Guía Práctica: Itinerario Formativo.
Desarrollo de Programas y Manipulación de Datos.

C++

Prof: Mileti

La programación en C++ se basa en funciones. Un programa en C++ puede tener una o más funciones, una de las cuales debe ser main(), que es la función principal en donde se inicia el programa. La incorporación de otras funciones propias del programador es opcional. A continuación un ejemplo con dos funciones: main() y suma().

```
#include<stdio.h>
//prototipo de la función suma
int suma(int parametro1, int parametro2);

int main(){
/* variables locales de la función main,
no puede ser utilizadas desde otras funciones. */
  int valor1, valor2, valor3;
  valor1=10;
  valor2=5;
  valor3=suma(valor1,valor2);
  printf("%d",valor3);
  return 0;
}

int suma(int parametro1, int parametro2) {
//variable local de la función suma
  int resultado;
  resultado=parametro1+parametro2;
  return (resultado);
}
```

Como se observa, la codificación se encuentra documentada mediante el uso de comentarios. Es posible introducir un comentario de dos maneras diferentes:

```
system("cls"); // limpia la pantalla
```

En la línea de código anterior el compilador de C++ a partir de // “cierra los ojos” y no tiene en cuenta los caracteres hasta que termine la línea.

```
  system("pause"); /* hace una pausa en la
  ejecución del programa */
```

En la línea de código anterior el compilador de C++ “cierra los ojos” a partir de /* y no tiene en cuenta los caracteres hasta que aparezca el cierre de comentario */

Instrucción #include

Esta instrucción le indica al compilador que recupere el código del archivo de cabecera indicado (archivos .h). En los archivos de cabecera se encuentran las funciones que podemos llegar a incorporar en nuestro programa en C++. Por ejemplo las funciones printf y scanf se encuentran en stdio.h, por lo tanto para utilizarlas es necesario la siguiente instrucción:

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

Declaración y tipos de variables:

Los nombres de las variables pueden contener una secuencia de letras, dígitos y caracter de subrayado. Todo nombre de variable debe comenzar con una letra o un caracter de subrayado. No está permitido utilizar como nombre de variable una palabra reservada de C++ (como while, por ejemplo) o una palabra que comience con un número.

Los tipos de datos indican que contendrá una variable:
`int`: Número entero con signo en un S.O. de 16 bits su rango va de -32763 a 32762.
`float`: Números decimales con punto flotante.
`char`: Caracter del código ASCII, representado por un número entero.
`bool`: Valor lógico que puede ser true (verdadero) o false (falso) .

Ejemplos de declaración de variables:

```
float promedio;
int cantidad;
char letra;
bool salir;
```

Variables globales y locales:

Las variables globales pueden ser utilizadas por cualquier función.
Las variables locales pueden ser utilizadas únicamente por la función en donde se encuentra declarada.

La diferencia está dada en el lugar en que se declara.

```
#include<iostream>
//variable global, disponible en todas las funciones.
int cantidad;

int main(){
//variable local a la función main, no visible fuera.
 int contador;
 return 0;
}
```

Operadores aritméticos:

Los siguientes son los operadores aritméticos que nos permitirán realizar cálculos matemáticos:

Operador	Cálculo matemático
-	Resta
+	Suma
*	Multiplicación
/	División
%	Resto
--	Decremento
++	Incremento

Operadores relacionales y lógicos

Los operadores relacionales nos permiten comparar un valor con otro, mientras que los operadores lógicos nos permiten conectar varias proposiciones lógicas utilizando las reglas de la lógica formal.

Operador relacional	Acción
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
==	Igual que
!=	Distinto que

Operador lógico	Acción
&&	AND (Y lógico)
||	OR (Disyunción lógica)
!	Not (Negación lógica)

Estructuras de selección (if - switch)

Las estructuras de selección, o condicionales, permiten bifurcar la ejecución de sentencias. Es decir, permiten ejecutar determinadas instrucciones de acuerdo a ciertas condiciones dadas en un determinado momento.

Mediante la instrucción **if** se puede conseguir que la ejecución de un bloque de instrucciones dependa del valor de una condición lógica.

```
if (temperatura<10)
   cout<<”Use un buen abrigo”;
```

En el código anterior sólo es mostrará el mensaje si el valor de la variable temperatura es menor que 10 (proposición lógica verdadero). En caso de querer ejecutar más de una sentencia se las encierra entre llaves para indicar que es un bloque de sentencias:

```
if (temperatura<10){
     cout<<”Mucho frío…”<<endl;
     Sleep(1000);
     cout<<”Use un buen abrigo.”;
}
```

También es posible ejecutar una serie de instrucciones en caso que no se cumpla con la condición propuesta:

```
if (promedio>=7)
   cout<<”Felicitaciones aprobó la
materia!!!”;
else
   cout<<”Lamentablemente ha desaprobado…”;
```

En caso de tener que comparar un valor con una gran cantidad de alternativas podemos hacer uso de la instrucción **switch**:

```
switch((int)promedio){
      case 1:
      case 2:
      case 3:
            cout<<"Rinde en marzo";
            break;
      case 4:
      case 5:
      case 6:
            cout<<"Rinde en diciembre";
            break;
      case 7 ... 10:
            cout<<"Aprobado!!!";
            break;
      default:
            cout<<"Nota no válida.";
}
```

Estructuras de repetición

Los ciclos de repetición, o bucles, nos permiten repetir un conjunto de instrucciones hasta alcanzar una condición. Esta condición puede estar predefinida, como el caso del for, o indefinida como en los ciclos while y do... while.

En el ciclo **while** el bloque de sentencias que contenga, se ejecuta mientras la condición propuesta sea verdadera.

```
#include<iostream>
int main(){
  int clave;
  using namespace std;
  cout<<"Ingrese número de clave: ";
  cin>>clave;
  while(clave!=123){
     cout<<"Clave incorrecta"<<endl;
     cout<<"Ingrese número de clave: ";
     cin>>clave;
  }
  cout<<"Bienvenido"<<endl;
  system("pause");
  return 0;
}
```

En el ejemplo anterior se solicita el ingreso de un número que será almacenado en la variable clave. Mientras el valor ingresado sea distinto a 123 solicitará una clave. Cuando deje de cumplirse la condición (123! =123) se sale del bucle y continúa la ejecución desde la primer instrucción a continuación del bloque de sentencias. En el bucle while es posible que nunca se ejecute el bloque de sentencias (si se evalúa la condición inicialmente como falsa).

Una variante del while es el **do...while**. A continuación un ejemplo:

```
#include<iostream>
  int main(){
  int clave;
  using namespace std;
  do{
    cout<<"Ingrese número de clave: ";
    cin>>clave;
  } while(clave!=123);
  cout<<"Bienvenido. Presione Enter por favor";
  cin.get();
  cin.ignore();
  return 0;
}
```

El do...while se interpreta como: “hacer mientras”. Es decir primero ejecuta el bloque de sentencias y luego evalúa la condición. En el código anterior, primero solicita el ingreso de un número y luego evalúa la condición. En caso de ser la condición verdadera, vuelve a ejecutar el bloque de sentencias. Este proceso se repite hasta que la condición deje de ser verdadera. Aquí nos aseguramos que el bloque de sentencias se ejecute al menos una vez.

El bucle for, al igual que while, repite un bloque de sentencias cierto número de veces, hasta que deje de cumplirse una condición. Su sintaxis es la siguiente:

```
for (inicio;condición;fin){
//bloque de instrucciones
}
```

La instrucción “inicio” solo se ejecuta una vez, al comienzo del ciclo. El bloque de sentencias se ejecutará repetitivamente mientras “condición” sea verdadera. En “fin” se suele utilizar una sentencia que genere un cambio en la condición. A continuación un ejemplo:

```
#include<iostream>
#include<windows.h>
int main(){
  int segundos;
  using namespace std;
  cout<<"En un minuto finaliza el programa!"<<endl;
  for (i=0;i<60;i++){
     Sleep(1000);//produce un retardo de 1 segundo
     cout<<i<<endl;
  }
  return 0;
}
```

La instrucción i=0 inicializa la variable i con valor cero. La condición esta dada por i<60, es decir mientras i sea menor a 60 se ejecutará el bloque de sentencias. Por cada iteración se ejecuta i++, que equivale a i=i+1, es decir incrementa la variable i en una unidad, por lo cual en algún momento i alcanzará el valor de 60 y culminará el ciclo for.

Funciones para tener en cuenta:

En C++ existe una gran cantidad de librerias que contienen funciones interesantes. A continuación las más utilizadas.

<stdio.h>
`printf()`: Escribe un texto en la pantalla.
`scanf()`: Lee un valor desde el teclado y lo asigna a una variable.
`getchar()`: Lee un carácter desde el teclado.

<iostream>
`cout<<`: Escribe un texto en la pantalla.
`cin>>`: Lee un valor desde el teclado y lo asigna a una variable.

<string.h>
`strcat()`: Concatena cadena de caracteres.
`strcmp()`: Compara cadenas de caracteres.
`Strcpy()`: Copia cadenas de caracteres.
`strlen()`: Devuelve el largo de una cadena de caracteres.

<windows.h>
`Sleep()`: Genera una pausa en la ejecución de un programa.
`Beep()`: Emite un determinado sonido.

<stdlib.h>
`rand()`: Genera un número pseudo-aleatorio.
`srand()`: Establece la semilla para el generador de números pseudo-aleatorios.
`exit()`: Finaliza la ejecución del programa.
`itoa()`: Convierte un entero a ASCII.
`atoi()`: Convierte un carácter ASCII en entero.
`randomize()`: Genera un secuencia de números al azar.
`system()`: Ejecuta un comando externo.

<math.h>
`pow()`: Realiza cálculo de potencia.

<graphics.h>
`initwindow()`: Inicializa el modo gráfico.
`settextstyle()`: Establece la fuente y el tamaño del texto.
`outtextxy()`: Muestra un texto en las coordenadas x,y.
`line()`: Traza una línea recta.
`recatngle()`: Dibuja un rectángulo.
`circle()`: Dibuja un circulo.
`setcolor()`: Establece el color de dibujo.
`setbkcolor()`:Establece el color de fondo.

Caracteres especiales

¿Y la ñ y los acentos? No salen bien...

```
printf("Cumplea%cos",164);// %c-->164(Ascii ñ)
cout<<"Cumplea/xA4os";  //  /x --> A4 (hexa ñ)
```