Guía simple de C++

Introducción a la Computación

NOTA:

Este material presentado puede ser descargado y distribuido libremente por cualquiera, es elaborado con el único propósito de fomentar la distribución de conocimiento y apoyar al estudiante.

Este documento es principalmente elaborado para los estudiantes de la asignatura de introducción a la computación del Decanato de Ciencias y Tecnología de la Ucla y distribuido originalmente en http://dcytintro.ddns.net/ , por lo que el material mostrado es orientado a lectores sin manejo del lenguaje y sin necesidad de aplicar conocimientos profundos que siguen una estructura de aprendizaje concreta y podría provocar que cualquier otro lector pueda verse confundido en ciertas secciones del documento.

Existen muchos lenguajes de programación para diversas funciones o propósitos, esto implica que cada uno de los lenguajes existentes tiene su propia sintaxis, ya sea por su funcionalidad, por su sistema de tipos(como el lenguaje declara los datos) o por su necesidad. Lo cierto es que lo que importa es aprender a manejar la lógica de la programación, pero al iniciarse en un nuevo lenguaje puede ser complicado adaptarse a su sintaxis.

En esta oportunidad se presenta una guía sencilla para iniciados en C++ orientado al aprendizaje de su sintaxis básica. Empecemos entonces con el cuerpo de un programa muy sencillo escrito en C++.

Si ya has programado en C antes encontrarás que no hay grandes cambios en la sintaxis básica de un programa, en la atención al detalle necesaria a la hora de declarar las variables, las librerías, las funciones y más. Esto es debido a que C++ es un lenguaje creado en base a C y adaptado para el paradigma de programación orientada a objetos.

Ojo, esta guía no es un manual explicativo de toda la lógica de la programación ni de un manejo profundo de C++, es una guía para ayudar a los estudiantes de introducción a la computación adaptarse al uso del lenguaje.

Observa el siguiente ejemplo de un programa sencillo y funcional:

Elementos principales:

```
#include <iostream>  ———————————  Librería estándar de C++

using namespace std;  ———  Espacio de nombres: Palabras clave de C++

int main( ){  ———————————  Función Principal

    //Comentario de una línea

    int num=0;  ———————————  Declaración de variable

    cout<<"Escribe un número del 1 al 10: "  —  Mostrar en pantalla

    cin>>num;  ———————————  Leyendo datos de ingresados

    cout<<"\nEscribiste: <<num;

    return 0;  ———————————  Indica fin exitoso del programa.
}
```

Estos elementos y más serán explicados a lo largo de esta guía para los que empiezan con este lenguaje. Empezando con la guía veamos los tipos de datos y su declaración.

1-Tipos de Datos Básicos:

Hay varios tipos de datos, pero estos son los que más usarás al inicio.

Declaración de dato	**Tipo de dato**	**Ejemplo**
char	Carácter (un solo dato)	'a','$','8'
int	Entero (número)	12 , 58, 400, -10
float	Flotante (decimal)	-14.5, 0.56, 150.87
string	Cadena de caracteres	"Hola Mundo"
bool	Booleano(lógico)	True / False

1.1

2-Variables:

Declaración:

```
tipo_de_dato  nombre_Variable ;

tipo_de_dato  nombre_Variable = valor_Variable ;
```

2.1

Ejemplo:

Variable sin valor inicial	**Variable con valor inicial**
char *variableChar ;*	***char*** *variableChar = 'a' ;*
int *variableEntera ;*	***int*** *variableEntera = 12 ;*
float *variableFlotante ;*	***float*** *variableFlotante = 20.56 ;*
string *variableString ;*	***string*** *variableString = "Hola mundo" ;*
bool *variableBooleana ;*	***bool*** *variableBooleana = true;*

2.2

3-Entrada y salida de datos:

Imprimir datos en pantalla (*cout*):

```
cout<<"Mensaje en pantalla";

cout<<"Mensaje en pantalla|n";     ( \n indica un salto de línea ).

cout<<"Mensaje en pantalla"<<endl ;     ( endl indica un salto de línea ).

cout<<nombre_variable;     ( muestra el valor de una variable ).
```

3.1

Leer entradas de usuario (*cin*):

```
cin>>nombre_variable ;     ( agrega la entrada del usuario a la variable ).
```

3.2

Hagamos que el usuario nos indique un número:

```
int numVariable ;

cout<<"Ingresa un número, por favor: " ;
cin>>numVariable ;

cout<<"|nEl número ingresado es: "<<numVariable ;
```

3.3

Más adelante en la guía verás el resto de elementos que conforman un programa sencillo de C++ y podrás probar ese código. Mientras tanto sigamos con la guía.

4-Operadores lógicos/aritméticos:

Símbolo	Operador
+, -, *, /, %	suma, resta, multiplicación, división, residuo.
=	igual a (asignación)
==	igual que (comparación)
!=	distinto que (comparación)
>, <	mayor que, menor que
>=, <=	mayor igual que, menor igual que
&&	“y”, and lógico (^)
||	“o”, or lógico (v)

4.1

5-Condicionales:

Condicionales, estructuras que nos permiten declarar una serie de órdenes que solo serán ejecutadas en caso de que un caso o condición (valga la redundancia) se cumpla. Para esto existen 2 estructuras principales:

If:

-Sintaxis:

```
if( condición ){

        //código si se cumple la condición
}

else {

        //código si ninguna condición anterior se cumple
}
```

5.1

Else es usado en caso de que la condición anterior (if) no se cumpla.

-Ejemplo:

```
int var_condicional=1 ;

if ( var_condicional == 1 ){

        cout<<"La condición se cumple";
}

else{

        cout<<" La condición no se cumple" ;
}
```

5.2

El resultado de esta pieza de código es: *La condición se cumple.*

Switch:

Switch funciona en base a una serie de casos posibles.

-Sintaxis:

```
switch( variable_caso ){

        case 1:

                cout<<" La variable es igual a 1" ;
                break;

        case 2:

                cout<<"La variable es igual a 2" ;
                break;

        default:

                cout<<"Ningún caso se cumple";
                break;
{
```

5.3

-Ejemplo:

```
int variable_caso= 3 ;

switch(  variable_caso ){

        case 1:
                cout<<"La variable es igual a 1" ;
                break;

        case 2:
                cout<<"La variable es igual  a 2";
                break;

        case 3:
                cout<<"La variable es igual a 3";
                break;

        default:

                break;
}
```

5.4

6-Contadores y Acumuladores:

Los contadores y acumuladores juegan un papel importante junto al uso de las estructuras iterativas (ciclos while, for y do_while), por lo que es importante entender su funcionamiento.

Contadores:

Un contador es una variable cuyo propósito es... pues contar. Ya sea de uno en uno o como lo desees y su funcionamiento es:

```
int contador=0 ;

contador=contador+1   //El contador sumará 1 a su valor cada ciclo.

contador++ ;  //Esta forma de declarar el contador es igual a la anterior.
```

6.1

Acumuladores:

El acumulador es una variable que suma el valor de otra variable hasta que el ciclo se termine, tal que:

```
acumulador=acumulador+variable_x;
```

6.2

-Supongamos que:

```
int sueldoTotal= 0 ;

int sueldo=0 ;

cout<<"Ingrese sueldo: " ;  cin>>sueldo;

sueldoTotal=sueldoTotal+sueldo;
```

6.3

De esta forma la variable de *sueldoTotal* irá sumando las cantidades de ingresadas del sueldo en cada ciclo... Ahora sí veamos ciclos.

7-Estructuras Iterativas:

Los ciclos son estructuras que permiten que una sección de código se ejecute en bucle mientras una condición se cumpla. Estos son principalmente tres:

While:

Un ciclo que se repite de forma indeterminada hasta que su condición no se cumpla.

```
while( condición ){

        /*
                pieza de código que necesites que se ejecute
        /*
}
```

7.1

Veamos un ejemplo, usemos el acumulador anterior(figura 6.3):

```
int respuesta = 1;
float sueldo,sueldoTotal;

while( respuesta == 1 ){

        cout<<"Ingrese sueldo: " ;  cin>>sueldo;

        sueldoTotal=sueldoTotal+sueldo;

        cout<<" \n Procesar otro sueldo ?  ( 1 - si  , 2 – no): ";
        cin>>respuesta ;
}
```

7.2

Se repetirá siempre que *respuesta* sea igual a 1. Ahora imaginemos que queremos que se repita en una cantidad de veces determinada, usamos un contador:

```
int i=0;  //Nuestro contador

while( i<5 ){

        //Donde 5 es la cantidad de veces que se repetirá

        cout<<"Este es el ciclo "<<i << endl;

        i++;  // hacemos que nuestra variable cuente al final de ciclo.
}
```

7.3

Esto da como resultado:

```
Este es el ciclo 1
Este es el ciclo 2
Este es el ciclo 3
Este es el ciclo 4
Este es el ciclo 5
```

7.4

Observe que en este ejemplo el número en la salida es dado por nuestro contador.

Para situaciones donde deseamos que un ciclo se repita una cantidad de veces fija (como el ejemplo anterior) existe el ciclo For, cuya sintaxis simplifica la declaración de un ciclo de repetición fija.

For:

El for cuenta con tres espacios, el valor de nuestro contador, la cantidad de veces que se repetirá el ciclo y como contará la variable:

```
for ( valor  de contador ; veces que se repite ;  forma de contar  ) {

        // código que necesitemos repetir

}
```

7.5

Hagamos el ejemplo anterior con while(figura 7.3), y comprenderás el funcionamiento y la sintaxis del for:

```
int i;  //contador de ciclo

for( i = 0 ;  i<5 ; i++ ){

        cout<<"Este es el ciclo "<< i << endl;

}
```

7.6

Este ciclo for tiene la misma funcionalidad que el último ejemplo del ciclo while.

Do while:

El ciclo do while tiene la misma funcionalidad que el while, con la diferencia de que la primera vez de su ejecución el programa entra en el ciclo sin importar la condición establecida, si una vez terminado el primer ciclo la condición se cumple entonces el ciclo se repetirá, sino seguirá ejecutando el resto del código fuera del ciclo.

While: Pregunta y luego ejecutada .
Do while: Ejecuta y luego pregunta.

Veamos su sintaxis:

```
Do {

        // código que necesitemos repetir

} while ( condición );
```

7.7

Probemos el primer ejemplo while (figura 7.2):

```
int respuesta= 0;
float sueldo,sueldoTotal;

Do {

        cout<<"Ingrese sueldo: " ;  cin>>sueldo;

        sueldoTotal=sueldoTotal+sueldo;

        cout<<" |n Procesar otro sueldo ?  ( 1 - si  , 2 – no): ";
        cin>>respuesta ;

} while (  respuesta == 1 );
```

7.8

En este caso observe que, aun siendo la variable respuesta igual a 0, el código dentro del ciclo será ejecutado de igual forma la primera vez.

8-Clases:

Esta es una guía de apoyo al estudiante, no un manual, por lo que no pararemos a explicarte toda la teoría que hay detras del paradigma de programación orientada a objetos (POO). Empecemos a explicar las clases en C++.

Normalmente al trabajar con clases se usan dos archivos, un header donde se declara los elementos de la clase (miClase.h) y un cuerpo donde se especifica el funcionamiento de dichos elementos (miClase.cpp), Pasemos a explicar la sintaxis de ambos.

Sintaxis:

miClase.h:

```
#include <iostream>

class miClase {

        private:

                //Aquí declaras tus atributos.

                int atributo_X;

        public:
                //Aquí declaras tus métodos

                miClase();  //constructor de clase.

                void set_AtributoX( int atriX ) ;
                int get_AtributoX( );

                float metodo_ Cualquiera() ;
```

Simplemente hacemos las declaraciones de los atributos de nuestra clase y los métodos set/get de dichos atributos, junto al resto de métodos que necesitemos que nuestra clase tenga.

Para este ejemplo se usa un atributo cualquiera para usar de representación de los atributos de una clase y sus métodos respectivos.

```
};
```

8.1

miClase.cpp:

```
#include "miClase.h"

miClase::miClase(){

        //Constructor de clase
        atributo_X=0;
}

void miClase::set_AtributoX( int atriX ){

        atributo_X = atriX;

}

int miClase::get_AtributoX( ){


        return atributo_X ;
}

float miClase::metodo_Cualquiera( ){

        //código del método

}
```

8.2

Incluimos el archivo **miClase.h** en nuestro archivo cpp y escribimos el código de los métodos declarados antes en el header.

En este caso se muestra el codigo simple que habría en un método set y get para un atributo de tipo entero *int .*

También se observa como inicializamos el atributo *atributo_X* en nuestro constructor con valor de 0.

Esa es la sintaxis básica de una clase simple con la que te encontrarás al inicio, sin embargo es puede ser confuso toda esa estructura sin un enfoque concreto, por lo que como se ha hecho hasta ahora con todos los elementos hasta ahora aquí mostrados ahora verás un ejemplo más realista pero manteniendo la simpleza para mayor facilidad de entendimiento.

En este ejemplo se mostrará la estructura anterior creando la clase ***estudiante***, que tendrá por atributos nombre, cédula y semestre cursado.

Ejemplo:

Estudiante.h :

```
#include <iostream>
#include <string>

class estudiante {

        private:

                string nombre, cedula;
                int semestre_cursado;

        public:

                estudiante( ) ;

                void  set_nombre( string nom) ;
                string get_nombre ( ) ;

                void set_cedula ( string ced ) ;
                string get_cedula ( ) ;

                void set_semestreCursado ( int sCursado );
                int get_semestreCursado ( ) ;
};
```

8.3

Este es un ejemplo más limpio de el código de una clase enfocado al analizar el mundo y los problemas con el paradigma de la programación orientada a objetos. Este ejemplo es una clase muy sencilla pero sigue los parámetros básicos de una clase y que fueron expresados en las figuras anteriores (8.1 , 8.2).

Ahora que ya declaramos la clase estudiante procedemos a escribir el código dentro de los métodos en nuestro archivo estudiante.cpp.

Estudiante.cpp

```
#include “Estudiate.h”

estudiate::estudiante( ){

        nombre = “ “ ;
        cedula = “ “ ;
        semestre_cursado=0;
}

void estudiante::set_nombre( string nom ){


        nombre = nom;
}

string estudiante::get_nombre( ){

        return nombre;
}

void estudiante::set_cedula( string ced ){

        cedula = ced;
}

string estudiante::get_cedula( ){

        return cedula ;
}

void estudiante::set_semestreCursado ( int sCursado ){

        semestre_cursado = sCursado ;
}

int estudiante::get_semestreCursado ( ) {

        return semestre_cursado ;
}
```

Por los momentos esta guía ha recorrido los primeros obstáculos con los que como estudiante te encontrarás en la asignatura de introducción a la computación, lo más importante es tomar en cuenta esta guía por lo que es, solo una guía sencilla para brindarte apoyo al inicio, hay mucho material en explicativo dado por los profesores y en el internet, considera siempre como mejor forma de aprender el aclarar tus dudas con tu profesor/a o alguien que ya haya lidiado con las mismas situaciones antes.

Si obtuviste esta guía por medio de terceros entonces puedes ir a su página de origen en la sección de C++, en la misma puedes entrar a una galería de imágenes que muestra ejemplos de los mismos elementos que fueron abarcados en esta guía y mucho más material que podría serte de ayuda y de interés.

Espero que esta guía haya podido ser de ayuda durante tu estudio, y si así fue espero que lo compartas a otras personas que consideres que lo necesiten. En caso contrario entonces pues simplemente ignóralo o has tus recomendaciones... cuando haya un sitio disponible donde puedas enviar tus recomendaciones.