El lenguaje C es un lenguaje de programación de propósito general, es uno de los más rápidos y potentes que existen. El lenguaje C ha demostrado ser un lenguaje extremadamente eficaz, hasta como para crear sistemas operativos.

* **Es un lenguaje Compilado.** A diferencia de otros lenguajes, que son lenguajes interpretados, los cuales necesitan del código fuente para funcionar (por ejemplo Basic), C es un lenguaje compilado esto quiere decir que convierte el código fuente en un fichero objeto, éste es enlazado con las librerías necesarias dando lugar a un fichero ejecutable.
* **Es un lenguaje de Nivel medio.** Esto quiere decir que combina elementos de lenguaje de alto nivel con la funcionalidad del lenguaje ensamblador, o sea que trabaja a un nivel cercano al computador, sin embargo, nos ofrece posibilidades de construir estructuras de datos equivalentes a los que manejan los lenguajes de alto nivel.
* **Es un lenguaje Estructurado.** Esto quiere decir que permite crear procedimientos en bloques dentro de otros procedimientos.
* **Es un lenguaje Portable.** Este lenguaje permite utilizar el mismo código en diferentes equipos y sistemas informáticos, o sea que es independiente de la arquitectura de cualquier máquina.
* **Es un lenguaje Relativamente Pequeño.** Este es un lenguaje económico en cuanto a expresiones se refiere, se puede describir en poco espacio y es fácil y rápido de aprender.
* **Tiene abundancia de Operadores y Tipos de Datos.** Este lenguaje prácticamente posee un operador para cada una de las posibles operaciones en código de máquina.

Pero al igual que otros lenguajes, el C tiene sus desventajas: .

* Carece de instrucciones entrada/salida.
* Carece de instrucciones de manejo de cadena de caracteres.
* La precedencia de los operadores dificultan la comprensión de algunas expresiones.

Todo programa en C, se compone de una o más funciones. La función que se ejecuta en primera instancia es una función llamada main( ), esta función es la más importante de todo el programa y es la que nunca debe faltar. A esta función no se le puede cambiar el nombre; main( ) es el cuerpo principal de nuestro código fuente. A su vez toda función del programa debe devolver un valor al programa, este valor se indica con la palabra reservada "return". Ciertas veces no queremos que la función devuelva ningún valor para estos casos simplemente le indicamos "return 0" o escribimos la palabra reservada "void" antes del nombre de la función.

**Tipos de Datos**

*Tipo* **int** /\* número entero \*/

En una variable de este tipo se almacenan números enteros (sin decimales). El rango de valores que admite es -32768 a 32767. Cuando definimos una variable lo que estamos haciendo es decirle al compilador que nos reserve una zona de la memoria para almacenar datos de tipo int.

*Tipo* **char** /\* caracter \*/

Las variables de tipo char sirven para almacenar caracteres. Los caracteres se almacenan en realidad como números del 0 al 255. Los 128 primeros (0 a 127) son el ASCII estándar. El resto es el ASCII extendido y depende del idioma y del ordenador. Para declarar una variable char solo escribimos: char letra ; y listo, ya existe una variable *"letra"* que almacenará datos de tipo caracter.

*Tipo* **float** /\* número real, punto flotante de simple precisión\*/

En este tipo de variable podemos almacenar números decimales, no sólo enteros como en los anteriores. El rango de posibles valores es del 3.4 E-38 al 3.4 E+38.

Para declarar un float, de forma similar, escribimos: float prom ; y con esto hemos declarado una variable de nombre *"prom"* que tendrá un dato de tipo float o coma flotante.

*Tipo* **double** /\* número real, punto flotante de doble precisión\*/

En las variables tipo double se almacenan números reales, estos tiene el rango de 1.79769 E-308 a 1.79769 E+308. Al igual que las anteriores para declararla escribimos: double num ; y tenemos una variable de nombre *"num"* que aceptará datos tipo double.

*Modificador* **unsigned** /\* sin signo\*/

Este modificador (que significa: sin signo) modifica el rango de valores que puede contener una variable. Sólo admite valores positivos. Este modificador se usa así: unsigned int total ; esto declara un variable de nombre "total", de tipo entero positivo.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tipo de Dato** | **Dato almacenado** | **Tamaño**  **(en bytes)** | **Rango** | **Observaciones** |
| **unsigned char** | Caracteres | 1 | 0 a 255 | Valores correspondientes a números ordinales de los 256 caracteres ASCII |
| **char** | Caracteres | 1 | -128 a 127 | char letra = 'A'; ó letra = 65; |
| **short int** | Entero | 2 | -32768 a 32767 | Proporciona un entero en el rango dado |
| **unsigned int** | Entero | 2 | 0 a 65535 | Un entero positivo |
| **int** | Enero | 2 | -32768 a 32767 | Equivalente al tipo short en máquinas de 16 bits |
| **long int** | Entero | 4 | -2147483648 a  2147483647 | tamaño(int) <= tamaño(long) |
| **unsigned long** | Entero | 4 | 0 a 4294967295 | entero largo sin signo |
| **float** | Números Reales | 4 | -3.4 E-38 a  3.4 E+38 | no tiene más de 5 dígitos significativos |
| **long double** | Números Reales (coma flotante  con doble precisión) | 10 | 3.4 E-4932  a  1.18 E+4932 | Puede tener hasta 19 dígitos significativos |
| **double** | Números Reales (coma flotante  con doble precisión) | 8 | 1.79769 E-308  a 1.79769 E+308 | Tiene hasta 16 dígitos significativos |
| **definido por el usuario** | Usando estructuras | xx | depende del tipo de dato | XXXXXXXXXXXXXX |

***scanf ( )*** /\* leer desde el teclado\*/

Al utilizar la palabra scanf(), nosotros podemos hacer que el usuario introduzca datos por el teclado. El uso de scanf() es de la siguiente manera: scanf("%i", &num); esta última expresión indica al programa que se debe solicitar un número entero (%i, es una especificación de formato y esto lo puedes ver en la sección de operadores y manipuladores), luego se indica el nombre de la variable en la que se va a guardar el dato, éste (el nombre) precedido por el signo ‘&’, que indica al programa la dirección de ésta variable.

***printf ( )*** /\* mostrar por pantalla\*/

El uso de printf(), es similar al de scanf(), lo que scanf() hace para leer una variable, printf() lo hace para imprimirla. De esta manera podemos imprimir en pantalla, ya sea un mensaje o el valor contenido en una variable. La sintaxis de printf() es así: printf("%d", num); esta sentencia lo que hace es imprimir el valor de la variable num en el lugar donde aparece el "%d". El printf() nos sirve también para mandar a imprimir en pantalla un mensaje así:

printf ("Bienvenidos al mundo de C") ;

**Indicadores de formato y secuencias de escape**

|  |  |
| --- | --- |
| ***%d*** | Número entero con signo, en notación decimal |
| ***%i*** | Número entero con signo, en notación decimal |
| ***%u*** | Número entero sin signo, en notación decimal |
| ***%o*** | Número entero sin signo, en notación octal (base 8) |
| ***%x*** | Número entero sin signo, en hexadecimal (base 16) |
| ***%X*** | Número entero sin signo, en hexadecimal, mayúsculas |
| ***%f*** | Número real, float (coma flotante, con decimales) |
| ***%e*** | Número real en notación científica |
| ***%g*** | Usa el más corto entre %e y %f |
| ***%c*** | Un único carácter |
| ***%s*** | Cadena de caracteres |
| ***%%*** | Signo de tanto por ciento: % |
| ***%p*** | Puntero (dirección de memoria) |
| ***%ld*** | Número entero largo con signo, long int (long) |
| ***%lf*** | Número real, double (coma flotante, con decimales) |

**Secuencias de escape**

|  |  |
| --- | --- |
| **Código** | **Significado** |
| “ \n “ | nueva línea |
| “ \r " | retorno de carro |
| " \f " | nueva página |
| " \t " | tabulador horizontal |
| " \b " | retroceso (backspace) |
| “ \’ “ | comilla simple |
| " \" " | Comillas |
| " \\ " | Barra |

**getch( ) *-* getche( ) – getc( ) – getchar( ) – gets( )**

Si lo que queremos es que el usuario introduzca un caracter por el teclado usamos las funciones getchar( ), getc( ), getch( ) y getche( ). Estas esperan a que el usuario introduzca un carácter por el teclado. La diferencia entre getc( ), getche( ), getchar( ) y getch( ) es que las tres primeras imprimen en pantalla la tecla que hemos pulsado y la última no.

**Operadores Lógicos**

|  |  |
| --- | --- |
| **Operador** | **Operación** |
| && | AND, en español Y. Da como resultado el valor lógico 1, si ambos operadores son distintos de cero |
| || | OR, en español O. El resultado es cero, si ambos operandos son cero |
| ! | NOT, en español NO. El resultado es cero si el operando tiene un valor distinto de cero |

**Operadores Aritméticos**

|  |  |
| --- | --- |
| **Operador** | **Operación** |
| + | Suma dos numeros, pueden ser enteros o reales |
| - | Resta dos números, pueden ser enteros o reales |
| \* | Multiplica dos números, pueden ser enteros o reales |
| / | División, el resultado depende de los tipos de datos |
| % | Módulo o resto de una división entera, los operandos tienen que ser enteros |
| ++ | Incrementa en uno la variable |
| -- | Decrementa en uno la variable |

**Operadores de Relación**

|  |  |
| --- | --- |
| **Operador** | **Operación** |
| < | Primer operando menor que el segundo |
| > | Primer operando mayor que el segundo |
| <= | Primer operando menor o igual que el segundo |
| >= | Primer operando mayor o igual que el segundo |
| == | Primer operando igual que el segundo |
| != | Primer operando distinto del segundo |

**Operadores de Asignación**

|  |  |
| --- | --- |
| **Operador** | **Operación** |
| = | Asignación simple |
| \*= | Multiplicación más asignación |
| /= | División más asignación |
| %= | Módulo más asignación |
| += | Suma más asignación |
| -= | Resta más asignación |

**Niveles de Precedencia**

|  |
| --- |
| **Operador** |
| ( )  [ ] |
| !  ++  -- |
| \*   /  % |
| +  - |
| ==  != |
| ! |
| && |
| || |
| =  \*=  /=  %=  +=  -= |

***Operadores avanzados***

Los operadores de **incremento, decremento** y **asignación compuesta** permiten modificar el contenido de una variable de forma eficiente y abreviada.

**Operadores Significado** A++, ++A Incrementa en 1 el valor de A (A=A+1) A--, --A Disminuye en 1 el valor de A (A=A-1)

A+=x A=A+x A-=x A=A-x A\*=x A=A\*x A/=x A=A/x

***Operadores “pre” y “post” y valor devuelto***

Si el operador **++** o **--** se coloca a la izquierda, se llama **preincremento** o **predecremento**, respectivamente. Si se coloca a la derecha, se llama **postincremento** o **postdecremento**.

***Bucles***

Los bucles nos ofrecen la solución cuando queremos repetir una tarea un número determinado de veces. Supongamos que queremos escribir 100 veces la palabra Bienvenido.

Con lo que sabemos, tendríamos que escribir 100 veces la expresión: printf ("Bienvenido\n"); , sin embargo un ciclo for nos facilitaría en grande esta tarea. Estos bucles pueden ser: ***for*** (para), ***while*** (mientras), ***do while*** (hacer-mientras).

*Sentencia* ***if***

La palabra if significa si (condicional), su sintaxis es como sigue:

|  |
| --- |
| if ( condición )  {     // instrucciones a ejecutar } |

Cuando se cumple la condición comprendida entre los paréntesis se ejecuta el bloque de sentencias inmediatamente siguiente al if.

Ejemplo:

|  |
| --- |
| int i = 0; printf("Ingrese un valor entero: "); scanf("%d",&i); if (i < 10) {    print("El valor ingresado es mayor es menor a 10");  } |

*Sentencia* ***if-else***

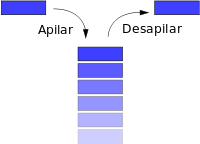
En el if, si la condición no se cumplía, entonces las instrucciones no se ejecutaban y el programa finalizaba o seguía con las siguientes instrucciones. En el if-else, (si-sino (en pseudocódigo)) si no se cumple la condición entonces se realizarán otras instrucciones (las del else).

Su sintaxis es la siguiente:

|  |
| --- |
| if ( condición ) {    // bloque que se ejecuta si se cumple la condición  } else {    // bloque que se ejecuta si no se cumple la condición  } |
|  |

**¿Qué es una pila?**

Una pila (*stack* en inglés) es una estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés *LastIn FirstOut*, último en *entrar*, primero en *salir*) que permite almacenar y recuperar datos. Esta estructura se aplica en multitud de ocasiones en el área de informática debido a su simplicidad y ordenación implícita de la propia estructura.



Para el manejo de los datos se cuenta con dos operaciones básicas:

* **apilar** (*push*), que coloca un objeto en la pila.
* **desapilar** (*pop*), que retira el último elemento apilado.

En cada momento sólo se tiene acceso a la parte superior de la pila, es decir, al último objeto apilado (denominado **tope de pila**). La operación **desapilar**  permite la obtención de este elemento, que es retirado de la pila permitiendo el acceso al siguiente (apilado con anterioridad), que pasa a ser el nuevo **tope de pila**.

**Operaciones**

Una pila cuenta con 2 operaciones imprescindibles: apilar y desapilar, a las que en las implementaciones modernas de las pilas se suelen añadir más de uso habitual.

* **Crear:** se crea la pila. (constructor)
* **Inicializar:** se deja la pila vacía, lista para ser utilizada.
* **Apilar:** se añade un elemento a la pila.
* **Leer:** agrega al tope de la pila un elemento ingresado desde el teclado por el usuario.
* **Desapilar:** se elimina el elemento superior de la pila.
* **Tope:** devuelve el elemento que está en la cima de la pila. No lo elimina.
* **PilaVacía:** devuelve cierto si la pila está vacía o falso en caso contrario.
* **Mostrar:** muestra por pantalla el contenido de la pila.