entero *distinto de cero* representa *verdadero*.

Las expresiones matemáticas permiten modelar situaciones reales, por ejemplo, mediante las expresiones aritméticas podemos modelar la forma de calcular el área de cualquier figura, también podemos representar la forma de calcular las raíces de un polinomio de segundo grado, o calcular el monto de una venta, etc. En cambio, las expresiones booleanas son la base para construir programas que pueden tomar decisiones. Veamos los siguientes ejemplos:

Ejemplo 2.2:

- a) La hipotenusa es igual a la raíz cuadrada de la suma de los cuadrados de catetos.

Codificación en C: `c=sqrt(a*a + b*b);`

- b) ¿x es un número par?

Sabemos que un número es par si es divisible entre 2, en otras palabras, si el residuo de la división entre dos es cero, lo cual se puede expresar con el operador de módulo, que devuelve el residuo de una división.

Codificación en C: `x % 2 == 0`

En el siguiente subtema se presentan los operadores básicos del lenguaje C, tanto aritméticos como booleanos.

2.4.1. Tipos de operadores

Los *operadores* son palabras o símbolos que hacen que permiten realizar operaciones con los datos de un programa, para cada tipo de datos hay una serie de operadores definidos.

Entre todos los operadores se distingue el operador de *asignación* "=", que se puede leer como "guarda un valor en la variable indicada", el valor puede ser una constante literal o el resultado de una expresión. Cabe señalar que este operador en pseudocódigo o diagrama de flujo lo hemos representado con una flecha apuntado hacia la izquierda ←.

Veamos algunos ejemplos:

```
/* modifica el estado de la variable radio con el valor 3.0*/  
radio = 3.0;
```

```
/* modifica el estado de la variable por el resultado de evaluar la expresión PI * radio * radio */
```




```
area = PI * radio *radio;
```

Los operadores *aritméticos* definidos en C son: “+” (suma), “-” (resta), “*” (multiplicación), “/” (división) y “%”(módulo). Este último representa el residuo de dividir dos números enteros, por ejemplo, si realizamos la división de 6 entre 15 (15/6), el cociente es 2 y el residuo es 3, al aplicar el operador módulo a estos valores tenemos:

```
15 % 6 → 3
```

```
21 % 4 → 1
```

En C también existen los operadores de *incremento* “++” y *decremento* “--”, éstos tienen el efecto de aumentar o decrementar en una unidad el valor de una variable, supongamos que estamos trabajando con la variable x:

```
x++;/* equivale a hacer: x = x + 1;*/  
x--;/* equivale a hacer: x = x - 1;*/
```

Por ejemplo, si el valor de x es 5 el resultado de aplicar ++y --es:

```
x++; → 6
```

```
x--; → 4
```

Los operadores que permiten construir expresiones booleanas son:

- **Operadores relacionales**, que manipulan expresiones aritméticas y son: “>” (mayor que), “<” (menor que), “>=” (mayor o igual), “<=” (menor o igual), “==” (igual), “!=” (distinto)
- **Operadores booleanos**, también llamados *lógicos*, manipulan únicamente expresiones booleanas y son: “!” (negación), “||” (disyunción) y “&&” (conjunción).

Observa que el operador de igualdad se escribe con dos símbolos de igualdad seguidos (==). El error más común es escribir una comparación con un sólo símbolo de igualdad, recuerda que (=) es el operador de asignación y su significado es totalmente distinto. En cuanto a los operadores *booleanos*, su significado es el siguiente:

Negación“!”, es un operador unario que cambia el valor de verdad de la expresión a la cual se le aplica. Por ejemplo, si el valor de verdad de expresión es verdadero entonces devuelve falso, y viceversa.

Por ejemplo, si x=2, y=3, z=5.



$$\begin{aligned}!(z > x) &\rightarrow !(5 > 2) \rightarrow !(1) \rightarrow 0 \text{ (falso)} \\!(x > y) &\rightarrow !(2 > 3) \rightarrow !(0) \rightarrow 1 \text{ (falso)}\end{aligned}$$

En pseudocódigo o diagrama de flujo se representa con la palabra en inglés NOT.

Conjunción“&&”, es un operador binario que se evalúa como verdadero sólo cuando las dos expresiones involucradas son verdaderas, en caso contrario devuelve falso.

Por ejemplo, si evaluamos las siguientes expresiones en el estado $x=2$, $y=3$, $z=5$.

$$\begin{aligned}(x > y) \ \&\& \ (z > y) &\rightarrow (2 > 3) \ \&\& \ (5 > 3) \rightarrow 0 \ \&\& \ 1 \rightarrow 0 \text{ (falso)} \\!(x > y) \ \&\& \ (z > y) &\rightarrow !(2 > 3) \ \&\& \ (5 > 3) \rightarrow ! \ (0) \ \&\& \ 1 \rightarrow 1 \ \&\& \ 1 \rightarrow 1 \text{ (verdadero)}\end{aligned}$$

En pseudocódigo y diagrama de flujo se representa con la palabra en inglés AND.

Disyunción“||”, también es un operador binario que devuelve únicamente devuelve falso si los dos operadores son falsos, en caso contrario devuelve verdadero. Nuevamente, tomemos el mismo estado de las variables $x=2$, $y=3$, $z=5$.

$$\begin{aligned}(x > y) \ || \ (z > y) &\rightarrow (2 > 3) \ || \ (5 > 3) &\rightarrow 0 \ || \ 1 \rightarrow 1 \text{ (verdadero)} \\(x > y) \ || \ (y > z) &\rightarrow (2 > 3) \ || \ (3 > 5) &\rightarrow 0 \ || \ 0 \rightarrow 0 \text{ (falso)}\end{aligned}$$

En pseudocódigo y diagrama de flujo se representa con la palabra en inglés OR.

Existen otro tipo de operadores, sin embargo, su estudio supera los objetivos de este curso por lo que no los revisaremos, si deseas saber más puedes consultar (Joyanes & Zohanero, 2005).

2.4.2. Evaluación de expresiones

La evaluación de las expresiones depende de tres cosas, principalmente, el estado de las variables que aparecen en la expresión, el significado de los operadores y su *precedencia*. Esta última se refiere a la prioridad de los operadores, es decir, el orden en el que se evalúan, eliminando con esto la ambigüedad de las expresiones, por ejemplo, si tenemos la expresión:

$$2 + 3 * 5$$

Podríamos evaluarla de dos diferentes formas: La primera es hacer primero la suma $2+3$ ($=5$) y después multiplicar el resultado por 5. De tal manera obtendríamos como resultado final 25. Otra manera sería realizar primero la multiplicación $3*5$ ($=15$) y sumar el resultado a 2, obteniendo 17 como resultado final. Pero sabemos que en matemáticas primero se realiza la multiplicación y después la



suma, en otras palabras, tiene mayor prioridad la multiplicación que la suma. Por lo tanto, el resultado correcto de la expresión $2 + 3 * 5$ es 17. En la siguiente tabla se muestra la precedencia de operadores de lenguaje C que se han presentado

Operadores	Prioridad
()	Mayor
-(unario), ++, --	
*, /, %	
+, -	
<, >, >=, <=	
==, !=	
&&,	
=	Menor

Tabla 2.4: Prioridad de operadores

Los operadores que se encuentran en el mismo nivel de precedencia se ejecutan de izquierda a derecha según aparecen en la expresión.

Veamos el siguiente ejemplo:

Ejemplo 2.3: Dada la siguiente expresión matemática para convertir grados centígrados (C) a su equivalente Fahrenheit (F)

Su codificación en C es: $F = (9.0/5.0)*C + 32;$

Se escribe 9.0 y 5.0 para que la división devuelva un número flotante, de lo contrario la división será entera. En este caso las variables F y C, deben ser declaradas como **float**.

Evaluando, paso a paso, la expresión en el estado de la variable $C = 30$ tenemos:

Expresión actual	Estados de las variables	
$F = (9.0/5.0)*C + 32;$	C	F
	30	¿?
$F = (9.0/5.0)*30 + 32;$	Igual al anterior	
$F = 1.8*30 + 32;$	Igual al anterior	
$F = 54 + 32;$	Igual al anterior	
$F = 86;$	C	F
	30	86



Observa que el único operador que cambia el estado de una variable es el de asignación “=”.

Ejemplo 2.4: Ahora evaluemos la expresión:

$(x \% 2 == 0)$

Considerando que el estado de la variable $x=24$

Expresión actual	Estados de las variables
$x \% 2 == 0$	X 24
$0 == 0$	Igual al anterior
1	Igual al anterior

En este caso la evaluación de la expresión no afecta el estado de la variable, esto es porque no contiene ningún operador de asignación, sin embargo, podemos decir que el resultado de su evaluación es verdadero.

2.5. Bibliotecas y funciones

El lenguaje C en realidad es un lenguaje reducido, en comparación con otros lenguajes de programación, por ejemplo, no tiene instrucciones de entrada y salida, y tampoco cuenta con operadores o funciones para calcular la raíz cuadrada de un número o su potencia, entre otras. Sin embargo, para compensar esto el lenguaje C ofrece un vasto conjunto de bibliotecas de funciones, que para fines prácticos se pueden considerar como parte de C. Cabe mencionar que también es posible definir nuevas bibliotecas, sin embargo, el lenguaje C tiene definidas diversas funciones de uso frecuente, que para fines de este curso son más que suficiente, por lo que este tema no se estudiará en esta ocasión.

Las funciones que más se utilizan están agrupadas en bibliotecas estándar, declaradas como archivos de cabecera, de tal manera que para utilizarlas se debe incluir en el archivo utilizando la directiva `#include` seguida del nombre del archivo encerrado entre “<>”

Las bibliotecas estándar que usaremos en este curso son:

- **stdio.h** en esta biblioteca se encuentran definidas las funciones estándar de entrada y salida – que representan la tercera parte de la biblioteca–, por ejemplo, declara la función `printf` que sirve para imprimir datos en pantalla y `scanf` que se utiliza para leer de datos ingresados mediante el teclado.
- **stdlib.h** incluye funciones para conversión numérica, asignación de memoria y tareas similares, llamadas funciones de utilería. En esta biblioteca se declara la función `system` mediante la cual se envían mensajes al sistema operativo para que ejecute una tarea.



- **math.h** declara funciones matemáticas, como la función **sqrt** que calcula la raíz cuadrada de un número.
- **ctype.h** declara funciones para prueba de clasificación de caracteres, por ejemplo, si es un dígito o un carácter.

Observa que todas las bibliotecas estándar tienen extensión “.h”

2.5.1. Funciones matemáticas

La siguiente tabla muestra algunas de las funciones predefinidas en lenguaje C declaradas en la biblioteca **math.h**.

Función	Descripción	Argumento	Resultado	Ejemplo
sqrt(x)	Raíz cuadrada	flotante	flotante	sqrt(900) = 90
exp(x)	Function exponencial	flotante	flotante	exp(2.0) = 2.718282
fabs(x)	Valor absoluto	entero o flotante	entero o flotante	fabs(-5) = 5
log(x)	Logaritmo neperiano de x	entero o flotante	flotante	log(0.5) = -0.693
log10(x)	logaritmo decimal de x	entero o flotante	flotante	Log10(0.5) = -0.301
floor(x)	Redondeo hacia abajo	flotante	entero	floor(6.5) = 6
ceil(x)	Redondeo hacia arriba	flotante	entero	ceil(6.5) = 7
sen(x)	seno de x	entero o real	flotante	sen(0.5) = 0
pow(x,y)	Devuelve la potencia de x elevada a	entero o flotante	flotante	pow(5,2) = 25
cos(x)	coseno de x	flotante	flotante	cos(0.5) = 1
sin(x)	seno de x	flotante	flotante	sin(0.0) = 0
tan(x)	Tangente de x	flotante	flotante	tan(0.0) = 0

Tabla 2.5: Funciones matemáticas

Con esta información es posible codificar algoritmos que requieran este tipo de operaciones. Por ejemplo, la fórmula para calcular el área de una circunferencia, que se aparece en el programa 2.2.

```
area = PI * radio * radio;
```

Se puede codificar de la siguiente manera:

```
area = PI * pow(radio,2);
```



2.5.2. Funciones de entrada y salida

En los programas que hemos visto aparece la función de salida estándar `printf`, que se encarga de imprimir un mensaje en la pantalla. La sintaxis general es:

```
printf(<cadena_de_control>, <lista_de_identificadores>);
```

Donde `<cadena_de_control>` representa el mensaje de texto que se desea desplegar en el monitor y siempre tiene que ir en comillas, opcionalmente puede incluir secuencias de escape o especificadores de control. Y `<lista_de_identificadores>` es una lista con los identificadores de las variables o las expresiones que serán desplegadas, separadas por comas.

Los especificadores de conversión se utilizan para imprimir valores dentro de la cadena de control e s p e c i f i c a d o s por una variable, una constante o una expresión. En la siguiente tabla se muestran los que más se usan.

Especificador	Acción
%d	Insertar un entero (int)
%i	Insertar un entero tipo (int)
%ld	Insertar un entero tipo (long)
%f	Insertar un número flotante tipo (float)
%lf	Insertar un número de tipo (double)
%c	Insertar un caracter (char)
%s	Insertar una cadena(char [])

Tabla 2.6: Especificadores de conversión

Ejemplo 2.5: Suponiendo que el estado de las variables es `radio=3` y `area=28.27`,

```
printf("El area del circulo con radio %d es %f \n",radio,area);
```

La salida de la instrucción anterior sería:

El área del círculo con radio 3 es 28.27

Observa que se imprime el texto tal cual, pero en vez de imprimir el especificador de conversión `%d` se imprime el valor de la primera variable que es `radio` y en el lugar del especificador `%f` se imprime el valor del siguiente argumento que es la variable `area`.



Ahora, si tenemos la instrucción:

```
printf("El perimetro es %.2f \n", PI*2*radio);
```

La salida sería:

El perimetro es 18.85

En este caso en el lugar del convertidor `%.2f` se imprime el resultado de evaluar la expresión **`PI*2*radio`** que es el segundo argumento, el número `.2` que aparece en el convertidor indica que sólo deben imprimirse dos decimales.

En lenguaje C la lectura de datos por medio del teclado se realiza con la función `scanf`, en la cual se deben de especificar de ante mano los tipos de datos que se desea recibir, además de los identificadores de las variables donde se desean almacenar.

La sintaxis de esta función es:

```
scanf(<cadena_de_control>,<lista_de_direcciones_de_variables>);
```

Donde `<cadena_de_control>` es una cadena con los códigos que controlarán la forma como se recibirán los datos desde teclado y la `<lista_de_direcciones_de_variables>` es una lista con las localidades de memoria de las variables donde se almacenarán los datos que el usuario del programa introduzca a través del teclado.

Dos observaciones importantes: en la especificación de la cadena de control se utilizan los mismos especificadores de conversión que para la función `printf` encerrados entre comillas y en la lista de direcciones los identificadores de las variables, anteponiéndoles a cada uno un símbolo de ampersan “&”, en el mismo orden que los especificadores de tipos que les corresponden.

Ejemplo 2.6: Suponiendo que se desea leer la base y la altura de un rectángulo y guardarlas en las variables de tipo `int` llamadas `base` y `altura`, de tal manera que el usuario ingrese los valores separados por una coma, digamos “5,2” entonces la instrucción sería:

```
scanf("%d,%d", &base, &altura);
```

Nota: Es frecuente que las personas olviden escribir el `&` antes del identificador de una variable, al utilizar la función `scanf`, cuestión que no es supervisada por el compilador y genera un error en el momento que se ejecuta el programa



2.6. Codificación de algoritmos

Para cerrar esta unidad desarrollemos un programa en C que resuelva el siguiente problema:

Descripción del problema: Se requiere un programa que se encargue de la venta de boletos en un cine. El sistema debe calcular el monto que se debe pagar por una cantidad determinada de boletos tomando en cuenta que el costo de cada boleto es de 45 pesos. También se encargará de cobrar, es decir, dado el pago debe calcular el cambio indicando el tipo y número de billetes o monedas que devolverá de cada denominación. Para evitarse problemas de cambio, los ejecutivos de CineESAD han decidido no aceptar monedas de denominación menor a 1 peso y tampoco billetes de denominación mayor a 500 pesos. También se debe suponer que siempre hay suficientes billetes y monedas de cualquier denominación para devolver el cambio.

Análisis: Los datos de entrada son el número de boletos (nboletos) y el monto del pago (pago), la salida del programa es el monto que se debe pagar por los boletos (total) y el monto del cambio (cambio), indicando el tipo y número de cada uno de los billetes o monedas que se devolverán.

Notemos que el precio de los boletos siempre es el mismo, así que se puede declarar una constante, llamémosla PRECIO. Así que para calcular el monto que el cliente debe pagar tenemos la siguiente fórmula:

$$\text{total} = \text{nboletos} * \text{PRECIO}$$

Y el monto del cambio se calcula con la siguiente fórmula:

$$\text{cambio} = \text{pago} - \text{total}$$

Para calcular cuántos billetes o monedas se tienen que devolver, se utilizarán los operadores de módulo y división. Por ejemplo, si el cambio es 360 se puede calcular el número de billetes de 100 dividiendo 360 entre 100, lo cual resulta ser 3 que corresponde al número de billetes, el resto del cambio es igual a 360 módulo 100, en este caso es 60.

Por último, los billetes sólo pueden ser de \$500, \$200, \$100, \$50y \$20y las monedas sólo son de \$10, \$5, \$2y \$1. Por el momento supondremos que el usuario siempre ingresa datos correctos.

Diseño del algoritmo: En la siguiente tabla se muestra el algoritmo que resuelve el problema.

Constantes:

$$\text{PRECIO} = 45$$



Variables:

nboletos: entero, representa el número de boletos que quiere el cliente.

total: entero, es la cantidad de dinero que el cliente debe pagar.

pago: entero, monto del pago del cliente.

cambio: entero, monto del cambio.

```
Inicio
    Imprimir " Proporciona el número de boletos"
    Leer nboletos
    total = nBoletos * PRECIO
    Imprimir "Proporciona tu pago"
    Leer pago
    cambio = pago - total
    Imprimir "Tu cambio es", cambio
    Imprimir "El número de billetes de 200 pesos es",
    cambio/200
    cambio = cambio módulo 200
    Imprimir "El número de billetes de 100 pesos es",
    cambio/100
    cambio = cambio módulo 100
    Imprimir "El número de billetes de 50 pesos es",
    cambio/50
    cambio = cambio módulo 50
    Imprimir "El número de monedas de 10 pesos es",
    cambio/10
    cambio = cambio módulo 10
    Imprimir "El número de monedas de 5 pesos es",
    cambio/5
    cambio = cambio módulo 5
    Imprimir "El número de monedas de 2 pesos es",
    cambio/2
    cambio = cambio módulo 2
    Imprimir "El número de monedas de 1 peso es", cambio
Fin
```

Algoritmo 2.1: ventaBoletos (pseudocódigo)



Nota: Observa que no es necesario utilizar variables para el número de billetes o monedas de las diferentes denominaciones, pues sólo se utiliza una vez el resultado del cálculo así que se puede imprimir directamente el resultado del mismo.

Para verificar que el algoritmo funciona, en la siguiente tabla se realiza una prueba de escritorio considerando que los datos de entrada son 5 boletos y el monto del pago son 500 pesos. En la primera columna aparece la instrucción que se ejecuta, en la siguiente el dato que suponemos se ingresa, después están las operaciones que se realizan en la ALU, en la cuarta columna se muestra los valores de las variables después de que se ha realizado la instrucción y en la última columna se indica el mensaje que se imprimirá en la pantalla, cuando sea el caso. Por otro lado, incluimos en la tabla de estado de la memoria la constante PRECIO sombreando el espacio correspondiente para indicar que no puede ser modificado.

Instrucción	Dato de entrada	Operaciones	Estado de la memoria (variables y constantes)					Dato de salida
Inicio	-	-	PRECIO	nBoletos	total	pago	cambio	-
			45	-	-	-	-	
Imprimir "Proporciona el número de boletos".	-	-	PRECIO	nBoletos	total	pago	cambio	Proporciona el número de boletos
Leer nBoletos	5	-	PRECIO	nBoletos	total	pago	cambio	-
			45	5	-	-	-	
total = nBoletos*PRECIO	-	total = 5*45 = 225	PRECIO	nBoletos	total	pago	cambio	-
				5	225			
Imprimir "Proporciona tu pago"	-	-	PRECIO	nBoletos	total	pago	cambio	Proporciona tu pago
				5	225	-		
Leer nBoletos	500	-	PRECIO	nBoletos	total	pago	cambio	-
			45	5	225	500	-	
cambio = pago - total	-	cambio = 500-225 = 275	PRECIO	nBoletos	total	pago	cambio	-
				5	225	500	275	
Imprimir "Tu cambio"	-	-	PRECIO	nBoletos	total	pago	Cambio	Tu cambio
				5	225	500	275	



es”, cambio.				es 275										
Imprimir “El número de billetes de \$200 es”, cambio/200	-	$275/200 = 1$	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td></td><td>5</td><td>225</td><td>500</td><td>275</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio		5	225	500	275	El número de billetes de \$200 es 1
PRECIO	nBoletos	total	pago	Cambio										
	5	225	500	275										
cambio = cambio módulo 200	-	$\text{cambio} = 275 \bmod 200 = 75$	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td></td><td>5</td><td>225</td><td>500</td><td>75</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio		5	225	500	75	-
PRECIO	nBoletos	total	pago	Cambio										
	5	225	500	75										
Imprimir “El número de billetes de \$100 es”, cambio/100	-	$75/100 = 0$	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td></td><td>5</td><td>225</td><td>500</td><td>75</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio		5	225	500	75	El número de billetes de \$100 es 0
PRECIO	nBoletos	total	pago	Cambio										
	5	225	500	75										
cambio = cambio módulo 100	-	$\text{cambio} = 75 \bmod 100 = 75$	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td></td><td>5</td><td>225</td><td>500</td><td>75</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio		5	225	500	75	-
PRECIO	nBoletos	total	pago	Cambio										
	5	225	500	75										
Imprimir “El número de billetes de \$50 es”, cambio/50	-	$75/50 = 1$	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td></td><td>5</td><td>225</td><td>500</td><td>75</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio		5	225	500	75	El número de billetes de \$50 es 1
PRECIO	nBoletos	total	pago	Cambio										
	5	225	500	75										
cambio = cambio módulo 50	-	$\text{cambio} = 75 \bmod 50 = 25$	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td></td><td>5</td><td>225</td><td>500</td><td>25</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio		5	225	500	25	-
PRECIO	nBoletos	total	pago	Cambio										
	5	225	500	25										
Imprimir “El número de billetes de \$20 es”, cambio/20	-	$25/50 = 0$	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td></td><td>5</td><td>225</td><td>500</td><td>25</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio		5	225	500	25	El número de billetes de \$50 es 0
PRECIO	nBoletos	total	pago	Cambio										
	5	225	500	25										
cambio = cambio módulo 20	-	$\text{cambio} = 25 \bmod 20 = 5$	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td></td><td>5</td><td>225</td><td>500</td><td>5</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio		5	225	500	5	-
PRECIO	nBoletos	total	pago	Cambio										
	5	225	500	5										
Imprimir “El número de monedas de \$10 es”, cambio/10	-	$5/10 = 0$	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td></td><td>5</td><td>225</td><td>500</td><td>5</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio		5	225	500	5	El número de moneda s de \$10 es 0
PRECIO	nBoletos	total	pago	Cambio										
	5	225	500	5										



cambio = cambio módulo 10	-	cambio = 5 mod 10 =5	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td>45</td><td>5</td><td>225</td><td>500</td><td>5</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio	45	5	225	500	5	-
PRECIO	nBoletos	total	pago	Cambio										
45	5	225	500	5										
Imprimir "El número de monedas de \$5 es", cambio/5	-	5/5= 1	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td>45</td><td>5</td><td>225</td><td>500</td><td>5</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio	45	5	225	500	5	El número de monedas de \$5 es 1
PRECIO	nBoletos	total	pago	Cambio										
45	5	225	500	5										
cambio = cambio módulo 5	-	cambio = 5 mod 5 =0	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td>45</td><td>5</td><td>225</td><td>500</td><td>0</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio	45	5	225	500	0	-
PRECIO	nBoletos	total	pago	Cambio										
45	5	225	500	0										
Imprimir "El número de monedas de \$2 es", cambio/2	-	0/2= 0	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td>45</td><td>5</td><td>225</td><td>500</td><td>0</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio	45	5	225	500	0	El número de monedas de \$2 es 0
PRECIO	nBoletos	total	pago	Cambio										
45	5	225	500	0										
cambio = cambio módulo 2	-	cambio = 0 mod 2 =0	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td>45</td><td>5</td><td>225</td><td>500</td><td>0</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio	45	5	225	500	0	-
PRECIO	nBoletos	total	pago	Cambio										
45	5	225	500	0										
Imprimir "El número de monedas de \$1 es", cambio	-	-	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td>45</td><td>5</td><td>225</td><td>500</td><td>0</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio	45	5	225	500	0	El número de monedas de \$1 es 0
PRECIO	nBoletos	total	pago	Cambio										
45	5	225	500	0										
Fin	-	-	<table> <tr> <th>PRECIO</th><th>nBoletos</th><th>total</th><th>pago</th><th>Cambio</th></tr> <tr> <td>45</td><td>5</td><td>225</td><td>500</td><td>0</td></tr> </table>	PRECIO	nBoletos	total	pago	Cambio	45	5	225	500	0	-
PRECIO	nBoletos	total	pago	Cambio										
45	5	225	500	0										

Implementación: Por último, sólo queda codificar el algoritmo en lenguaje C

*/*Descripción: Simulador de la caja de cobro de un cine.*/*

*/*Directivas al procesador */*

#include<stdio.h>*/* Funciones de entrada y salida */*

#include<stdlib.h>*/* Funciones del sistema */*

/ Función Principal */*

main()

{

*/*Declaración de variables y constantes */*

constint precio = 45;



```

int nBoletos, total, pago, cambio;

/*Mensaje de bienvenida*/
printf("***** Venta de boletos CineESAD*****\n\n");

/*Datos de entrada*/
printf("Proporcione el numero de boletos que desea comprar:\t");
scanf("%d",&nBoletos);

/*Calculamos el total de la venta*/
total = nBoletos*precio;
printf("El total es *** %d pesos *** \n\n",total);

/*Leemos el pago y calculamos el
cambio*/ printf("Indique el monto de
su pago: "); scanf("%d",&pago);

/*Calculamos el cambio y lo devolvemos*/
cambio = pago - total;
printf("\n\n El monto de su cambio es %d\n",cambio);
printf("\n\t%d billetes de $200", cambio/200);
cambio = cambio%200;
printf("\n\t%d billetes de $100", cambio/100);
cambio = cambio%100; printf("\n\t%d billetes de $50", cambio/50);
cambio = cambio%50;
printf("\n\t%d billetes de $20", cambio/20);
cambio = cambio%20;
printf("\n\t%d monedas de          $10", cambio/10);
cambio = cambio%10;
printf("\n\t%d monedas de          $5", cambio/5);
cambio = cambio%5;
printf("\n\t%d monedas de          $2", cambio/2);
cambio = cambio%2;
printf("\n\t%d monedas de          $1", cambio);
printf("\n\nCineUNAD le agradece su preferencia\n\n");
system("pause");/*hará una pausa antes de terminar la ejecución*/
}/*fin main*/

```

Programa 2.3: ventaBoletos.c

El resultado de la ejecución del programa utilizando los mismos datos de entrada de la prueba de escritorio es:



```
Proporcione el numero de boletos que desea comprar: 5
El total es *** 225 pesos ***
Indique el monto de su pago: 500

El monto de su cambio es 275

1 billetes de $200
0 billetes de $100
1 billetes de $50
1 billetes de $20
0 monedas de $10
1 monedas de $5
0 monedas de $2
0 monedas de $1

CineMar le agradece su preferencia
Presione una tecla para continuar . . . _
```

Figura 2.3: Ejecución del programa ventaBoletos.c



Para desarrollar más tus habilidades de programador de una forma divertida consulta: <http://code.org/> en donde existen distintas actividades que te ayudan a mejorar en la resolución de problemas.



Se sugiere consultes un videotutorial <http://www.microsoftvirtualacademy.com/training-courses/curso-aprendiendo-a-programar> para incrementar tus conocimientos.



Actividades

La elaboración de las actividades estará guiada por tu figura académica, mismo que te indicará, a través de la Planificación de actividades de la figura académica, la dinámica que tú y tus compañeros (as) llevarán a cabo, así como los envíos que tendrán que realizar.

Para el envío de tus trabajos usarás la siguiente nomenclatura: BPRG_U2_A1_XXYZ, donde BPRG corresponde a las siglas de la asignatura, U2 es la unidad de conocimiento, A1 es el número de actividad, el cual debes sustituir considerando la actividad que se realices, XX son las primeras letras de tu nombre, Y la primera letra de tu apellido paterno y Z la primera letra de tu apellido materno.

Autorreflexiones

Para la parte de **autorreflexiones** debes responder las *Preguntas de Autorreflexión* indicadas por tu figura académica y enviar tu archivo. Cabe recordar que esta actividad tiene una ponderación del 10% de tu evaluación.

Para el envío de tu autorreflexión utiliza la siguiente nomenclatura:

BPRG_U2_ATR_XXYZ, donde BPRG corresponde a las siglas de la asignatura, U2 es la unidad de conocimiento, XX son las primeras letras de tu nombre, y la primera letra de tu apellido paterno y Z la primera letra de tu apellido materno

Cierre de la unidad

Con esto concluimos la segunda unidad del curso de Fundamentos de Programación. A lo largo de esta Unidad se han presentado los componentes básicos de un programa de computadora, así como las diferencias y usos de las variables y constantes. También revisamos las expresiones matemáticas, los tipos de operadores que se usan en los programas de computadora y la forma en que se realiza la evaluación de expresiones matemáticas. Así mismo analizamos las bibliotecas y funciones disponibles, tales como funciones matemáticas, funciones de entrada y salida. Finalmente realizamos la codificación de algoritmos.



Es aconsejable que revises nuevamente el material en caso de que algún tema de los vistos en esta unidad no te haya quedado lo suficientemente claro, de lo contrario te invitamos a revisar la siguiente unidad.

Fuentes de consulta



Fuentes básicas

- Böhm, C., & Jacopini, G. (1966). Flow diagrams, Turing machines, and languages only with two formation rules". *Communications of the ACM*, 9 (5), 366-371.
- Cairó, O. (2005). Metodología de la programación: Algoritmos, diagramas de flujo y programas. México, D.F.: Alfaomega.
- Joyanes, L., & Zohanero, I. (2005). Programación en C. Metodología, algoritmos y estructuras de datos. aspaño: Mc Graw Hill.

Fuentes complementarias

- Kernighan, B., & Ritchie, D. (1991). El lenguaje de programación C. México: Prentice-Hall Hispanoamericana.
- López, L. (2005). Programación estructurada en lenguaje C. México: Alfaomega.
- Reyes, A., & Cruz, D. (2009). Notas de clase: Introducción a la programación. México, D.F.: UACM.





Fuentes electrónicas

- Guerrero, F. (s.f.). mailxmail.com. Recuperado el 15 de 8 de 2010, de <https://web.archive.org/web/20221129015722/http://www.mailxmail.com/curso-introduccion-lenguaje-c>
- Villela, H. T. (20 de agosto de 2010). Manual de C. Consultado el 25 de marzo de 2020 en: <https://diarium.usal.es/mlperez/files/2012/06/lenguajec-unix-gcc.pdf>