Lenguaje C

Manual de ayuda

Por: Luis Zambrano G.

Inicio: Domingo 25 de febrero de 2024

Fin: Sábado 9 de marzo de 2024

Contenido

Parte I

INTRODUCCIÓN BÁSICA AL LENGUAJE C

Capítulo 1

Variables

Las variables son lo más importante en el lenguaje de programación, ya que almacena algún tipo de dato y tiene un identificador y tipo.

En el lenguaje C las variables deben de empezar siempre con una letra y pueden continuar con más letras, números o sub guion, las variables que utilizan como primer carácter el sub guion si están permitidas, pero se las utiliza para el sistema porque muchas veces el compilador de C las utiliza, ya que es para poderlas diferenciar de las variables del código hecho por nosotros.

Antes de usar una variable siempre se debe de declarar primero y debe de ir al principio del programa, ya que es buena práctica de programación, esto significa que al declararse una variable se le indica al compilador de C que se va a utilizar dicha variable.

Las palabras que no se pueden utilizar como variables son las reservadas por el lenguaje C, como se muestra en siguiente enlace: ***Tabla 1.1***

Las variables tienen los siguientes atributos:

- **Nombre** - Es el nombre cuando la declaramos
- **Dirección** - Es donde se encuentra
- **Valor** - Es el contenido literal
- **Tipo** - Es lo que se distingue entre variables
- **Tiempo de vida** - Es hasta cuando se puede utilizar
- **Ámbito** - Es dónde se puede utilizar

1.1. Tipos

En esta sección se encuentran los tipos de variables utilizados en el lenguaje C.

- **char** - Carácter, 1 byte, [-128, 127]
- **int** - Entero, 2 o 4 byte, [-2147483648, 2147483647]
- **float** - Real simple, 4 bytes, $[-3.4^{-38}, 3.4^{38}]$
- **double** - Real doble, 8 bytes, $[-1.7^{-308}, 1.7^{308}]$
- **void** - Sin valor, 0 bytes, Sin valor
- * - Puntero, guarda direcciones de las variables
- **NULL** - Puntero nulo, no apunta a ninguna dirección
- **void** * - Puntero genérico, se utiliza cuando no se sabe el tipo de dato que va apuntar

1.2. Modificadores de tipos

Es agregarle una característica extra al tipo de dato.

- **short** - Aplicables a enteros, 2 byte, [-32768, 32767]
- **long** - Aplicables a enteros y reales, duplica el rango del tipo, ejemplo: long int (4 bytes es igual al int), long long int (8 bytes), long double (12 bytes), long long double

1.3. Modificadores de signos

Por defecto es signed.

- **signed** - Signo por defecto
- **unsigned** - Enteros sin signos, 4 bytes, [0, 4294967296]

1.4. Palabras Reservadas

Tabla 1.1: Palabras Reservadas

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

Capítulo 2

Comentarios

Se utilizan como ayuda para que el programador tenga una explicación de algún comando, función, etc.

2.1. De 1 Línea

/* Comentario */ - Comentarios de una línea

2.2. Multilíneas

/* Comentarios de varias líneas*/ - La última línea se cierra con */

Capítulo 3

Operadores

Sirven para operar variables.

3.1. Aritméticos

Son para hacer operaciones aritméticas.

- **+:** Suma las variables de tipo entero y real, ejemplo: $a + b$;
- **-:** Resta las variables de tipo entero y real, ejemplo: $a - b$;
- ***:** Producto, multiplica las variables de tipo entero y real, ejemplo: $a * b$;
- **/:** Cociente, divide las variables de tipo entero y real, ejemplo: a/b;
- **%:** Resto, se obtiene solo de la división en las variables de tipo entero, ejemplo:$a\,\%b$;

La prioridad de estos operadores es la siguiente:

1. **+, -:** Como operador unitario, ejemplo: +25; Indica que 25 es positivo
2. ***, /, %:** cuando se opera más de 2 variables
3. **+, -:** cuando se opera más de 2 variables

3.2. De dirección de memoria

Sirven para trabajar con punteros.

- **&:** Se obtiene la dirección de memoria, comando de ejemplo: #

3.3. Relacionales

Se utilizan para comparar elementos entre sí.

- **==:** Igual a, comando de ejemplo: a == b;

- **!=:** No es igual a, sentencia: a != b;
- **>:** Mayor que, comando de ejemplo: a > b;
- **<:** Menor que, sentencia: a < b;
- **>=:** Mayor o igual que, comando de ejemplo: a >= b;
- **<=:** Menor o igual que, sentencia: a <= b;

3.4. Lógicos

Sirven para obtener un valor de verdad sea verdadero o falso dependiendo de sus cláusulas.

- **!:** Negación, comando de ejemplo: !(a >= b);
- **&&:** Y lógica, sentencia: (a < b) && (c > d);
- **||:** O lógica, sentencia: (a < b) || (c > d);

El siguiente orden es de mayor a menor prioridad: !, &&, ||

3.5. A nivel de bits

Sirven para operar a nivel de bits.

- ***¡¡¡CONSULTAR!!!***

3.6. Asignación

Sirven para asignar valores a las variables.

- **=:** Asigna el valor de b a la variable a: (a = b);
- ***=:** Multiplica a por b y el valor lo asigna a la variable a: (a *= b);
- **/=:** Divide a por b y el valor lo asigna a la variable a: (a /= b);
- **%=:** Calcula el resto de la división y el valor lo asigna a la variable a: (a%= b);
- **+=:** Suma a por b y el valor lo asigna a la variable a: (a += b);
- **-=:** Resta a por b y el valor lo asigna a la variable a: (a-= b);

3.7. Prioridad de operadores

Convenio de qué se resuelve primero cuando hay varias operaciones.

1. **Matemáticos**
2. **Relacionales**
3. **Lógicos**

3.8. Incremento y decremento

Sirven para incrementar o decrementar en una unidad la variable.

- **a++:** Pos incremento, primero asigna y después suma en una unidad.
- **++a:** Pre incremento, primero suma en una unidad y después asigna.
- **a—:** Pos decremento, primero asigna y después resta en una unidad.
- **—a:** Pre decremento, primero suma en una unidad y después asigna.

Capítulo 4

Estándar de entrada y salida de datos

Sirven para ingresar o mostrar datos.

4.1. #include <stdio.h>

Es una librería para entrada por teclado y salida por pantalla de los datos.

- **stdin:** Suma las variables de tipo entero y real, ejemplo: $a + b;$
- **stdout:** Resta las variables de tipo entero y real, ejemplo: $a - b;$

Salida estándar: printf(CadenaDeControl, dato1, dato2, ...);

- **CadenaDeControl:** Contiene los tipos de datos y formato para mostrar el mensaje
- **dato1, dato2, ...:** Contiene las variables, constantes y datos de salida

Entrada estándar: scanf(CadenaDeControl, var1, var2, ...);

- **CadenaDeControl:** Contiene los tipos de datos a ingresar
- **var1, var2, ...:** Variables del tipo de los códigos de control

4.2. Código de formato

Sirven para convertir los datos que salen por pantalla.

- **%d:** El dato se convierte a entero decimal
- **%o:** El dato se convierte a octal
- **%x:** El dato se convierte a hexadecimal
- **%u:** El dato se convierte a entero sin signo

- **%c:** El dato se convierte a carácter
- **%e:** El dato se considera de tipo float, se convierte a notación científica
- **%f:** El dato se convierte a float
- **%g:** El dato se convierte a float, se convierte a %e o a %f
- **%s:** El dato a mostrar es una cadena de caracteres
- **%lf:** El dato se considera double
- ***¡¡¡CONSULTAR!!!***: El dato se convierte a binario

4.3. Secuencia de escape

Sirven para mostrar caracteres especiales.

- **a:** Alarma
- **b:** Retroceso de espacio
- **f:** Avance de página
- **n:** Retorno de carro y avance de línea
- **r:** Retorno de carro
- **t:** Tabulación de 4 espacios
- **v:** Tabulación vertical
- **:** Barra inclinada
- **?:** Signo de interrogación
- **":** Doble comillas
- **000:** Número octal
- **xhh:** Número hexadecimal
- **0:** Cero, nulo

4.4. Directivas del preprocesador

El preprocesador es aquel primer programa que se ejecuta al empezar a compilar.

Las directivas no son sentencias en C.

Para escribir en el preprocesador todas las directivas estas empiezan con # y no terminan con punto y coma.

La directivas más utilizadas don: #include y #define

La directiva #include sirve para incluir en nuestro programa librerías o código externo.

Si deseamos incluir librerías propias de C este es el formato: #include <nombreDelArchivo>

Si deseamos incluir librerías nuestras o de terceros este es el formato: #include "nombreDelArchivo".
Tener en cuenta que si el archivo está en la carpeta del proyecto se lo puede escribir directamente de lo contrario se debe poner la ruta.

La directiva #define sirve para definir macros para valores constantes u operaciones y este es el formato: #define MACRO valor

Las macros como buena práctica de programación se las debe declarar en mayúsculas.

Capítulo 5

Estructuras de control

Permiten modificar el flujo del programa.

Existen 3 estructuras de control básicas.

- **Secuencia:** Se ejecutan los comandos uno tras otro.
- **Selección:** También llamada decisión, y de acuerdo a la condición cambia el flujo del programa.
- **Repetición:** Es un bucle que ejecuta los comandos hasta que la condición sea falsa.

5.1. Sentencias

Son los comandos que ejecuta el programa y pueden ser simples o compuestas.

Sentencias simples: Son las que finalizan con punto y coma, por ejemplo: printf("Hola Mundo");

Sentencias compuestas: Son el conjunto de sentencias simples encerradas entre llaves, ejemplo:

```
{
 sentencia1;
 sentencia2;
}
```

5.2. Selección if

Es la principal sentencia de selección en C, ejemplo:

```
if(condición)
 sentencia
```

5.3. Selección if - else

Si no se cumple la condición cambia el flujo del programa, ejemplo:

```
if(condición)
 sentencia
else
 sentecia
```

5.4. Selección anidada

Es cuando la estructura de control tipo selección tiene dentro del cuerpo una o más estructuras.

5.5. Ejercicio

Hacer un programa que ordene tres números.

```c
#include <stdio.h>

int main(void){
  int numero1;
  int numero2;
  int numero3;

  printf("====== PROGRAMA QUE ORDENA ASCENDENTEMENTE TRES N%cMEROS ======\n\n", 223);
  printf("Ingrese el primer n%cmero: ", 163);
  scanf("%d", &numero1);
  printf("\n");
  printf("Ingrese el segundo n%cmero: ", 163);
  scanf("%d", &numero2);
  printf("\n");
  printf("Ingrese el tercer n%cmero: ", 163);
  scanf("%d", &numero3);
  printf("\n");
  if((numero1 == numero2) && (numero1 == numero3)){
    printf("Los tres n%cmeros son iguales y su valor es: %d", 163, numero1);
  }else{
    if((numero1 < numero2) && (numero1 < numero3)){
      if(numero2 < numero3){
        printf("El orden es: %d, %d, %d", numero1, numero2, numero3);
      }else{
        printf("El orden es: %d, %d, %d", numero1, numero3, numero2);
      }
    }else{
      if(numero2 < numero3){
        if(numero1 < numero3){
          printf("El orden es: %d, %d, %d", numero2, numero1, numero3);
        }else{
          printf("El orden es: %d, %d, %d", numero2, numero3, numero1);
        }
      }else{
        if(numero1 < numero2){
          printf("El orden es: %d, %d, %d", numero3, numero1, numero2);
        }else{
          printf("El orden es: %d, %d, %d", numero3, numero2, numero1);
```

```
        }
      }
    }
  }
}
```

Programa en C 5.1: Ordena tres números de forma ascendente

5.6. Selección switch

Se utilizan para seleccionar una de entre múltiples alternativas.

Su selección se basa en el valor de una variable o de una expresión simple.

La sintaxis se muestra a continuación:

```
switch(selector){
 case etiqueta: sentencias; break;
 case etiqueta: sentencias; break;
 ...
 default: sentencias; /* OPCIONAL*/
}
```

Tener en cuenta que el selector solo puede ser de tipo int o char.

5.7. Ejercicio

Hacer un programa que muestre un mensaje dependiendo del tipo de nota, si es A sobresaliente, B Excelente, C muy bueno, D pasable, E malo, F muy malo.

```
#include <stdio.h>

int main(void){
  char nota;

  printf("====== PROGRAMA QUE MUESTRA UN MENSAJE DEPENDIENDO DE LA NOTA ======\n\n");
  printf("Ingrese la nota seg%cn el rango (A-F): ",163);
  scanf("%c", &nota);

  switch(nota){
    case 'A': case 'a':
      printf("Su nota es SOBRESALIENTE.\n");
      break;
    case 'B': case 'b':
      printf("Su nota es EXCELENTE.\n");
      break;
    case 'C': case 'c':
      printf("Su nota es MUY BUENO.\n");
      break;
    case 'D': case 'd':
      printf("Su nota es PASABLE.\n");
      break;
    case 'E': case 'e':
      printf("Su nota es MALA.\n");
```

```
25          break;
26        case 'F': case 'f':
27          printf("Su nota es MUY MALA.\n");
28          break;
29        default:
30          printf("La letra de la nota ingresada no es v%clida.\n", 160);
31      }
32    }
```

Programa en C 5.2: Mostrar mensaje según la nota

5.8. Repetición while

Se utilizan para repetir comandos hasta que la condición sea falsa.

Se lo conoce como ciclo condicionado.

Al ciclo while se lo conoce como pre chequeo, porque primero chequea la condición y después ejecuta los comandos.

En la condición del ciclo hay una o más variables involucradas.

El ciclo while puede que no se ejecute si de entrada la condición es falsa.

Las variables involucradas en la condición deben pasar por las siguientes etapas.

1. **Inicialización:** Todas las variables tienen que haber sido inicializadas.
2. **Prueba/condición:** Primero hace un chequeo del valor que está en la condición y después ejecuta los comandos.
3. **Modificación:** Estar atento que al menos una variable de la condición debe de modificarse y se debe de asegurar que no entre en un ciclo infinito, tener mucho cuidado para no crear ciclos infinitos.

La sintaxis se muestra a continuación con una sentencia:

```
while(condición)
 sentencia;
```

1em La sintaxis de varias sentencias se muestra a continuación:

```
while(condición){
 sentencias1;
 sentencias1;
 ...
}
```

5.9. Ejercicio

Solicitar al usuario el ingreso de varios números enteros, termina el programa cuando se hayan ingresado 5 números pares.

```c
#include <stdio.h>

int main(void){
  int numero;
  int contador = 0;
  int numerosParesIngresados = 0;

  printf("====== PROGRAMA QUE SOLO PERMITE INGRESAR CINCO NUMEROS PARES ======\n\n");

  while(numerosParesIngresados < 5){
    printf("Hay %d n%cmeros pares ingresados y se han ingresado en total %d n%cmeros, ingrese el siguiente n%cmero: ", numerosParesIngresados, 163, contador, 163, 163);
    scanf("%d", &numero);
    if((numero % 2) == 0){
      numerosParesIngresados++;
    }
    contador++;
    printf("\n");
  }
}
```

Programa en C 5.3: Ingreso de cinco números pares

5.10. Repetición for

Se utilizan para repetir comandos hasta que la condición sea falsa y es también conocido como el ciclo incondicionado.

Se lo utiliza cuando se requiere repetir comandos un número fijo de veces.

La sintaxis se muestra a continuación:

```
for(inicialización; condición/iteración; incremento)
sentencias
```

1em Las partes se muestran a continuación:

- **Inicialización:** Inicializa las variables de control del bucle, se pueden utilizar una o más variables de control.
- **Condición:** Contiene la expresión lógica donde hace que el ciclo realice las iteraciones mientras esta condición sea verdadera.
- **Incremento:** incrementa o decrementa las variables de control.
- **Sentencias:** Son las instrucciones que se repetirán.

5.11. Ejercicio

Solicitar al usuario N números enteros positivos y mostrar su raíz cuadrada.

```c
#include <stdio.h>

int main(void){
  int cantidadDeRaices;
  int i;
  unsigned int valorDelNumero;

  printf("====== PROGRAMA QUE MUESTRA LA RAIZ CUADRADA DE LOS NUMEROS ENTEROS POSITIVOS ======\n\n");
  printf("Cu%cntos n%cmeros enteros positivos desea saber su ra%cz cuadrada: ", 160, 163, 161);
  scanf("%d", &cantidadDeRaices);

  for(i = 1; i <= cantidadDeRaices; i++){
    printf("\n");
    printf("Ingrese el valor #%d: ", i);
    scanf("%d", &valorDelNumero);
    printf("La ra%cz cuadrada de %d es: %.2f", 161, valorDelNumero, sqrt(valorDelNumero));
  }
}
```

Programa en C 5.4: Raíz cuadrana de N números positivos

5.12. Repetición do - while

El ciclo do - while, se ejecuta al menos una vez.

Este ciclo primero se ejecuta y después chequea la condición.

La sintaxis se muestra a continuación:

```
do
 sentencias
while(expresión);
```

1em Tener en cuenta que termina el comando con punto y coma.

5.13. Ejercicio

Dadas las edades de N personas se desea saber la edad promedio.

```c
#include <stdio.h>

int main(void){
  int cantidadDePersonas;
  int i;
  int edad;
  int sumaDeEdades = 0;

  printf("====== PROGRAMA QUE MUESTRA EL PROMEDIO DE EDADES ======\n\n");

  printf("Cu%cntas edades va a ingresar: ", 160);
```

```
12      scanf("%d", &cantidadDePersonas);
13
14      for(i = 1; i <= cantidadDePersonas; i++){
15        do{
16          printf("El rango de edad es de 0 - 100, ingrese la edad # %d: ", i);
17          scanf("%d", &edad);
18        }while((edad < 0) || (edad > 100));
19        sumaDeEdades += edad;
20      }
21
22      printf("El promedio de edad de las %d personas es: %d", cantidadDePersonas, sumaDeEdades /
        cantidadDePersonas);
23    }
```

Programa en C 5.5: Raíz cuadrana de N números positivos

Parte II

INTERMEDIO

Capítulo 6

Introducción a la modularización

Módulo es cada una de las partes que resuelve un subproblema del problema.

Cada módulo tiene una tarea bien definida.

Los módulos para resolver su problema pueden ser usados por otros tipos de módulos.

Un módulo se puede comunicar con otro a través de una interfaz asignada a la comunicación donde debe de estar bien definida.

Existen dos tipos de módulos que son: funciones y procedimientos.

La modularización utiliza aquella técnica llamada divide y vencerás.

Para la resolución de un problema se descompone en módulos.

Cada módulo se divide en nuevos módulos hasta que el problema se reduce a actividades básicas.

Cuando se termine cada módulo se van fusionando y conectando los módulos hasta resolver el problema original.

6.1. Ventajas de modularizar código

1. Reutilización de código
2. Ayuda al desarrollo y codificación ágil de los códigos en menos tiempo y poco esfuerzo
3. Mejor modelización de los problemas
4. Mayor legibilidad
5. Etc.

Capítulo 7

Funciones

Una función es un grupo de sentencias que realizan una tarea concreta y como resultado de ella retorna un único valor como respuesta, caso contrario si devuelve cero o más de un valor no es función.

El retorno de la función se guarda en el nombre de esta, conceptualmente se puede decir o afirmar que el nombre de la función tiene asignado un lugar en la memoria, dicho de otra forma es parecida a como funciona una variable, pero en el ámbito estricto y riguroso no es variable.

Al declarar una función el compilador de C, guarda un espacio de memoria para el nombre de la función.

7.1. Declaración de funciones

A continuación se muestra la sintaxis para declarar una función:

```
tipoDeRetorno nombreDeLaFunción(listaDeParámetros){
 cuerpoDeLaFunción
 return (espresión);
}
```

A continuación se muestra una función que suma dos números y devuelve el resultado:

```
1  int suma(int num1, int num2){
2    int resultado;
3
4    resultado = num1 + num2;
5    return resultado;
6  }
```

Programa en C 7.1: Función que suma dos números y devuelve el resultado