Programación en Lenguaje C

Introducción

El Lenguaje de programación C fue implantado por primera vez en los años 70 por Dennis Ritchie para una computadora DEC PDP-11 corriendo el sistema operativo UNIX.

El lenguaje C, a diferencia de lenguajes como Basic y Pascal que son lenguajes de alto nivel, es un lenguaje de nivel medio ya que combina las propiedades de un lenguaje de alto nivel con la funcionalidad del lenguaje ensamblador.

Es esa misma característica la que le da a este lenguaje más potencia y dominio sobre los recursos del sistema. Entre una de las características más importantes de C está la portabilidad, esto es, un programa escrito en C se puede trasladar fácilmente entre distintos sistemas operativos como Windows, DOS o Linux.

Tipos de Datos

Las computadoras pueden trabajar con varios tipos de datos, los algoritmos y programas operan sobre estos. Existen dos clases de datos: datos simples y datos compuestos.

Los distintos tipos de datos se representan como un conjunto o secuencia de dígitos binarios (bits). Los lenguajes de programación de alto nivel nos permiten basarnos en abstracciones para no manejar los detalles de representación interna.

Los tipos de datos simples son: numéricos (enteros y reales), lógicos y caracteres.

Datos numéricos

Este tipo de datos se divide en enteros y reales.

Los enteros son aquellos números que no tienen componente fraccionario o decimal y dentro de la computadora son un subconjunto finito de los números enteros. Estos números pueden ser negativos o positivos y el rango es de –32,768 a 32,767.

El tipo de datos "real" son aquellos números que tienen una parte fraccionaria y pueden ser positivos y negativos dentro de la computadora forman un subconjunto de los números reales. Para representar números muy pequeños o muy grandes se emplea la notación de punto flotante, que es una generalización de la notación científica. En esta notación se considera al número real como mantisa y al exponente la potencia de 10 a la que se eleva este número.

Datos lógicos

Este tipo de dato es aquel que solo puede tomar uno de 2 valores: verdadero (true) o falso (false). En lenguaje C no existe el tipo lógico pero se puede implementar con un número entero conociendo que 0 es falso y cualquier número diferente de cero verdadero.

Caracteres

El dato tipo carácter puede tomar un valor de un conjunto finito y ordenado de caracteres o símbolos que la computadora reconoce (código ASCII). Este tipo de dato ocupa un byte y almacena un solo carácter.

Existe también el dato tipo cadena (compuesto) que es una sucesión de caracteres que se encuentran delimitados por comillas, la longitud de una cadena es el número de caracteres comprendidos entre los delimitadores.

Tipos de datos	Descripción	Memoria
Int	Entero	2 bytes
Char	Carácter	1 byte
Float	Flotante	4 bytes
Double	Flotante de doble precisión	8 bytes

Constantes y variables

Una constante es un dato que permanece sin cambio durante el desarrollo del algoritmo o durante la ejecución del programa. La mayoría de los lenguajes de programación nos permiten el manejo de diferentes tipos de constantes, estas pueden ser enteras, reales, caracteres y cadenas. En lenguaje C una constante se define por medio de la instrucción #define (directiva del procesador).

Una variable es un dato cuyo valor puede cambiar durante el desarrollo del algoritmo o ejecución del programa. Hay diferentes tipos de variables: enteras, reales, caracteres y cadenas. Una variable que es de cierto tipo solo puede tomar valores que correspondan a ese tipo. Si se intenta asignar un valor de tipo diferente se producirá un error.

Una variable se identifica por dos atributos: el nombre de la variable (identificador) y el tipo de la variable. El identificador se puede formar con caracteres alfanuméricos y el carácter de subrayado (_) empezando siempre por una letra. No se admiten como identificadores palabras reservadas del lenguaje de programación que se esté utilizando. Los nombres de variables que se elijan para el algoritmo o programa deben ser significativos y tener relación con el objeto que representa. En lenguaje C la sintaxis para definir una variable es:

```
tipo_de_dato identificador;
```

Contador

Los procesos repetitivos requieren contar los sucesos y acciones internas, una forma de hacerlo es mediante un contador. Un contador es una variable cuyo valor se incrementa o decrementa en una cantidad constante en cada repetición. La forma en que se construye un contador es la siguiente:

```
int contador = 1; //variable con valor inicial de 1
contador = contador+1;
contador += 1;
contador ++;
```

Acumulador

Un acumulador o totalizador es una variable cuya función es almacenar cantidades resultantes de operaciones sucesivas. Realiza la misma función que un contador con la diferencia de que el incremento o decremento es variable en lugar de constante.

```
int acumulador = 0;

acumulador = acumulador + valor;

acumulador += valor;
```

Indicadores o banderas

Una bandera, también denominada interruptor o conmutador es una variable que puede tomar uno de dos valores (verdadero o falso) a lo largo de la ejecución del programa y permite comunicar información de una parte a otra del mismo.

```
int primo;

primo = 0;

primo = 1;
```

Expresiones

Las expresiones son combinaciones de constantes, variables, símbolos de operación (operadores), paréntesis y nombres de funciones especiales. Por ejemplo:

```
sqrt ((p-a)*(p-b)*(p-c));

(a+b+c) / p;
```

Una expresión toma un valor que se determina por el resultado de la ejecución de las operaciones indicadas, tomando los valores de las variables y constantes y aplicando las prioridades de las operaciones.

Según el tipo de operadores que se emplee en la expresión. Estas se clasifican en aritméticas, lógicas, de caracteres o mixtas.

Expresiones aritméticas

Este tipo de expresiones nos sirve para representar formulas matemáticas y utilizan los operadores siguientes:

Operador	Acción
+	Suma
-	Resta
*	Multiplicación
/	División
%	Módulo
++	Incremento
--	Decremento

Las expresiones que tienen 2 o más operadores requieren reglas de precedencia que permiten determinar el orden en que habrán de efectuarse dichas operaciones. Por ejemplo:

```
5 + 8 * 2 //Primero se hace 8 * 2 y luego se suma 5

5+16 = 21

10%3 = 1 //Regresa el residuo de la división entera
```

Nota: En caso de coincidir la prioridad de varios operadores en una expresión el orden en el que se efectúan es de izquierda a derecha.

Expresiones lógicas

Estas emplean los operadores lógicos:

Operador	Acción
Operadores Relacionales	
>	Mayor que
>=	Mayor o igual
<	Menor que
<=	Menor o igual
==	Igual
!=	Diferente
Operadores lógicos	
&&	Y
||	O
!	Negación

Por ejemplo:

```
5>3 && 6>2

año%4==0 && año%100!=0 || año%400==0
```

Nota: Las expresiones lógicas se utilizan para formar condiciones en los programas.

Estructura general de un programa en C

Todo programa en C consta de una o más funciones, una de las cuales se llama main. El programa siempre comenzará por la ejecución de la función main. Las definiciones de las funciones adicionales pueden preceder o seguir a main.

Cada función debe contener:

1. Una cabecera de la función, que consta del nombre de la función, seguido de una lista opcional de argumentos encerrados con paréntesis.
2. Una lista de declaración de argumentos, si se incluyen éstos en la cabecera.
3. Una sentencia compuesta, que contiene el resto de la función.

Los comentarios pueden aparecer en cualquier parte del programa, mientras estén situados entre los delimitadores /* */ o comenzar con // si solo es una línea.

```
/*Estructura de un programa en C*/

#include <nombre_de_la_biblioteca>
#define nombre_de_la_constante valor

//Función principal

void main(){

definición de variables;

instrucciones del programa;

}
```

Ejemplo:

```
/*Primer programa en C*/

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

void main(){

//Imprime el texto en pantalla

printf ("Hola, mundo!");

getch();

}
```

Entrada y salida de datos en C

Como se mencionó anteriormente, es necesario conocer las entradas y las salidas del problema para poder resolverlo. Para que C reciba estos datos utilizamos las funciones de biblioteca scanf para la entrada y printf para la salida. Su sintaxis es la siguiente:

```
//Lee el valor y lo guarda en una variable
scanf ("%d", &variable);

//Imprime en pantalla el texto y el valor
printf ("El valor es %d", variable);
```

Ejemplo:

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

void main() {

int r, a, p;

printf ("Programa que calcula el área y perímetro de un circulo");
printf ("\nIntroduce el radio: ");

scanf ("%d", &r);
```

```
a = M_PI * pow(r,2);

p = M_PI * (r+r);

printf ("\nEl área es: %d", a);
printf ("\nEl perímetro es: %d", p);

getch();
}
```

En ocasiones, queremos que el programa reciba una entrada de tipo caracter. Para lograr esto podemos declarar una variable de tipo carácter y guardarla en getch() o getche(). La diferencia entre ellos es que getche() muestra en pantalla el carácter introducido, mientras que getch() lo mantiene oculto. Su sintaxis es la siguiente:

```
char opc;

opc=getch();

opc=getche();
```

Estructuras de decisión

Estructura de decisión If-Else

Cuando el programador desea especificar en un algoritmo o programa 2 o más caminos alternativos se debe utilizar una estructura de decisión, la cual evalúa una condición y en función del resultado, realiza una parte de la estructura u otra. Las condiciones se especifican mediante expresiones lógicas. Una de las estructuras de decisión es la estructura if-else y su sintaxis es la siguiente:

```
//Si se trata de dos o más instrucciones, estas van entre llaves { }
if (condición)
{
        Instrucciones a ejecutar cuando la condición es verdadera;
}
else
{
        Instrucciones a ejecutar cuando la condición es falsa;
}
```

Estructura de decisión múltiple (switch)

La estructura switch evalúa una expresión que puede tomar n valores distintos, según con cual de estos valores coincida, se ejecutaran ciertas acciones, es decir, el programa o algoritmo seguirá un determinado camino entre los n posibles.

La sintaxis de la estructura switch es:

```
switch (expresión entera) {

case exp_constante_1:
  acciones a realizar cuando la expresión tiene el valor exp_constante_1;
break;
```

```
    case exp_constante_2:
      acciones a realizar cuando la expresión tiene el valor exp_constante_2;
    break;

    ...especificar todos los casos

    default:
acciones a realizar cuando la expresión no coincide con ninguno de los casos;
    break;

    }
```

Estructuras de repetición

Las computadoras están diseñados especialmente para aquellas aplicaciones en las cuales una operación o conjunto de ellas deben de repetirse varias veces. A las estructuras que repiten una secuencia de instrucciones un número determinado de veces se les denomina bucles y se llama iteración al acto de repetir la ejecución de una secuencia de acciones.

Estructuras de control de repetición while y do/ while

En estas estructuras el conjunto de instrucciones que forman parte del bucle se repite mientras se cumple una determinada condición.

La diferencia entre ellas es que la estructura while compruba la condición de continuación del ciclo al principio de este, antes de ejecutar las condiciones del bucle.

La estructura do/ while prueba la condición después de ejecutar las instrucciones del bucle y por lo tanto esta se ejecutará por lo menos una vez.

La sintaxis de la estructura while es:

```
while (condición)
{
     Instrucciones que se repetirán mientras la condición sea verdadera;
}
```

La sintaxis de la estructura do/ while es:

```
do
{
    Instrucciones que se repetirán al menos una vez mientras la condición
    sea verdadera;
}
while (condición);
```

Estructura de control de repetición for

La estructura for maneja de manera automática todos los detalles de repetición controlada por un contador. La sintaxis de la estructura for es:

```
for (contador; condición; variación)
{
    Secuencia de instrucciones que se desean repetir;
}
```