Primeros algoritmos.

Laboratorio de Computación I
Tecnicatura Superior en Programación.
UTN-FRRQ

Tabla de Contenido

Primeros algoritmos

1.1 Bucles y tomas de decisiones

En lenguaje C existen los siguientes lazos iterativos:

- **for**
- **while**
- **do while**

y los siguientes bloques de control de flujo:

- **if else**
- **switch**

1.1.1 Bucle for

La sintaxis del bucle for es:

```
for (inicialización, condición, incremento) {
      // sentencias
}
```

En primer lugar, conviene destacar el hecho de la gran flexibilidad del bucle *for* de C. En C, el bucle *for* puede no contener inicialización, condición o incremento, o incluso pueden no existir dos e incluso las tres expresiones del bucle. El bucle for se ejecuta siempre que la condición sea verdadera, es por ello que puede llegar a no ejecutarse. Veamos algunos ejemplos de bucles *for*:

Ejemplo 1

```
int i, suma=0;
for (i=1; i<=100; i++) {
      suma=suma+i;
}
```

Ejemplo 2

```
char d;
for (; ;)
{
      d=getc(stdin);
      printf ("%c", d);
      if (d=='s')
            break;
}
```

Como se observa en este último ejemplo, el bucle *for* no posee ninguna expresión. Para salir de él se usa la sentencia *break*.

1.1.2 Bucle while

La sintaxis del bucle *while* es:

```
while (condición) {
      // sentencias
}
```

Donde la sentencia puede no existir (sentencia vacía), pero siempre debe existir la condición. El bucle *while* se ejecuta mientras la condición sea verdad. Veamos algunos ejemplos de bucles *while*:

Ejemplo 1

```
int i=1, suma=0;
while (i<=100)
{
      suma=suma+i;
      i++;
}
```

Ejemplo 2

```
while (getc(stdin)! ='\x1B'); /* Bucle que espera hasta que se
                              */ /* pulse la tecla Esc */
```

1.1.3 Bucle do while

Al contrario que los bucles *for* y *while* que comprueban la condición al comienzo de la misma, el bucle *do/while* comprueba la condición luego de ejecutar el bloque, lo cual provoca que el bucle se ejecute como mínimo una vez. La sintaxis del bucle *do/while* es:

```
do
{
   // sentencias
} while (condición);
```

El bucle *do/while* se ejecuta mientras la condición sea verdad. Veamos algunos ejemplos de bucle *do/while*:

Ejemplo 1

```
int num;
do
{
      scanf ("%d", &num);
} while (num>100);
```

Ejemplo 2

```
int i, j;
do
{
      scanf ("%d", &i);
      scanf ("%d", &j);
} while (i<j);
```

1.1.4 Bloque de control if

El lenguaje C posee un concepto muy amplio de lo que es verdadero. Para C, cualquier valor que sea distinto de cero es verdadero, siendo por tanto falso solo si el valor cero. Es por ello que una expresión del tipo *if(x)* será verdad siempre que el valor de la variable x sea distinto de cero, sea cual sea el tipo de la variable x.

El concepto de sentencia en C es igual que el de otros muchos lenguajes. Por sentencia se entiende en C cualquier instrucción simple o bien, cualquier conjunto de instrucciones simples que se encuentren encerradas entre los caracteres { y }, que marcan respectivamente el comienzo y el final de una sentencia.

La forma general de la sentencia *if* es:

```
if (condición)
{
	sentencia;
}
else
{
	sentencia;
}
```

Siendo el *else* opcional. Si la condición es verdadera se ejecuta la sentencia asociada al *if*, en caso de que sea falsa la condición se ejecuta la sentencia asociada al *else* (si existe el *else*). Veamos un ejemplo de sentencias *if*:

```
int a, b;
if (a>b)
{
	b--;
	a=a+5;
}
else
{
	a++;
	b=b-5;
}
```

Las sentencias de control *if* pueden ir anidadas. Un *if* anidado es una sentencia *if* que es el objeto de otro *if* o *else*.

1.1.5 Bloque de control switch

La forma general de la sentencia *switch* es:

```
switch(variable)
{
	case const1:
		// sentencias
		break;
	case const2:
		// sentencias
		break;
	// ...
	default:
		// sentencias
}
```

Donde variable debe ser de tipo *char* o *int*, y donde const1, const2, ..., indican constantes de C del tipo de datos de la variable. Dichas constantes no pueden repetirse dentro del *switch*. El *default* es opcional y puede no aparecer, así como los *break* de los case. La sentencia *switch* se ejecuta comparando el valor de la variable con el valor de cada una de las constantes, realizando la comparación desde arriba hacia abajo. En caso de que se encuentre una constante cuyo valor coincida con el valor de la variable, se empieza a ejecutar las sentencias hasta encontrar una sentencia *break*. En caso de que no se encuentre ningún valor que coincida, se ejecuta el *default* (si existe).

Ejemplo:

```
int valor;
switch(valor)
{
	case 0:
		cont++;
		break;
	case 5:
		cont--;
		break;
	default:
		cont=-10; /* Se ejecuta si valor no es 0 o 5 */
}
```

1.2 Acumuladores y contadores

Llamamos acumuladores o contadores a variables que llevarán una cuenta (no necesariamente incremental, sino también decreciente). El contador más típico es el utilizado en el bucle *for*:

```
int i;
for (i=0; i<10; i++)
{
	// ...
}
```

La variable "i" es un contador incremental en donde en cada iteración del bucle se incrementa en una unidad.
Supongamos que debemos realizar un programa que pida 10 números al usuario y muestre la suma de los mismos. Para ello necesitamos definir un contador (para contar 10 veces) y un acumulador (para ir sumando los números ingresados)

Definimos ambas variables:

```
int i;
int suma=0;
```

Luego escribimos el bucle utilizando el contador para iterar 10 veces:

```
for (i=0; i<10; i++)
{
	// ...
}
```

Dentro del bucle, deberemos pedir al usuario que ingrese un número y almacenarlo en una variable auxiliar, que luego sumaremos con el acumulador "suma". Esto se repetirá 10 veces.

El programa final quedaría:

```
int i;
int suma=0;

for (i=0; i<10; i++)
{
	int aux;
	scanf ("%d", &aux);
	suma=suma+aux;
}
printf ("la suma es: %d", suma);
```

Luego de guardar en la variable "aux" el número ingresado, mediante "scanf", guardamos en "suma" el valor que tenía previamente más el valor de "aux":

```
suma=suma+aux;
```

Esta sentencia también puede escribirse de una forma más compacta de la siguiente manera:

```
suma+=aux;
```

1.3 Máximos y mínimos

Si queremos calcular cuál fue el máximo número ingresado y el mínimo, en el programa del ejemplo anterior, deberemos declarar dos variables, una donde se guardará el máximo calculado y otra el mínimo:

```
int max, min;
```

Para poder realizar el algoritmo, deberemos preguntar, cada vez que el usuario ingresa un número (es decir, después de la línea del *scanf*) si el número ingresado es mayor que el valor que tenemos como mayor actualmente (es decir, el valor almacenado en la variable "max"):

```
if(aux>max)
```

En el caso que el número ingresado (aux) sea mayor que el que tenemos hasta el momento como máximo (max), entonces el nuevo máximo es el valor en "aux":

```
if(aux>max)
{
	max = aux;
}
```

Aplicamos la misma lógica para detectar el mínimo:

```
if(aux<min)
{
	min = aux;
}
```

Colocamos los dos bloques "if" dentro del bucle, y al finalizar el mismo, quedarán en las variables "max" y "min" los valores máximos y mínimos ingresados respectivamente.

Para que este programa funcione correctamente, deberemos dar valores iniciales a las variables "max" y "min" ya que no tienen definido ninguno.
Para asegurarnos de que el primer número ingresado por el usuario sea tanto el máximo como el mínimo (hecho que es cierto si solo se ingresa un número), deberemos darle a "max" y "min" valores tales que no importa qué número ingrese el usuario, se cumplan las condiciones para entrar a los bloques *if*.

Si queremos que se cumpla "aux>max" para cualquier valor de "aux":

```
if(aux>max)
	max = aux;
```

Deberemos inicializar "max" con **el valor más chico posible permitido** que pueda almacenarse en dicha variable (recordemos que cada tipo de dato tiene cierta capacidad de almacenamiento)

Si queremos que se cumpla "aux<min" para cualquier valor de "aux":

```
if(aux<min)
	min = aux;
```

Deberemos inicializar "min" con **el valor más grande posible permitido** que pueda almacenarse en dicha variable.

Existen constantes que poseen los números más grandes y pequeños que pueden almacenarse en cada tipo de dato, para poder usar estas constantes deberemos incluir la biblioteca "limits.h", por lo que nuestro programa quedaría terminado de la siguiente forma:

```
#include <stdio.h>
#include <limits.h>

void main(void)
{
	int i;
	int suma = 0;
	int max = INT_MIN; // constante definida en limits.h
	int min = INT_MAX; // constante definida en limits.h

	for (i=0; i<10; i++)
	{
		int aux;
		scanf ("%d", &aux);
		suma=suma+aux;

		if(aux>max)
		{
			max = aux;
		}

		if(aux<min)
		{
			min = aux;
		}
	}
	printf ("la suma es: %d\n", suma);
	printf ("El máximo es: %d\n", max);
	printf ("El mínimo es: %d\n", min);
}
```