



Primeros Algoritmos



Primeros Algoritmos

Al igual que en el Ingreso en C las sentencias repetitivas, el if, los operadores de comparación son exactamente iguales (su estructura) que en Javascript.

Vamos a hacer un repaso de cada uno de ellos.

Sentencia IF

Se usa para tomar decisiones está evaluá básicamente una operación lógica, es decir una expresión que da como resultado verdadero o falso, y ejecuta la pieza de código siguiente siempre y cuando el resultado sea verdadero.

En C a diferencia de Javascript no tendremos el valor booleano, por ende una operación será falsa cuando de como resultado un cero, mientras que será verdadera cuando el valor sea distinto a cero.

IF

El if lo declaramos de la siguiente forma:

```
if(condicion)
{
    //SI EL VALOR DE LA CONDICION ES CERO NO SE EJECUTA
    //SI EL VALOR DE LA CONDICION ES DISTINTO A CERO, SE EJECUTA
}
```

En donde si la condición (que puede ser una variable, un número o una operación relacional) da como resultado **cualquier número distinto a cero**, se ejecutará el código que está dentro de las llaves {}, mientras que si da **un cero** el código dentro de las llaves no se ejecutará.

Operaciones Lógicas

Hay dos tipos de operaciones que vamos a ver que se pueden hacer.

Intersección: Una de ellas es la Intersección o comunmente conocida como AND , se la declara usando los simbolos `&&` y sólo dara cómo true si es que las dos operaciones que se comparen dan true, sino dara como resultado false.

Unión: La otra de ellas es la unión o comunmente conocida como OR , se la declara usando los simbolos `||` y sólo dara cómo false si es que las dos operaciones que se comparen dan false , sino dara como resultado true siempre.

Tabla de Verdad

operación intersección o "and":

Variable 1		Variable 2		Resultado
false	&&	false	=	false
false	&&	true	=	false
true	&&	false	=	false
true	&&	true	=	true

operación unión o "or":

Variable 1		Variable 2		Resultado
false		false	=	false
false		true	=	true
true		false	=	true
true		true	=	true

Operadores Relacionales

Existen ciertos símbolos que nos permiten hacer comparaciones, estas operaciones dan como resultado 1 o 0

Vamos a ver las siguientes:

- Mayor ($>$)

- Menor ($<$)

- Igual($==$)

- Distinto($!=$)

- Mayor igual ($>=$)

- Menor igual ($<=$)

Else

Cuando el if resulte falso (valor cero) aveces necesitamos realizar algún tipo de acción.

Para eso usamos la sentencia else que simplemente se ejecuta si es que el if dio 0.

Para declarar un else necesitamos declarar un if primero, no existe else sin if, pero si puede existir if sin un else.

if/else/if/else

Dentro de cada if, podemos escribir más if, o incluso if/else. Lo mismo en los else, también podemos escribir la cantidad de if que necesitemos.

No se aconseja usar else if en una misma línea ya que es más prudente a errores.

En la siguiente diapositiva verán un ejemplo de if/else anidados y la forma correcta de hacerlos

Ejemplo en código

```
if(condicion)
{
    if(condicion)
    {
        if(condicion)
        {
        }
    }
    else
    {
    }
}
else
{
    if(condicion)
    {
    }
}
```

Forma correcta

```
if(condicion)
{
    if(condicion)
    {
        if(condicion)
        {
        }
    }
    else
    {
    }
}
else if(condicion)
{
    {
    }
}
```

Forma no recomendada



Sentencia SWITCH

SWITCH

Esta estructura permite considerar decisiones para más de dos posibilidades.

A diferencia del if, está sólo se utiliza para verificar la igualdad entre todas las posibilidades que pongamos.

Evalúa siempre una condición (En C sólo evalúa int y char) ya que los valores string no existen

SWITCH

```
switch(variable)
{
    case 1:
        printf("Hola");//Si la variable es igual a 1
                        //se imprime hola
        break;

    default:
        printf("Chau");//Si la variable no vale 1
                        //se imprime chau
        break;
}
```

En donde si la variable es igual a 1 se ejecutaria el código del case 1, mientras que si no lo es siempre se ejecutaria el default, el default es opcional y no siempre es necesario, lo necesario es que al menos tenga un case, el break nos sirve para ponerle punto final a ese segmento de código y no siga con el código.

SWITCH

Siempre vamos a usar más de un case, preferiblemente más de dos ya que sino seria lo mismo usar un if/else, lo que hace la sentencia switch es solucionarnos el hecho de usar tantos if para una misma funcionalidad.

El **default** siempre nos va a servir para negar todos los **case** que tengamos, lo que significa que si lo que usamos no está en un **case**, pasa automáticamente al **default** si este existe.

La condición del switch puede no ser una variable, pero no es recomendable ya que no nos serviria de nada, seria lo equivalente a poner if(1) o if(0) se puede hacer pero no nos sirve de nada.

Otros ejemplos

```
switch(numeroRandom)
{
    case 1:
    case 3:
    case 5:
    case 7:
    case 9:
        printf("Es impar");
        break;

    case 2:
    case 4:
    case 6:
    case 8:
    case 10:
        printf("Es par");
        break;
}
```

Acá vemos otro ejemplo claro de otra forma de usar el switch, no hace falta poner el código y un **break** después de cada **case**, sólo cuando hacen una acción distinta.

BREAK

Lo que hace el **break** es simplemente terminar la sentencia actual ya sea (switch, if, else, while, for, entre otras).

Se usa cuando no necesitamos más que siga la sentencia, en el caso de los switch se es necesaria ya que sino el programa seguiria leyendo para abajo hasta encontrar un **break** o hasta que termine la sentencia.

El **case** de la última línea antes de cerrar la llave , o el **default** (si es que hay) no necesitarian un **break** al final, pero por buenas practicas lo ponemos en cada **case** y cada **default** aunque sea el último.



Estructuras Repetitivas



Variables de Control

Cuando usemos estructuras repetitivas (Especialmente en el for) usaremos una **variable de control** que tiene cómo objetivo contar la cantidad de **iteraciones** (Veces que se repite el algoritmo)

Está variable que por lo general la declaramos con la letra **i** me puede servir para muchas cosas, por ejemplo si yo quiero que haya sólo 5 iteraciones me es crucial el uso de está variable.

También para calcular el promedio de **n** cantidad de números posibles está variable nos será de vital importancia para dicho calculo.

Por general la variable de control la inicializamos en cero, aunque puede ser inicializada en cualquier valor y contar de manera descendente o de manera ascendente.

While

El while va a ser la primer estructura que vamos a ver hoy es el while.

Esta estructura repite el código que tiene dentro mientras su condición sea **verdadera (!=0)**. Sintacticamente funciona parecido al **if**, pero su funcionamiento es distinto.

```
int condicion;  
condicion = 3;  
  
while(condicion)//Mientras la condicion sea distinto a cero seguira ejecutando  
{  
    printf("%d\n",condicion);  
    condicion--;  
}
```

3
2
1

RESULTADO EN CONSOLA

While

Por lo general la condición va a ser una operación de relación ya sea (mayor, menor, igual, distinto, etc) y operaciones lógicas como el (AND, OR, NOT).

La diferencia que tenemos en el if es que while es más peligroso, hay que tener cuidado con este tipo de sentencias ya que si se le da mal uso en un lenguaje de bajo nivel como C podríamos dejar sin memoria a la maquina y por ende provocar el reinicio.

Posibles Errores

Como dijimos antes siempre tenemos que asegurarnos que nuestra condición pueda dar **falsa (0)** en algún momento, ya que sino haria que el programa ejecute dicha estructura repetitiva infinitas veces lo que consumiria toda la **ram** posible.

```
while(1)
{
    printf("HOLA MUNDO");
}
```

A diferencia de lo que pasaba en el **if** cuando pasabamos un **número distinto de cero** directamente, no pasaba nada, sino que iba a entrar al if y terminar.

En el **while**, si llegamos a hacer un **while(1)** (Por ejemplo) la condición nunca terminaria y la página se nos congelara.

Posibles Errores

Otro posible error sería conseguir que nuestra variable u operación lógica nunca llegue a ser falsa (valor cero).

Por ende ocurriría lo mismo que en el ejemplo anterior, la estructura repetitiva nunca terminaría y la pc se podría reiniciar por problemas de ram.

```
int i;  
i = 0;  
  
while(i > -1)  
{  
    printf("HOLA MUNDO");  
    i++  
}
```

En este ejemplo ocurriría como el anterior, la variable de control comienza en 0, y la condición evalúa que la variable debe ser mayor a -1, si cada vez que entre al while se hace un `i++`, la condición va a ser siempre 1.

Do While

—
Esta condición la usaremos cuando necesitamos que la sentencia se ejecute al menos una vez, por ejemplo para ingresar un sexo, una clave, o un número o todo lo que indique ingreso de datos.

Ejemplo

```
int numero;  
int suma;  
  
suma = 0;  
  
do  
{  
    printf("Ingrese un numero: ");  
    scanf("%d",&numero);  
    suma = suma + numero;  
}  
while(suma < 100);  
  
printf("La suma llego o supero el 100");
```

A diferencia del while, lo que hace está estructura repetitiva es primero ejecutarse al menos una vez, en el **do** nos pedir un número y hara la suma, luego ejecutara un **while** en el que si nos da **1**, se volvera a ejecutar el do.

Una vez que el while nos de **0**, el programa seguira, la desventaja de este método es que el mensaje no cambiara, y en caso de error al ingresar por ejemplo una contraseña no es tan orientativo al usuario.

Al final del while se pone ; ya que está terminando la sentencia.



Sentencia for



For

A diferencia del while, nosotros vamos a usar el for en las ocasiones que sepamos cuantas iteraciones vamos a realizar, para el for, necesitamos si o si una variable de control, que se va a incrementar o decrementar automáticamente una vez que termine su iteración. Por lo general a esta variable la denominaremos i, puede comenzar por cualquier valor, pero por lo general como en cualquier contador, la inicializamos en cero.

Su sintaxis es `for(inicialización,condición,incremento)`


Ejemplo

```
int i;


for(i=0;i<10;i++)
{
    printf("Ingrese un numero: ");
    scanf("%d",&numero);
    suma = suma + numero;
}

printf("La suma de los 10 numeros es %d ", suma)
```

En donde *i* es la variable de control, al declarar el `for` abrimos parentesis y primero seteamos el valor de *i*, luego ponemos la condición por ejemplo `i < 10` (cuando *i* sea 10 el `for` termina), por último ponemos una acción de decremento o incremento, como en cualquier estructura repetitiva, tenemos que asegurarnos de que siempre termine y evitar los bucles infinitos.



Acumuladores y Contadores



Contador y Acumulador

-El contador me permite contar cada iteración que yo haga en mi estructura repetitiva, ya sea while, do while o for, dicho contador siempre debe inicializar en cero (si cuenta de forma creciente) ya que por lo general se cuenta empezando del cero.

El acumulador me permite almacenar un valor total (que podría ser sueldoTotal, cantidadTotal, sumaTotal, etc), siempre se inicializa en cero y es crucial su uso en las estructuras repetitivas.

Ejemplo en Código

```
int contador;  
int acumulador;  
int numero;  
  
contador = 0;  
acumulador = 0;  
  
while(acumulador < 1000)  
{  
    printf("Ingrese un numero");  
    scanf("%d",&numero);  
    acumulador = acumulador + numero;  
    contador++;  
}  
  
printf("La suma de los %d numeros es %d y es mayor a 1000", contador , acumulador);
```



Máximos y Mínimos



Máximos y Mínimos

Para saber el máximo o el mínimo de un número ingresado, primero debemos declarar sus respectivas variables.

Luego **NO DEBEMOS INICIALIZARLAS** en **NINGÚN VALOR** ya que inicializarlas en un valor (Por ejemplo el cero) lograríamos que si ingreso sólo números negativos, su máximo sería cero siempre.

En el ingreso usabamos las banderas, para asignar el valor del primer número a la variable máximo y mínimo

Máximos y Mínimos

Ahora, vamos a usar con lo que sabemos y si hay presente un contador, el valor de dicho contador como si fuera una bandera.

-Para que yo necesite calcular el valor mínimo, debo asignar al primer número que se pide como el valor mínimo (O sea el primer número que entra, será el mínimo por defecto en la primer iteración). Para calcular el mínimo en cada iteración debo preguntar si el número mínimo que tengo en mi variable es mayor al número ingresado. Si da verdadero, el número mínimo será el número que acabo de ingresar.

Máximos y Mínimos

-Para que yo necesite calcular el valor máximo, debo asignar al primer número que se pide como el valor máximo(O sea el primer número que entra, será el máximo por defecto en la primer iteración). Para calcular el máximo en cada iteración debo preguntar si el número máximo que tengo en mi variable es menor al número ingresado. Si da verdadero, el número máximo será el número que acabo de ingresar.

Ejemplo en código

```
int numero;
int i;
int minimo;
int maximo;

for(i=0;i<10;i++)
{
    printf("Ingrese numero: ");
    scanf("%d",&numero);

    if(numero > maximo || i == 0)
    {
        maximo = numero;
    }

    if(numero < minimo || i == 0)
    {
        minimo = numero;
    }
}

printf("El numero maximo es %d y el numero minimo es %d",maximo,minimo);
```