2

Introducción a la Programación en Lenguaje C

El componente principal de un programa en C es la función. Una función es una unidad lógica que se encarga de implementar una funcionalidad diferenciada por el programador.

Contenido

¿Qué es el Lenguaje C?

C es un lenguaje de programación desarrollado en los Laboratorios Bell de AT&T en 1972. Fue diseñado y escrito por una persona, Dennis Ritchie, quien trabaja en conjunto con Ken Thompson en el sistema operativo UNIX. El UNIX se concibió como una especie de taller lleno de herramientas para el especialista en programación y C se convirtió en la herramienta más importante de todas. Casi todas las herramientas de programación suministradas con UNIX, incluyendo el compilador C y casi todo el sistema operativo, se escriben ahora en C.

A mediado de los años setenta el UNIX se extendió fuera de los Laboratorios Bell y se comenzó a utilizar ampliamente en las universidades. C comenzó entonces a sustituir a los lenguajes más familiares disponibles en UNIX.

Entre las características más importantes del C está la fiabilidad, regularidad, simplicidad y facilidad de uso. Pertenece al grupo de lenguajes llamados "estructurados", puesto que facilita la programación estructurada.

La genealogía del C es fácil de trazar. El lenguaje C combina generalidad, simplicidad, un uso hábil de los tipos de datos y "contacto con la computadora".

Así pues, el dominio especial del C es la programación de sistemas, ya que es un lenguaje de relativo bajo nivel que permite especificar cada detalle de la lógica de un programa para conseguir un máximo rendimiento de la computadora, pero a la vez es un lenguaje de alto nivel que oculta los detalles de la arquitectura de la computadora mejorando el rendimiento de la programación.

Estructura de un Programa en C

Un programa en C puede ser definido como una colección de funciones, cada una de las cuales cumple una labor específica. De las funciones que componen un programa en C existe una con particular importancia: la función main. Esta función constituye el cuerpo principal del programa y es la primera en ejecutarse, una vez que la aplicación ha sido puesta en funcionamiento.

Dentro de las funciones pueden existir datos e instrucciones. Los datos, en forma de variables y constantes, constituyen la información a ser procesada por las instrucciones. Los datos definidos dentro de una función se consideran locales, y solamente pueden ser accedidos desde dentro de la función donde han sido declarados.

También es posible definir datos fuera de las funciones, en cuyo caso se denominan como globales, lo que significa que pueden ser accedidos desde cualquier función, siempre y cuando no exista una redefinición local.

Las instrucciones sólo pueden ser definidas dentro de las funciones.

Datos en C

Un dato es un valor actual en un programa, que puede ser constante o variable. Las constantes son contienen valores fijos que no cambian durante la ejecución del programa. Las variables contienen valores que pueden cambiar.

En C existen dos tipos fundamentales de datos: entero y punto flotante. De estos se derivan dos tipos adicionales: carácter y doble precisión. A partir de los cuatro tipos básicos de datos, es posible derivar cualquier otro tipo posible.

Enteros

Un entero es un valor positivo o negativo que no tiene parte fraccionaria. El tamaño de un entero depende del diseño de una computadora en particular. Normalmente, la longitud de enteros es de 16 bits, lo que significa que el rango de valores representables con un entero va desde –32768 hasta 32767.

La palabra clave para declarar una variable entera es int. Una vez que se ha declarado una variable entera, es posible asignar un valor a ella utilizando el operador de asignación =.

```
int fecha_de_graduacion, puesto, numero_estudiante;

numero_estudiante= 31670;
puesto= 473;
fecha_de_graduacion= 1985;
```

Punto Flotante

Punto flotante es una representación numérica de datos reales. Los datos reales son valores que poseen parte entera y parte fraccionaria y que pueden ser negativos o positivos.

La palabra clave para declarar una variable real en C es float. Esta palabra define lo que se denomina como un valor real en simple precisión y tiene una longitud de 4 bytes.

```
float PI= 3.1415927;
```

Carácter

El tipo de dato carácter corresponde con el de un entero cuyo tamaño es de sólo 8 bits, con lo que es posible representar valores entre –128 y 127 ambos inclusive. Sin embargo, normalmente se emplea para representar caracteres alfanuméricos.

La palabra clave para declarar una variable carácter es char. Se muestra un ejemplo a continuación:

```
char sexo= 'F';
```

Doble Precisión

El tipo de dato de doble precisión representa valores reales con una precisión equivalente al doble de la utilizada por el tipo float. De hecho, este tipo de dato ocupa 8 bytes en memoria (el doble que el tipo de dato float) y es extensamente usado en la librería de funciones matemáticas del C.

La palabra clave para declarar variables de tipo doble precisión es double. Un ejemplo de declaración de una variable en doble precisión, se muestra a continuación:

```
double Factor_de_Ajuste= 3.4672871633;
```

Modificadores

En adición a los tipos de datos primarios, es posible definir ciertas variantes, mediante el uso de modificadores. Los modificadores son instrucciones que actúan sobre ciertos tipos de datos primarios alterando sus características. Entre ellos encontramos long, short y unsigned.

Long

El modificador long permite duplicar el tamaño de almacenamiento (y por ende el rango de representación) de ciertos tipos de datos. Puede ser aplicado sobre los tipos de dato entero y doble precisión.

Al ser usado sobre el tipo de dato entero, se obtiene un entero largo. Este entero ocupa un espacio de memoria de 4 bytes. A continuación se muestra un ejemplo:

```
long int A= 2390489;
long B= 346721;
```

El ejemplo de la variable B, muestra como puede ser omitida la palabra int cuando se define un entero en C. La presencia de sólo la palabra long, hace que el compilador asuma que el tipo de dato básico seleccionado sea entero.

Cuando se emplea el modificador long en combinación con un dato de tipo doble precisión, se obtiene una representación de punto flotante de cuádruple precisión.

```
long double Factor_de_Ajuste= 0.004378281782478367;
```

Short

El modificador short tiene un efecto exactamente opuesto al del modificador long; disminuye a la mitad el tamaño de almacenamiento del tipo de dato sobre el cual aplica. Sólo es válido aplicarlo sobre el tipo de dato entero y en la mayor parte de las implementaciones de compiladores de C en entorno DOS o Windows, no tiene efecto alguno, de tal forma que definir una variable entera o una variable entera corta es exactamente igual. La forma de construcción de la definición de una variable entera corta es como se muestra:

```
short int A= 23;
short B= 34;
```

Unsigned

El modificador unsigned anula la capacidad de los datos de tipo entero y char para representar valores negativos. Esto se consigue, cambiando la interpretación el bit normalmente usado para el signo. Esto trae como consecuencia, que el rango de números positivos representados se duplique. Puede ser aplicado sobre enteros ordinarios, largos y cortos y sobre el tipo de dato carácter. A continuación se muestran ejemplos de tipos de dato sin signo:

```
unsigned int Edad= 21;              // entero ordinario sin signo
unsigned long CI= 15785873;         // entero largo sin signo
unsigned char A;                    // carácter sin signo
```

Clases de Almacenamiento

Otro aspecto relacionado con la definición de datos en C, son las clases de almacenamiento. Éstas están referidas al lugar de memoria donde se deposita la información correspondiente a una variable y al alcance de la misma.

Las clases de almacenamiento son cuatro: automáticas, registro, estática y externa.

Automáticas

Las variables automáticas son aquellas que usadas ordinariamente. Aunque se puede utilizar la palabra clave auto para declararlas, su uso no es obligatorio, de tal forma que al no colocar de manera explícita la clase de almacenamiento de una variable, el compilador asumirá que se trata de una variable automática.

El alcance de una variable automática se limita al bloque en el cual aparece. Mientras dicho bloque se esté ejecutando, la variable existe. Tan pronto como el bloque en cuestión finaliza su ejecución, la variable deja de existir. En este caso existir significa ocupar espacio en memoria.

Si una variable automática es declarada fuera de cualquier función, su alcance será global (cualquier función podrá tener acceso a ella) y su período de vida será igual al de la aplicación

donde se encuentra. Por el contrario, si una variable automática es declarada dentro de una función, su alcance estará restringido a la función en cuestión y su tiempo de vida, será igual al tiempo que tome en ejecutarse dicha función. Si la función es invocada varias veces durante la ejecución de una aplicación, cada vez las variables locales (automáticas declaradas dentro de la función) comienzan su existencia y al finalizar la función, terminan. Cada renacer de una misma variable local, no guarda relación alguna con un caso anterior.

Registro

La clase de almacenamiento registro indica al compilador, que el espacio de memoria donde se almacenará la información correspondiente a una variable será uno de los registros internos del microprocesador, claro está, si alguno está disponible.

Este mecanismo permite optimizar enormemente el tiempo de acceso a la información contenida en la variable. La palabra clave para hacer uso de esta clase de almacenamiento es register y la sintaxis de construcción de un variable de este tipo es como se muestra a continuación:

```
register int A;
register unsigned long B;
```

Estática

Las variables estáticas proporcionan a una función la capacidad de recordar lo hecho en una instancia anterior.

La clase de almacenamiento estática, obliga al compilador a reservar espacio de memoria en el segmento de datos para una variable aún cuando ésta sea local. Esto provoca que dicha variable tenga una vida útil igual a la de la aplicación en su totalidad. Esto hace que variables locales, una vez inicializadas, no desaparezcan sino hasta que finalice su ejecución la aplicación. Esto permite que el contenido de dicha variable local conserve el último valor de una instancia a otra de ejecución de la función que la contiene.

La palabra clave para declarar una variable estática es static. La forma de declarar una variable estática es como se muestra a continuación:

```
static int estatus;
static long double Ajuste= 0.46767236;
```

Externa

La clase de almacenamiento externa se emplea cuando se escriben programas en múltiples archivos y es necesario ganar acceso a una variable declarada en un archivo, desde una porción de código ubicada en otro. En dicho caso, en el módulo donde es declarada la variable, esto se

hace de la forma convencional, pero en el otro archivo desde donde se quiere ganar acceso a la variable, esta es declarada anteponiendo la palabra clave extern, como se muestra en el ejemplo:

```
extern int A;
```

Instrucciones o Sentencias en C

Las instrucciones o sentencias son comandos que indican dentro de un programa, que acciones tomar. Las instrucciones pueden incluir expresiones aritméticas, lógicas y condicionales, llamadas a funciones y estructuras de control

Expresiones Aritméticas

Las expresiones aritméticas son sentencias constituidas por operandos y operadores (aritméticos) que constituyen expresiones matemáticas. Los operadores son símbolos que representan alguna operación particular que se puede realizar sobre un dato. El valor del propio dato (que puede ser una variable o una constante) se llama operando. Los operandos están sometidos a estrictas reglas de precedencia que permiten que el compilador decida cuál debe ejecutarse primero. La sintaxis para construcción de las expresiones aritméticas es similar al de las expresiones matemáticas.

```
c= a + b;        // calcula la suma de a y b
d= a%2;          // calcula el residuo de la división entera de a entre 2
--x;             // decrementa el valor de la variable x en 1
```

Expresiones Lógicas

Las expresiones son similares a las aritméticas, sólo que involucrando operadores lógicos.

```
c= a & b;        // devuelve el resultado de ejecutar la operación y
                 // lógico entre los bits de a y b
a= ~c;           // calcula el complemento a 1 de c
```

Expresiones Condicionales

Las expresiones condicionales son similares a las expresiones aritméticas; la diferencia con respecto a las anteriores estriba en que el resultado de dicha expresión será interpretado en términos de veracidad o falsedad. Además, la escritura de expresiones lógicas típicamente involucrará el uso de operadores relacionales.

```
a >= b;          // devuelve verdad (1) si a es mayor o igual que b y
                 // falso (0) en caso de que no lo sea
a == b;          // devuelve verdad si a es igual a b
```

```
a ¡= b;          // devuelve verdad si a es diferente de b
```

Llamadas a Funciones

Las llamadas a funciones se realizan indicando el nombre de la función a invocar, incluyendo la lista de argumentos que corresponda. Si la función "retorna" un valor, opcionalmente podrá indicarse la variable donde dicho valor será almacenado.

```
a= sin(x);       // calcula el seno de la x y deposita el resultado en a
printf("HOLA");  // muestra el texto HOLA en pantalla
scanf("%d",&X);  // lee un número entero por teclado y lo deposita en X
```

Estructuras de Control .....

Las estructuras de control son aquellas instrucciones que permiten interrumpir la ejecución secuencial de un programa, con base en una condición dada. Existen dos tipos de estructuras de control: las de selección y las de iteración.

Estructuras de Control de Selección

Las estructuras de control de selección, son aquellas que permiten ejecutar o no una porción de código del programa, basada en una condición dada. Las estructuras de control de selección pueden ser simples o múltiples.

Las estructuras de control de selección simple permiten decidir si una porción de código se ejecutará o no, o escoger entre dos porciones de código. En lenguaje C, la estructura de control de selección simple viene representada por la sentencia if.

La construcción sintáctica de la sentencia if es como se muestra a continuación:

```
if (expresión_condicional)
   sentencia(s);
```

La palabra clave if le dice al compilador que si la condición especificada (entre paréntesis) es cierta, se ejecute la sentencia a continuación; si la condición no es cierta, la sentencia a continuación se omite y el programa continúa con la primera instrucción luego de la estructura de control.

La sentencia bajo la acción del if puede ser una única instrucción o un conjunto de ellas, como se muestra en el ejemplo:

```
if (expresión_condicional)
{
   instrucción1;
   instrucción2;
   instruccion3;
```

```
}
```

Las llaves abierta y cerrada indican que el conjunto de instrucciones entre ellas encerradas, están bajo la acción de la sentencia if.

La versión extendida de la sentencia if, incluye el especificar qué hacer si la condición evaluada es falsa. La sintaxis en dicho caso es como se muestra:

```
if (expresión_condicional)
        sentencia1;
else    sentencia2;
```

Si la condición es cierta, se ejecuta sentencia1 y si no, se ejecuta sentencia2; sentencia1 y sentencia2 pueden ser reemplazadas por conjuntos de instrucciones debidamente encerradas entre llaves.

Las estructuras de control de selección múltiple permiten seleccionar cual porción de código ejecutar de entre un conjunto de 1 o más opciones. En lenguaje C la estructura de control de selección múltiple viene representada por sentencia switch. La sintaxis de la sentencia switch se muestra a continuación:

```
switch (clave)
{
   case A:
        sentenciaA;
        break;

   case B:
        sentenciaA;
        break;

   default:
        sentencia_por_defecto;
}
```

El argumento de la sentencia switch es un entero que permitirá seleccionar la sentencia o grupo de ellas según la clave coincida o no con algunos de los casos especificados. Por ejemplo, si la clave coincide con el caso A, entonces se ejecuta sentenciaA; si la clave coincide con el caso B, entonces se ejecuta sentenciaB y así sucesivamente. El caso por defecto (identificado por la palabra default) se va a ejecutar cuando la clave dada no coincida con ninguno de los casos especificados. La presencia de la cláusula por defecto es opcional. De no existir dicha cláusula, al no coincidir la clave con ninguno de los casos especificados, la ejecución del programa va a la primera instrucción luego de la estructura de control.

Nótese que en el ejemplo anterior se empleó la sentencia break para finalizar cada conjunto de sentencias asociadas a una clave particular. Esto se hace para obligar a que luego de que se ejecuten todas las sentencias asociadas a una clave en particular, se bifurque la ejecución del programa a la primera instrucción luego de la estructura. Si no se coloca la sentencia break, se

ejecutarán las instrucciones a continuación, no importando que las sentencias siguientes pertenezcan o estén asociadas a otra clave dentro de la estructura switch.

Estructuras de Control de Iteración

Las estructuras de control de iteración permiten que una sentencia o conjunto de ellas se ejecute repetidamente mientras o hasta que se de una condición determinada. En lenguaje C, existen tres estructuras de control de iteración: while, do-while y for.

La estructura de control de iteración while permite especificar que una porción de código se ejecuta repetidamente, mientras la condición especificada sea verdad. La sintaxis de construcción de la sentencia while es como se muestra:

```
while (expresión_condicional)
   sentencia;
```

Mientras la condición sea verdad, se ejecuta la sentencia. Al igual que en casos descritos anteriormente, es posible que un grupo y no una sola sentencia, estén bajo la acción de la estructura de control, como se muestra a continuación:

```
while (expresión_condicional)
{
   sentencia1;
   sentencia2;
   sentencia3;
}
```

Las llaves permiten agrupar un conjunto de sentencias bajo la influencia de la estructura de control while. En el caso de que la condición sea verdad, se ejecutarán las sentencias encerradas en las llaves.

Ya que la condición es verificada antes de ejecutarse las sentencias, es posible que estas no lleguen a ejecutarse nunca, ya que puede darse el caso de que la condición sea falsa la primera vez que es verificada.

La estructura de control do-while funciona de manera similar a la estructura while. La diferencia estriba en el hecho de que la condición que se verifica para permitir o no una nueva iteración del código bajo la acción de la estructura de control, es chequeada al final de cada iteración. Esto trae como consecuencia práctica, que el código bajo la acción de la estructura de control, se ejecutará al menos una vez.

La sintaxis de construcción asociada a esta estructura de control es como se muestra:

```
do{
   sentencia1;
   sentencia2;
   sentencia3;
```

```
} while (expresión_condicional);
```

La estructura de control for en lenguaje C, viene a ser una ampliación de la estructura while, donde se contemplan, además de la condición de iteración, la inicialización de variables involucradas con el código que itera así como la actualización automática de éstas u otras variables, luego de culminar cada iteración.

La sintaxis de construcción es como se muestra:

```
for (inicialización; expresión_condicional; actualización)
   sentencia;
```

La estructura de control for opera de la siguiente manera. Al entrar en funcionamiento dicha estructura, son ejecutadas las sentencias correspondientes a la inicialización. En la inicialización pueden incluirse cualquier tipo de expresión válida en lenguaje C y pueden existir tantas sentencias de inicialización como sea necesario.

El campo expresión_condicional contendrá una expresión cuyo resultado será evaluado en términos de verdad o falso y que será verificada antes de cada iteración; permitirá decidir si se ejecuta o no el código bajo la acción de la estructura.

El campo actualización contendrá una o más sentencias consistentes en expresiones válidas en C, que permitirán la actualización de variables tras cada iteración.

Los campos descritos, se separan con punto y coma. Las sentencias de inicialización y las sentencias de actualización estarán separadas entre sí por comas.

```
for (i=0; i<10; i++)
```

Al ejecutarse la sentencia for, el primer paso es inicializar la variable i a cero; luego de esto se verifica la condición. Si la condición arroja una resultado igual a verdad, se ejecutan las sentencias bajo a la acción de la estructura o se haya encontrado una sentencia continue.

Una vez que se han ejecutado todas las sentencias bajo la acción de la estructura, se lleva a cabo la sentencia de actualización, que en este caso incrementa el contenido de la variable en 1. Este proceso se repetirá hasta que la condición de iteración se falsa o se haya encontrado una sentencia break.

Funciones en C

En C, toda función está compuesta de un encabezado que contiene el nombre de la función y la lista de argumentos, así como del cuerpo de la función donde se incluyen sentencias para declaración de variables y para ejecución de comandos.

La sintaxis para construir el encabezado de una función es como sigue:

```
[tipo] nombre_de_la_funcion ([lista de argumentos])
```

El tipo, se refiere a la naturaleza de la información que es asignada al nombre de la función. Éste puede ser cualquiera de los tipos de datos primarios o variantes disponibles en C. Si este es el caso, el nombre de la función tendrá asociado un valor. Asimismo el tipo puede adoptar el valor void en cuyo caso, la función no tendrá asociado valor alguno a su nombre.

El nombre de la función es una etiqueta, que identifica de manera única a la función dentro de un programa y que servirá para hacer referencia a ella. Esta etiqueta puede estar constituida por hasta 31 caracteres que pueden ser números, letras del alfabeto anglosajón y algunos caracteres especiales. El nombre de una función nunca iniciará con un número.

Las lista de argumentos está constituida por la definición de una serie de contenedores que servirán como receptáculo de la información que es transferida a la función al ser invocada su ejecución.

```
int suma (int a, int b)
{
   return a+b;
}
```

La función del ejemplo cuyo propósito es calcular la suma de dos números enteros, recibe el nombre de suma, recibe dos argumentos de tipo entero (int) llamados a y b respectivamente y devuelve como resultado un valor de tipo entero, que en el caso de esta función corresponde a la suma de los valores contenidos en los argumentos. La instrucción return acompañada de la expresión aritmética a+b, se encargan de sumar los argumentos y e "retornarlos" como respuesta de la función.

La manera como se declaró esta función, permite que la misma sea invocada como se muestra a continuación:

```
main ()
{
   int valor1;
   int valor2;
   int resultado;

   valor1= 50;
   valor2= 60;

   resultado= suma(valor1,valor2);
}
```

La función main descrita contiene la declaración de tres variables enteras y la posterior asignación de valores a dos de dichas variables. La última sentencia invoca a la función suma, proporcionándole como argumento los valores de las variables valor1 y valor2 y almacenando el resultado de la función en la variable resultado.

El programa constituido por las funciones suma y main descritas anteriormente es completo y producirá un ejecutable válido tras su compilación, aunque no solicite datos al usuario ni produzca resultados visibles por pantalla. Para incorporar esta funcionalidad, habría que hacer algunas modificaciones, como se muestra a continuación:

```
#include <stdio.h>

int suma (int a, int b)
{
   return a+b;
}

main ()
{
   int valor1;
   int valor2;
   int resultado;

   printf("\nIngrese el valor del primer sumando");
   scanf("%d",&valor1);
   printf("\nIngrese el valor del segundo sumando");
   scanf("%d",&valor2);

   printf("\nEl resultado de la suma es %d:",suma(valor1,valor2);
}
```

En este ejemplo, se incluyen las funciones printf y scanf. Estas funciones permiten mostrar cadenas de texto por pantalla así como leer información proveniente del teclado. Estas funciones pertenecen a la librería llamada stdio.h; por este motivo, debe agregarse la sentencia #include <stdio.h> de tal manera que el enlazador sepa donde encontrar la definición de dichas funciones, durante el proceso de creación del archivo ejecutable.

La función printf responde a la siguiente sintaxis:

```
int printf (const char *cadena_de_formato,…);
```

donde el argumento formato es una cadena de caracteres que describe el formato que será utilizado para mostrar los valores suministrados como argumento (representados por …). Luego de la cadena de formato, se especifican uno o más argumentos adicionales que corresponderán con los valores que deban ser mostrados por pantalla.

En el caso del programa anterior, y analizando la última sentencia de la función main se tiene lo siguiente:

1. La cadena de formato comienza con una secuencia de escape identificada como \n. Este código indica que debe hacerse un salto de línea antes de mostrar los siguientes datos.
2. Luego se encuentra la cadena de texto "El resultado de la suma es". Este texto será desplegado tal cual en la pantalla

3. Luego aparece el carácter de control %d. Esto indica que el primer argumento luego de la cadena de formato, debe ser mostrado por pantalla considerando que se trata de un número entero
4. La cadena de formato culmina con el carácter ':'
5. Luego de la cadena de formato se indica como primer (y único en este caso) argumento, dado por el resultado que "retorna" la función suma

Al ejecutar la instrucción se ejecutará la función suma, la cual operará sobre los argumentos dados, para luego mostrar el texto indicado en la cadena de formato.

La función scanf debe escribirse de la siguiente manera:

```
int scanf (const char *cadena_de_formato,…)
```

donde el argumento formato especifica la manera como debe ser interpretada la información ingresada por teclado. Los argumentos pueden ser uno o más y hacen referencia a las variables donde serán almacenados los valores ingresados.

En el caso de la primera sentencia scanf utilizada en la función main del programa anterior, se tiene que la cadena de formato contiene el carácter de control '%d'. Esto indica que el valor ingresado por teclado va a ser interpretado como un número entero. El argumento indicado (uno en este caso) corresponde a la variable valor1, previamente declarada dentro de la función. Al ejecutarse la función, el programa se detendrá a esperar que el usuario ingrese un dato y una vez que presione la Intro, la función procesará e interpretará los datos ingresados como un número entero; dicho número será almacenado en la variable valor1.

Nótese que la variable valor1 está precedida del carácter &. Esto se debe, a que para que la función pueda depositar el resultado de su proceso en la variable dada como argumento, esta debe ser suministrada por referencia. Esto significa, que la función scanf está recibiendo la dirección de memoria donde se encuentra localizada la variable valor1, en lugar de la variable en si.