

# SinglePath games design



**Versión 3.0**

Acerca del Autor

Capítulo 1 Introducción a C++

Capítulo 2 Las partes de un programa de C++

Capítulo 3 Variables y Constantes

Capítulo 4 Expresiones y Secuencias

Capítulo 5 Funciones

Capítulo 6 Clases

Capítulo 7 Secuencias de control

Capítulo 8 Punteros

Capítulo 9 Referencias

Capítulo 10 Herencia

Capítulo 11 Arreglos o Matrices

Capítulo 12 Polimorfismo

Capítulo 13 Clases especiales

Capítulo 14 Cadenas de caracteres

Capítulo 15 Gráficos BGI

Capítulo 16 Comentarios Finales

Capítulo 17 Bibliografía y Agradecimientos

Capítulo 18 El Fin

## Acerca del Autor



Mi Nombre es Alvaro Tejada, soy de Lima, Perú, tengo 26 años y junto con mi novia Milly, estudio Computación e Informática. Soy experto de la categoría de programación en Xpertia, bajo el pseudónimo de Blag. Y dueño de mi propia "compañía" (Ficticia por el momento) desarrolladora de software multimedia, SinglePath games design. Que espero en un tiempo no muy lejano, llegar a constituir como una de la mejores empresas de desarrollo multimedia. Mis inicios en la programación datan aproximadamente del año 1991, cuando vivía en el DF, en México. Allí lleve un curso de introducción al Pascal, aunque claro, en esos tiempos a pesar de que me llamaban la atención las computadoras, no tenía claro que quería hacer. Fue recién cuando vi por primera vez el juego Mortal Kombat de Midway que decidí que quería ser programador. En ese entonces tenía, si no me equivoco, una Macintosh Plus con Think Pascal. Para ser sinceros, no entendía nada y el único programa que logré hacer fue uno que venía como ejemplo en el manual de usuario y que nunca pude modificar. Luego el tiempo pasó y en el año 1997 ingresé a la Universidad de Lima en la cual llevé el curso de "Técnicas de Programación I", en el cual tuve mi primer contacto con C++, aunque aún no con los resultados que yo esperaba (Siempre la programación es difícil al principio). Fue a principios de 1998, en febrero más o menos, que mi papá me compró un libro de Visual Basic "Visual Basic 6 Bible" y uno de C++ "Teach Yourself C++ in 21 Days". Y así comencé mi carrera en el mundo de la programación. No voy a negar que al principio me costó bastante y que mis primeros programas eran bastantes simples, pero sentía la satisfacción de haberlos hecho yo mismo.

El tiempo pasaba y me retiré de la Universidad de Lima, para ingresar a Cibertec, en donde conocí por primera vez lo que era el lenguaje "Java". Ya una vez que mis conocimientos de programación habían aumentado, y mis programas y juegos mejorado, decidí explorar otros lenguajes de programación, como Visual C++, Euphoria, Visual FoxPro, QBasic, SQLServer, VBA, JavaScript, HTML, dBase III plus y Pascal. Actualmente, trabajo como Consultor SAP. Hasta ahora he desarrollado cerca de 32 programas y 10 juegos en distintos lenguajes, los cuales pueden descargarse gratuitamente de SinglePath games design.

La idea de escribir un libro de programación la tuve porque como programador autodidacta, he leído muchísimos libros y tutoriales, y pensé que tal vez podía dar una pequeña contribución, espero que les guste y escríbanme si tienen alguna duda, queja o comentario.

## Capítulo 1 - Introducción a C++

El lenguaje de programación C++ es uno de los lenguajes llamados de alto nivel, porque una sola de sus instrucciones equivale a millares de código de máquina.

C++ es un lenguaje de programación creado por Bjarne Stroustrup, hecho en base al famoso y comercial lenguaje C. Cuyas capacidades fueron extendidas para convertirlo en un lenguaje orientado a objetos. La cual es la metodología de programación más difundida y utilizada actualmente. Por último, solo queda mencionar que C++ es un lenguaje robusto y ha demostrado con el pasar del tiempo de que es un lenguaje que debe ser aprendido por cualquier persona que quiere programar y tiene cierta base, porque además de bueno, el lenguaje C++, es uno de los más difíciles de aprender.

Bueno, después de esta breve introducción, podemos pasar al capítulo 2 "Las partes de un programa en C++".

## Capítulo 2 - Las partes de un programa en C++

Para comenzar, tenemos el formato general de un programa hecho en C++:

```
#include <iostream.h>
#include <conio.h>

int main()
{
    clrscr();
    cout<<"Hola Mundo";
    getch();
}
```

Aquí esta su explicación punto por punto:

`#include <iostream.h>`    //Librería necesaria para cout.

`#include <conio.h>`    //Librería necesaria para getch() y clrscr();

`int main()`    //Rutina o función principal

//Determina donde empieza el código a ejecutar

`clrscr();`    //Borra el contenido de la pantalla

`cout<<" ";`    //Escribe en la pantalla

`getch();`    //Hace una copia de la pantalla

`}`    //Indica que el código terminó

## Una parte importante de todo código de programación son los comentarios:

Los comentarios, son partes del código que sirven como referencia para otros programadores que desean modificar el código fuente. Además, si un día formulas un código complejo, a los tres meses podrías no saber como funciona, pero si lo has comentado no tendrás problemas.

Los comentarios son de dos tipos:

// --> Comenta una línea

/\* \*/ --> Comenta un bloque de código, de varias líneas

Aquí hay un pequeño ejemplo:

```
#include <iostream.h> //Librería para cout y cin
#include <conio.h>      //Librería para clrscr() y getch()

int main()
{
    /*Este es un comentario
    continúa hasta que se cierre
    el slash*/
    clrscr(); //Borra la pantalla
    cout<<"Hola Mundo"; //Escribe un mensaje en la pantalla
    //Este es otro tipo de comentario, que solo comenta una línea
    getch(); //Copia la pantalla
}
```

## Capítulo 3 - Variables y Constantes

### Que es una Variable:

En C++, una variable es un lugar para guardar información, es un espacio en memoria que guarda datos por un periodo determinado de tiempo. Hay dos tipos, unsigned y signed. Todas las variables son signed por default (Signed incluye negativos).

### Tipos de Variables:

**Bool** --> Verdadero o Falso  
**unsigned short int** --> 0 a 65,535  
**short int** --> -32,768 a 32,767  
**unsigned long int** --> 0 a 4,294,967,295  
**long int** --> -2,147,483,648 a 2,147,483,647  
**int** --> -32,768 a 32,767  
**unsigned int** --> 0 a 65,535  
**char** --> 256 caracteres  
**float** --> 1.2e-38 a 3.4e38  
**double** --> 2.2e-308 a 1.8e308

### Definiendo una Variable:

Para declarar una variable, primero debe escogerse su tipo, ya sea int, long, o short int. Luego, hay que darle un nombre que, sea de preferencia relacionado con la función que va a realizar. Las variables deben ser llamadas tal como han sido declaradas. Edad no es lo mismo que EDAD o que EdaD. Por ejemplo:

```
#include<iostream.h> //Librería para cout y cin
#include<conio.h>      //Librería para clrscr() y getch()

int main()
{
    unsigned short clases; //Declaración de variables
    long total;
    int alumnos;
    total = clases * alumnos; //Asignamos a la variable Total, la multiplicación de
```

```
//clases por alumnos
cout<<"El total es: "<<total; //Imprimimos el valor de la variable Total
getch(); //Copiamos la pantalla
}
```

## Creando más de una variable a la vez:

Se puede declarar más de una variable a la vez, separándolas por comas:

```
#include<iostream.h> //Librería para cout y cin
#include<conio.h> //Librería para clrscr() y getch()

int main()
{
    unsigned int miEdad,tuEdad; //Declaramos dos variables en la misma línea
    long Dia,Mes,Anho; //Declaramos tres variables en la misma línea
    getch(); //Copiamos la pantalla
}
```

## Asignando valores a las variables:

Para asignarle valores a una variable, se utiliza el símbolo "=":

```
#include<iostream.h> //Librería para cout y cin
#include<conio.h> //Librería para clrscr() y getch()

int main()
{
    unsigned short clases = 5; //Declaración de variables
    long total;
    int alumnos = 25; //Asignamos el valor 25 a la variable Alumnos
    total = clases * alumnos; //Asignamos a la variable Total, la multiplicación de
    //las variables Clases por Alumnos
    getch();
}
```



Otro ejemplo:

```
#include<iostream.h> //Librería para cout y cin
#include<conio.h>      //Librería para clrscr() y getch()

int main()
{
    unsigned short int Ancho = 5,Largo; //Declaración de variables
    Largo = 10;

    unsigned short int Area = (Ancho * Largo); /*Declaramos la variable Area y le
    asignamos el valor de la multiplicación de la variable Ancho por la variable
    Largo*/

    cout<<"Ancho:"<<Ancho<<"\n";    // "\n" Sirve para bajar una línea.
    cout<<"Largo:"<<Largo<<"\n";    //Imprimimos los valores de las variables
    cout<<"Area:"<<Area<<"\n";
    getch();    //Copiamos la pantalla
}
```

### Usando typedef:

typedef, es un comando que te permite crear un alias para la declaración de funciones, esto es, que puedes crear un alias para unsigned short int y evitar tener que escribirlo cada vez que lo necesites.

```
#include<iostream.h> //Librería para cout y cin
#include<conio.h>      //Librería para clrscr() y getch()
typedef unsigned short int USHORT; //Declaramos un tipo de variable
//utilizando typedef

int main()
{
    USHORT Ancho = 5; //Utilizamos el tipo de variable que hemos creado para
    USHORT Largo = 2; //poder crear nuevas variables
```

```
USHORT Area = Ancho * Largo; //Asignamos a la variable Area, el valor de la
//multiplicación de la variable Ancho por la variable Largo
cout<<"Area: "<<Area<<endl;    //endl, cumple la función de "\n"
getch(); //Copiamos la pantalla
}
```

### Caracteres especiales:

- \* "\n" --> Nueva Línea o Salto de Carro
- \* "\t" --> Tab o espacio
- \* "\b" --> Backspace o retroceder una línea
- \* "\"" --> Comilla doble
- \* "\"" --> Comilla simple
- \* "\?" --> Signo de interrogación
- \* "\\" --> backslash

### Constantes:

Al igual que las variables, las constantes guardan información, pero a diferencia de ellas, esta información no puede ser modificada.

#### Constantes Literales:

Una constante literal, es un valor ingresado directamente en tu programa, en cualquier lugar que lo necesite.

```
int miEdad = 23;
```

miEdad es una variable de tipo int (Entero); 39 es una constante literal. No puedes asignar un valor a 39 y su valor no puede cambiarse.

#### Constantes Simbólicas:

Una variable simbólica, es una variable que está representada por un nombre, tal como una variable está representada. A diferencia de una variable, sin embargo, su valor no puede ser modificado.

```
Estudiantes = clases * estudiantesporclase;
```

clases y estudiantesporclase son variables simbólicas. En cada lugar que el programa lea estudiantes, asignará el valor de clases \* estudiantesporclase.

### Definiendo constantes con "const":

Utilizar const es muy útil, ya que se le puede dar un tipo a las variable y volverla constante.

```
const unsigned short int estudiantesporclase = 15;
```

### Constantes Enumeradas:

Las constantes enumeradas, te permiten crear nuevos tipos y luego definir dichas variables con valores que están restringidos a un grupo de posibles variables.

```
enum COLOR {ROJO,AZUL,VERDE,BLANCO,NEGRO};
```

COLOR es el nombre de la enumeración, en otras palabras, un nuevo tipo. ROJO es una variable simbólica con el valor "0", AZUL="1", etc. Se pueden inicializar con el valor que uno elija. {ROJO=100,AZUL,VERDE=500};

Ejemplo:

```
#include<iostream.h>    //Librería para cout y cin
#include<conio.h>        //Librería para clrscr() y getch()

int main()
{
enum Dias {Domingo,Lunes,Martes,Miercoles,Jueves,Viernes,Sabado};
//El enum, es una variable que contiene multiples valores, identificados por un
//índice
int opcion;
cout<<"Ingresa un día (0-6): ";
cin>>opcion; //Leemos un valor ingresado por teclado
if(opcion == Domingo || opcion == Sabado)
//Si el número que hemos ingresado es igual al valor de los índices (0 -
//Domingo) o (6 - Sabado), ejecuta una acción
```

```
cout<<"Ya estás en el fin de semana"; //Mostramos un mensaje  
else //Si el número ingresado, esta fuera del rango (0-6)  
cout<<"\nTodavía sigues en clases"; //Mostramos un mensaje  
getch(); //Copiamos la pantalla  
}
```

## Capítulo 4 - Expresiones y Secuencias

### Que son Secuencias:

En C++, una secuencia controla la ejecución, evalúa una expresión, o no hace nada.

```
x = a + b;
```

A diferencia del álgebra, en la cual  $x$  sería igual a " $a + b$ ", en C++,  $x$  obtendrá el valor de la suma de  $a + b$ .

### Bloques y Secuencias compuestas:

En cualquier lugar que puedas poner una secuencia, puedes poner una secuencia compuesta. Una secuencia compuesta, ejecuta una serie de expresiones. Comienza con una `{` y termina con una `}`.

```
{  
temp = a;    //En este ejemplo, al utilizar una variable temporal,  
a = b;       //podemos intercambiar el valor de A por el de B y el  
b = temp;    //valor de B por el de A.  
}
```

### Expresiones:

Una expresión debe devolver un valor, por lo tanto  $3+2$  devuelve el valor 5. Todas las expresiones son secuencias.

```
3.2 //Devuelve el valor 3.2
```

```
PI //Devuelve el valor 3.14
```

```
SegundosPorMinuto //Constante entera que devuelve el valor 60
```

**Operadores:**

Un operador es un símbolo que obliga al compilador a tomar una acción.

**Operador de Asignación:**

El operador de asignación provoca que la variable izquierda tome el o los valores de las variables derechas.

```
x = a + b;  
z = 10 - x;
```

**Operadores Aritméticos:**

Existen cinco operadores matemáticos: suma(+), resta(-), multiplicación(\*), división(/) y módulo(%).

```
x = 5 + 5; //10  
y = 10 - 8; //2  
z = 6 * 4; //24
```

**División Entera y Módulo:**

La división entera, toma solamente la parte entera de la división. La división de módulo, toma solamente el residuo de la división.

```
x = 10 / 5; //2  
y = 10 % 5; //0  
z = 6 % 4; //2
```

**Incremento y Decremento:**

El operador incremento (++) eleva el número en un dígito. El operador decremento (--) lo reduce en un dígito.

```
c++;  
c = c + 1;  
//Son equivalentes
```

**Prefix y Postfix:**

El prefix se escribe antes del nombre de la variable, e incrementa el valor y luego lo asigna. El postfix se escribe después de la variable, y asigna el valor y luego lo incrementa.

```
int a = ++x; //Prefix
```

```
int b = x++; //Postfix
```

**Otro ejemplo:**

```
#include <iostream.h> //Librería para cout y cin
#include <conio.h>      //Librería para clrscr() y getch()

int main()
{
    int miEdad = 23; //Declaración de variables
    int tuEdad = 23;
    cout<<"Yo tengo: "<<miEdad<<" años de edad.\n";
    cout<<"Tu tienes: "<<tuEdad<<" años de edad.\n";
    miEdad++; //Asignamos el valor y lo incrementamos en 1
    ++tuEdad; //Incrementamos el valor en 1 y lo asignamos
    cout<<"Un año pasó...\n";
    cout<<"Yo tengo: "<<miEdad<<" años de edad.\n";
    cout<<"Tu tienes: "<<tuEdad<<" años de edad.\n";
    cout<<"Un año pasó...\n"; //Mostramos un mensaje
    cout<<"Yo tengo: "<<miEdad++<<" años de edad.\n";
    cout<<"Tu tienes: "<<++tuEdad<<" años de edad.\n";
    cout<<"Escribamoslo de nuevo.\n"; cout<<"Yo tengo: "<<miEdad<<" años de
    edad.\n";
    cout<<"Tu tienes: "<<tuEdad<<" años de edad.\n";
    getch(); //Copiamos la pantalla
}
```

## Operadores Relacionales:

Son usados para determinar cuándo dos números son iguales o si uno es mayor que el otro:

|                 |    |           |             |
|-----------------|----|-----------|-------------|
| Igualdad        | == | 100 == 50 | → falso     |
| Desigualdad     | != | 100 != 50 | → verdadero |
| Mayor que       | >  | 100 > 50  | → verdadero |
| Mayor igual que | >= | 100 >= 50 | → verdadero |
| Menor que       | <  | 100 < 50  | → falso     |
| Menor igual que | <= | 100 <= 50 | → falso     |

## El operador IF:

Este operador examina una condición dada y ejecuta una acción determinada:

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()
int main()
{
    int miNum=5,tuNum;    //Declaración de variables
    cout<<"Ingresa un número:"; //Mostramos un mensaje en la pantalla
    cin>>tuNum;          //Leemos el valor ingresado por teclado
        if(tuNum > miNum)
//Comparamos si el valor de la variable tuNum es mayor que el valor de miNum
            cout<<"\nGanaste, tu número es mayor que el mio";
    getch(); //Copiamos la pantalla
}
```



**El operador ELSE:**

Este operador es utilizado junto con IF, si la condición no se cumple se ejecuta otra acción:

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()
int main()
{
    int miNum = 5, tuNum;    //Declaración de variables
    cout<<"Ingresa un número:";    //Mostramos un mensaje
    cin>>tuNum;            //Leemos un valor ingresado por teclado
        if(tuNum > miNum)
            //Comparamos si la variable tuNum es mayor que la variable miNum
            cout<<"\nGanaste, tu número es mayor que el mio";
        else
            //En el caso de la variable tuNum sea menor que la variable miNum
            cout<<"Gané!, Mi número es mayor que el tuyo";
    getch();    //Copiamos la pantalla
}
```

**Cuándo usar llaves en IF-ELSE:**

Por lo general, una simple instrucción IF-ELSE, no necesita llaves, pero cuando las instrucciones que va a ejecutar son mayores de una línea o hay IF dentro de otros IF, debes utilizar llaves para evitar la confusión y el código pueda ejecutarse:

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()
int main()
{
    int miNum = 5, tuNum;    //Declaración de variables
    cout<<"Ingresa un número:";    //Mostramos un mensaje
    cin>>tuNum;    //Leemos el valor ingresado por teclado
```

```

        if(tuNum > miNum)
//Comparamos si la variable tuNum es mayor que la variable miNum
        {
            cout<<"\nGanaste, tu número es mayor que el mio\n";
            cout<<"Que te parece si empezamos otra vez";
        }
    else
//En el caso de la variable tuNum sea menor que la variable miNum
        {
            if( tuNum < 1 )
//No podemos ingresar números negativos
            cout<<"El cero y los números negativos no juegan!";
            else
//Si el número ingresado es positivo
            cout<<"Gané!, Mi número es mayor que el tuyo";
        }
    getch(); //Copiamos la pantalla
}

```

### Operadores Lógicos:

Los operadores lógicos son usados para concatenar operadores relacionales:

|     |               |                            |
|-----|---------------|----------------------------|
| AND | &&            | → expresión1 && expresión2 |
| OR  | //(alt + 124) | → expresión1    expresión2 |
| NOT | !             | → !expresión1              |

#### AND:

Sólo es verdadero cuando ambas expresiones son verdaderas:

```
if((5 == 5) && (8 == 8))
```

**OR:**

Basta que una expresión sea verdadera para que sea verdadero:

```
if((5 == 5) || (6 == 8))
```

**NOT:**

Si es verdadera, la vuelve falsa y viceversa

```
if(5 != 5) o if(!(x == 5))
```

**El operador condicional(Ternario):**

Es un IF-ELSE automático:

```
(expresión1)?(expresión2):(expresión3)
```

Si la expresión1 es verdadera, devuelve el valor de la expresión2, si es falsa, devuelve el valor de la expresión3.

```
#include <iostream.h> //Librería para cout y cin
#include <conio.h>     //Librería para clrscr() y getch()
int main()
{
    int x,y,z; //Declaración de variables
    cout<<"Ingresa dos números.\n"; //Mostramos un mensaje
    cout<<"Primero: ";
    cin>>x; //Leemos un valor ingresado por pantalla
    cout<<"Segundo: ";
    cin>>y;
    if(x > y) //Si la variable X es mayor que la variable Y
        z = x; //Asignamos el valor de X a Z
    else //En caso contrario
        z = y; //Asignamos el valor de Y a Z
    cout<<"z: "<<z; //Imprimimos el valor de Z
```

```
    z = (x > y) ? x : y; //Operador Ternario
//Si X es mayor que Y, asignamos el valor de X a Z, en caso contrario
//asignamos el valor de Y a Z
cout<<"z: "<<z;
getch(); //Copiamos la pantalla
}
```

## Capítulo 5 - Funciones

### ¿Qué es una función?

Una función es un subprograma que puede interactuar con los datos y devolver un valor (solamente un valor). Las funciones pueden devolver un valor (int, long, float) o no devolver nada (void), además pueden recibir parámetros con los cuáles interactuar (int, char, long) o no recibir ninguno.

```
unsigned short int EncontrarArea(int largo, int ancho);
```

-----

Tipo que retorna      Nombre      Parámetros que recibe

### Ejemplo de una función

```
#include <iostream.h>      //Librería para cout y cin
#include <conio.h>          //Librería para clrscr() y getch()

int Area(int largo, int ancho) //Prototipo de la función
{
    return largo * ancho;
//La función devuelve el valor de la multiplicación de Largo por Ancho
}

int main()
{
    int largoTerreno, anchoTerreno, areaTerreno; //Declaración de variables
    clrscr();          //Borra la pantalla
    cout<<"\nQue tan ancho es tu terreno? "; //Mostramos un mensaje
    cin>>anchoTerreno; //Leemos un valor ingresado por teclado
    cout<<"\nQue tan largo es tu terreno? ";
    cin>>largoTerreno;

    areaTerreno = Area(largoTerreno, anchoTerreno); //Llamamos a la función
```

```
//y asignamos el valor devuelto a la variable areaTerreno
cout<<"\nTu terreno tiene ";
cout<<areaTerreno;
cout<<" pies cuadrados\n\n";
getch();    //Copiamos la pantalla
}
```

## Variables Locales y Globales

Las variables pueden estar definidas en dos capas diferentes de un programa, las variables locales, pertenecen a una función y solo pueden ser modificadas o utilizadas por esta, ya que al terminar la función, su valor desaparece. Las variables globales en cambio, pueden ser utilizadas por cualquier función o método del programa, lo cual las hace peligrosas, porque cualquiera puede cambiar su valor.

## Ejemplo de variables locales y globales

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()

int altura,peso;         //Variables Globales

int Edad(int edad)       //Prototipo de la función
{
    return edad;         //Devuelve el valor de la variable Edad
}
```

```
int main()
{
    clrscr();      //Borra la pantalla
    int edad,nedad;  //Variables Locales
        cout<<"Cual es tu altura?";  //Mostramos un mensaje
        cin>>altura;                  //Leemos el valor ingresado por teclado
        cout<<"Cual es tu peso?";
        cin>>peso;
        cout<<"Cual es tu edad?";
        cin>>edad;
        nedad = Edad(edad); //Asignamos a la variable nedad, el resultado
//que devuelve la función Edad
        cout<<"Tu altura es "<<altura;
        cout<<" Tu peso es "<<peso;
        cout<<" y tu edad es "<<nedad;
    getch();      //Copiamos la pantalla
}
```

## Capítulo 6 - Clases

### ¿Qué es una clase?

Una clase es una declaración que sirve como un contenedor para nuevas variables creadas por nosotros mismos. Que pueden a su vez, contener a otras variables, ya sean de tipo int, long, char, etc. Y pueden contener métodos y propiedades, lo cual ayuda a tener un mejor control sobre ellas. Las clases pueden representar situaciones o elementos de la vida real, lo cual nos ayuda a elaborar mejores programas y que son por ende más fáciles de mantener y actualizar. Cada elemento que forma la clase, es llamado Objeto.

### Ejemplo de una clase

```
class Estudiante
{
    unsigned int codigo;
    unsigned int edad;
    void Comentarios();
}
```

### Definir un objeto

Los objetos se declaran de la misma manera en que se declararía una variable normal.

```
Estudiante Alvaro; //Variable Alvaro de tipo Estudiante
```

### Acceder a los miembros de las clases

```
Alvaro.codigo = 108;
```

```
Alvaro.edad = 24;
```

### Miembros Privados y Miembros Públicos

Todos los miembros de una clase, son privados por default, es decir, que si no se especifica lo contrario, van a ser privados.



Para poder acceder y asignar valores a los miembros de la clase, estos deben ser públicos.

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()

class Estudiante        //Creamos una clase
{
public:                  //Definimos sus elementos como públicos
    unsigned int codigo; //Declaración de variables de la clase
    unsigned int edad;

};

int main()
{
    Estudiante Alvaro; //Creamos un objeto o instancia de la clase
    Alvaro.edad = 24;  //Llamamos a la variable edad de la clase y le
//asignamos un valor
    cout<<"El estudiante Alvaro, tiene "<<Alvaro.edad<<" años";
//Mostramos en pantalla, el valor de la variable edad, que pertenece al objeto
//Alvaro
    getch(); //Copiamos la pantalla
}
```

### Conservar algunos datos como Privados

Para un mejor control del programa, se deben dejar algunos datos como privados y crear funciones que accedan a ellos y nos permitan trabajar.

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()

class Estudiante        //Creamos nuestra clase
{
public:                  //Variables de clase públicas
```

```

        unsigned int gCodigo();      //Este método, obtendrá el valor de la
//variable codigo
        void sCodigo(unsigned int Cod); //Este método, asignará un valor a la
//variable codigo

        unsigned int gEdad();
        void sEdad(unsigned int Edad);

        void Comentarios(); //Este método mostrará un mensaje en la pantalla

private:
        unsigned int Codigo;
        unsigned int Edad;
};

```

\* Para acceder a estos datos y modificarlos se utiliza lo siguiente:

**Estudiante Alvaro;**

**Alvaro.sCodigo(108); //Se asigna el valor "108" a la variable sCodigo**

### Implementando los metodos de la clase

A continuación está un ejemplo de como implementar nuestra clase Estudiante.

```

#include <iostream.h>      //Librería para cout y cin
#include <conio.h>          //Librería para clrscr() y getch()

class Estudiante          //Creamos nuestra clase
{
public:                   //Declaración de variables de la clase pública
        unsigned int gCodigo(); //Obtenemos el valor de Codigo
        void sCodigo(unsigned int Cod); //Establecemos el valor de Codigo
        void Comentarios(); //Mostramos un mensaje en la pantalla

```

```

private:           //Declaración de variable de la clase privadas
    unsigned intCodigo;
};

unsigned int Estudiante::gCodigo() //Este método pertenece a la clase
{
    return Codigo; //Devuelve el valor de la variable Codigo
}

void Estudiante::sCodigo(unsigned int Cod) //Recibe un parámetro
{
    Codigo = Cod; //Asigna a la variable Codigo, el valor del parámetro
}

void Estudiante::Comentarios()
{
    cout<<"Alvaro Tejada. Programador\n"; //Muestra un mensaje
}

int main()
{
    Estudiante Alvaro; //Creamos un objeto o instancias de nuestra clase
    Alvaro.sCodigo(108); //Establecemos el valor de la variable Codigo
    cout<<"El código de Alvaro es: "<<Alvaro.gCodigo()<<"\n";
    //Mostramos el valor de la variable Codigo
    Alvaro.Comentarios(); //Mostramos un mensaje
    getch(); //Copiamos la pantalla
}

```

## Constructores y Destructores

Para inicializar algún miembro de alguna clase, se utiliza un constructor, los constructores pueden recibir parámetros, pero no pueden retornar ningún valor, ni siquiera Void. Este constructor, debe tener el mismo nombre que la

clase.

Luego de haber declarado el constructor, debes declarar un destructor, para que libere la memoria utilizada. los destructores siempre tienen el nombre de la clase antepuestos por el símbolo ~ (alt+126), los destructores, no reciben argumentos, y tampoco retornan un valor.

```
#include <iostream.h> //Para las librerías cout y cin
#include <conio.h>      //Para las librerías clrscr() y getch()

class Estudiante      //Creamos nuestra clase
{
public:                //Declaración de variables de la clase pública
    Estudiante(int Codigo);          //Declaramos nuestro constructor
    ~Estudiante();                   //Declaramos nuestro destructor
    unsigned int gCod();              //Método para obtener el código
    unsigned int sCod(int cod);       //Método para establecer el código
    void Comentarios();               //Método que imprime un mensaje
private:               //Declaración de variables de la clase privada
    unsigned int vCod;    //Variable que representa al código
};

Estudiante::Estudiante(int Codigo) //Constructor, le da un valor por defecto
{
    vCod = Codigo; //Asignamos el valor del parámetro a una variable
}

Estudiante::~~Estudiante() //Destructor, debemos eliminar nuestro constructor
{
}

unsigned int Estudiante::gCod() //Este método nos devuelve el código
{
    return vCod;
}
```

```
unsigned int Estudiante::sCod(int cod) //Este método asigna un valor al código
{
    vCod = cod;
}

int main()
{
    clrscr(); //Borra la pantalla
    Estudiante Alvaro(108); //Creamos un objeto de la clase, con un valor por
    //defecto
    cout<<"El código de Alvaro es: "<<Alvaro.gCod(); //Imprimimos el código
    Alvaro.sCod(250); //Asignamos un nuevo código
    cout<<"\nEl nuevo código de Alvaro es: "<<Alvaro.gCod(); //Lo imprimimos
    getch(); //Copia la pantalla
}
```

## Estructuras

En realidad, las estructuras y las clases cumplen la misma función, por lo cual, pueden reemplazarse los comandos "class" por "struct". La diferencia básica es que en las Estructuras, los miembros son públicos por default. Este es solo un accidente histórico, en el paso de C a C++.

## Capítulo 7 - Secuencias de Control

### ¿Qué son las secuencias de control?

Las secuencias de control, son métodos o funciones, que nos permiten ejecutar una instrucción más de una vez, o por un número ilimitado de veces o hasta que se cumpla una condición dada, y que generalmente es matemática.

Para mi punto de vista, este es uno de los puntos más importantes, ya que casi todos o todos los lenguajes de programación los utilizan, aunque no por eso, significa que no sean fáciles de aprender y utilizar.

### While o Mientras

While o Mientras(En pseudo-código), permite ejecutar una instrucción o un conjunto de instrucciones mientras la condición dada sea verdadera.

```
#include <iostream.h> //Librería para cout y cin
#include <conio.h>      //Librería para clrscr() y getch()

int main()
{
    int counter = 0; //Declaramos una variable, para usar como contador

    while(counter < 5) //Mientras counter sea menor a 5
        // ejecuta la instrucción.
    {
        counter++; //Aumenta a counter en uno.
        cout<<"Hola\n"; //Imprimimos un mensaje
    }
    getch(); //Copia la pantalla
}
```

```

#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()

int main()
{
    clrscr(); //Para borrar la pantalla.
    int num; //Declaramos una variable de tipo entero
    cout<<"Ingresa un número del 1 al 10: ";
    cin>>num; //Leemos el valor ingresado por teclado
    while(num>0 && num<11) //Mientras num sea mayor a 0 y menor que 11
    {
        cout<<"\nTu número es num: "<<num; //Imprimimos el mensaje
        num++; //Aumentamos a nuestro contador en 1
    }
    getch();
}

```

### Do...While o Hacer...Mientras

Do...While o Hacer...Mientras(En pseudo-código). Aunque es similar al While, el Do...While, tiene la ventaja de que ejecuta las instrucciones antes de verificar si la condición se cumple, por lo cual, siempre se ejecutará por lo menos una vez.

```

#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()

int main()
{
    clrscr(); //Para borrar la pantalla.
    int num; //Declaramos una variable de tipo entero

    cout<<"Ingresa un número del 1 al 10: ";
    cin>>num; //Leemos el valor ingresado por el teclado
}

```

```

do //Hacerlo una vez
{
    cout<<"\nTu numero es num: "<<num;
    num++; //Aumentamos nuestro contador en 1
}while(num > 0 && num < 11); //Y verificar si num es mayor a 0
                                //y menor que 11, para seguir ejecutándolo
getch(); //Copia la pantalla
}

```

### For o Para

El For o Para(En pseudo-código), se utiliza para asignar un valor, verificar una condición y aumentar la variable, todo en una sola línea.

```

#include <iostream.h> //Librería para cout y cin
#include <conio.h>     //Librería para clrscr() y getch()

int main()
{
    clrscr(); //Para borrar la pantalla.
    int num;
    for(num = 0; num < 6; num++) //Para num = 0, mientras que num sea
//menor que 6, aumentamos num en 1
        cout<<"\nEl número es: "<<num; //Imprimimos el valor de num
    getch(); //Copia la pantalla
}

```

### Switch o Casos

El Switch o Casos(En pseudo-código), permite escoger entre una amplia gama de posibilidades, sin tener que ejecutar millones de líneas de if, else, etc.

Es muy útil a la hora de generar menues, que piden al usuario, escoger una opción.



```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()

int main()
{
    clrscr();    //Limpia la pantalla
    int opcion; //Declaramos una variable de tipo entero
    do //Hacer
    {
        cout<<"1)\n";
        cout<<"2)\n";
        cout<<"3)\n";
        cout<<"Escoja una opción: ";
        cin>>opcion; //Leemos el valor ingresado por teclado
    }while(opcion<0 || opcion>4); //Ejecutamos el bloque mientras la
//opción sea menor que 0 o mayor que 4
    switch(opcion) //Verifica el valor de opción
    {
        case 1: //Si es uno
            cout<<"Eligio la opción 1";
            break; //El break sirve para que no ejecute
            //las demas opciones.
        case 2: //Si es dos
            cout<<"Eligio la opción 2";
            break;
        case 3:
            cout<<"Eligio la opción 3";
            break;
        default:
            cout<<"Esa no es una opción valida";
            break;
    }
    getch();} //Copia la pantalla
```

## Capítulo 8 - Punteros

### ¿Qué son los punteros?

Un puntero es una variable que guarda una dirección de memoria. En realidad este no es un punto que domino, porque en realidad es bastante complicado sobre todo para los principiantes en programación, de todos modos, trataré de explicarlos de la mejor manera posible.

### Ejemplo de las Direcciones de Memoria

Cada variable está asignada a un espacio en memoria determinado, con los punteros, nosotros podemos especificar en que espacio de memoria localizar nuestras variables.

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()

int main()
{
    unsigned short shortVar = 5; //Declaramos una variable

    cout<<"shortVar: "<<shortVar; //Imprimimos el valor de la variable
    cout<<"Direccion de shortVar: "<<&shortVar; //Imprimimos la dirección
//de la variable
    getch();
}
```

/\*El símbolo & delante del shortVar, nos da la dirección que tiene en memoria shortVar\*/

## ¿Cómo guardar esa dirección en un Puntero

Como mencioné, toda variable declarada tiene una dirección en memoria, y esta puede ser almacenada en un puntero.

```
int howMany = 8; //Creamos e inicializamos una variable
int * pMany = 0; //Creamos e inicializamos un puntero, lo más
//recomendable, es siempre comenzar inicializándolo con cero, para
//evitar crear un Wild o Stray Pointer.
pMany = &howMany; //Se le asigna al puntero, la dirección de la variable.
```

//También se puede declarar e inicializar al mismo tiempo.

```
int howMany = 8;
int * pMany = &howMany;
```

## Para que usar Punteros

Los punteros son muy usados, para manejar datos en el área libre de memoria, acceder a datos y funciones de clases, pasar variables por referencia a las funciones, etc.

### Operador New

El operador New, se utiliza para crear un objeto en el área libre de memoria. El resultado del operador New, es una dirección de memoria que debe ser asignada a un puntero. Más adelante veremos para que nos sirve esto exactamente

```
unsigned long int * pPointer; //Se declara el tipo del puntero
pPointer = new unsigned long int; //Se crea en la memoria libre
```

```
*pPointer = 77; //Se le asigna un valor
```

### El operador Delete

Cuando terminas de utilizar un puntero, debes regresarlo al área libre de memoria, esto se hace utilizando el operador Delete. Los punteros no se eliminan automáticamente y pueden provocar una pérdida innecesaria de memoria.

```
delete pPointer; //Eliminamos el puntero, liberando memoria
```

### Ejemplo

```
#include <iostream.h> //Librería para Cout y Cin
#include <conio.h>      //Librería para clrscr() y getch()

int main()
{
    int iVal = 8;      //Declaramos un entero y le asignamos un valor
    int * pLocal = &iVal; //Declaramos un puntero y le asignamos la
//dirección de nuestra variable
    int * pStore = new int; //Creamos un nuevo puntero sin asignar
    *pStore = 5; //Le asignamos un valor
    cout<<"iVal: "<<iVal<<"\n"; //Imprimimos nuestra variable
    cout<<"*pLocal: "<<*pLocal<<"\n"; //Imprimimos nuestro puntero
    cout<<"*pStore: "<<*pStore<<"\n"; //Imprimimos el nuevo puntero
    delete pStore; //Liberamos la memoria de nuestro puntero
    pStore = new int; //Volvemos a asignarle memoria
    *pStore = 3; //Le asignamos un valor
    cout<<"*pStore: "<<*pStore<<"\n"; //Lo imprimimos
    delete pStore; //Liberamos la memoria del nuevo puntero
    delete pLocal; // Liberamos la memoria de nuestro puntero
    getch(); //Copiamos la pantalla
}
```

### Creando y Eliminando Objetos en el área libre de memoria

Los objetos en el área libre de memoria, se crean de la misma manera que las variables, se crean, se inicializan y se se asignan.

Cada vez que se llama al operador New sobre un objeto, se llama al default constructor y cuando se usa el operador delete, se llama al destructor.

```
#include <iostream.h> //Librería para Cout y Cin
#include <conio.h>      //Librería para clrscr() y getch()

class Estudiante //Creamos una clase, llamada Estudiante
{
public: //Declaramos sus métodos públicos
    Estudiante(); //Constructor
    ~Estudiante(); //Destructor
private: //Declaramos sus variables privadas
    int vCod; //Código del estudiante
};

Estudiante::Estudiante() //Definimos al constructor
{
    cout<<"LLamando al constructor.\n";
    vCod = 108; //Asignamos un valor a la variable del código
}

Estudiante::~~Estudiante() //Definimos el destructor
{
    cout<<"LLamando al destructor.\n"; //Solo muestra un mensaje
}

int main()
{
    cout<<"Estudiante Alvaro...\n";
    Estudiante Alvaro; //Creamos un objeto de la clase Estudiante
    cout<<"Estudiante *pSid = new Estudiante...\n";
    Estudiante *pSid = new Estudiante; //Creamos un puntero a un objeto
    //de la clase Estudiante
    cout<<"delete pSid...\n";
    delete pSid; //Liberamos la memoria del puntero
    cout<<"Terminando la demostración...\n";
    getch(); //Copiamos la pantalla
}
```

## Accesando a los miembros de Datos

Para poder acceder a los miembros de datos de un objeto creado en al área libre de memoria, se debe utilizar al operador `->`.

```
#include <iostream.h> //Librería para cout y cin
#include <conio.h>      //Librería para getch() y clrscr()

class Estudiante //Declaramos nuestra clase
{
public: //Miembros públicos de la clase
    Estudiante() {vCode = 108;} //Declaración con función en línea
    ~Estudiante() {}
    unsigned int gCode() const {return vCode;} //Al ser constante, no se
    //puede modificar
    void sCode(unsigned int cod) {vCode = cod;}
private: //Miembros privados de la clase
    unsigned int vCode;
};

int main()
{
    Estudiante * pAlvaro = new Estudiante; //Creamos un puntero a un objeto de la
    //clase estudiante
    cout<<"El código de Alvaro es: "<<pAlvaro->gCode(); //Obtenemos el código
    pAlvaro->sCode(205); //Establecemos el valor del código
    cout<<"El nuevo código de Alvaro es: "<<pAlvaro->gCode(); //Obtenemos el
    //código nuevamente
    delete pAlvaro; //Borramos el puntero al objeto de la clase que creamos
    getch(); //Copia la pantalla
}
```

Cuando declaramos un función "InLine", o "En Línea", simplemente estamos creando el método conjuntamente con su asignación, además, esto hace que

cada vez que el método es llamado, se envía a si mismo, en vez de generar una copia que puede reducir la memoria cada vez que vuelva a ser llamado.

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para getch() y clrscr()

class Estudiante    //Declaramos nuestra clase
{
public:    //Miembros públicos de la clase
    Estudiante(); //Constructor
    ~Estudiante(); //Destructor
    unsigned int gCode() const {return *vCode;} //Declaración con función
    //en línea que devuelve un puntero al código y es constante
    void sCode(unsigned int cod) {*vCode = cod;}
private:    //Miembros privados de la clase
    unsigned int *vCode;
};

Estudiante::Estudiante() //Definimos al constructor
{
    vCode = new unsigned int(108); //Asignamos un valor
}

Estudiante::~~Estudiante() //Definimos al destructor
{
    delete vCode; //Borramos el puntero al código
}
```

```
int main()
{
    Estudiante * Alvaro = new Estudiante; //Creamos un puntero a un
//objeto de la clase estudiante
    cout<<"El código de Alvaro es: "<<Alvaro->gCode(); //Obtenemos el
//código
    Alvaro->sCode(205); //Establecemos el valor del código
    cout<<"El nuevo código de Alvaro es: "<<Alvaro->gCode();
//Obtenemos el código nuevamente
    delete Alvaro; //Borramos el puntero al objeto de la clase
    getch(); //Copiamos la pantalla
}
```



## Capítulo 9 - Referencias

### ¿Qué son las referencias?

Las referencias son alias que se le pueden dar una variable, es como un sobrenombre y todo lo que se le haga al alias le ocurrirá a la variable y viceversa. Se reconocen porque al comenzar el nombre de la variable, se les antepone el símbolo &.

### Ejemplo de Creación y uso de Referencias

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()

int main()
{
    int miNum;    //Creamos una variable
    int &rNewNum = miNum;    //Creamos una referencia para la variable que
    //creamos arriba

    miNum = 14;    //Asignamos un valor a nuestra variable
    cout<<"miNum: "<<miNum<<"\n";    //Imprimimos la variable
    cout<<"rNewNum: "<<rNewNum<<"\n";    //Imprimimos la referencia

    NewNum = 24;    //Asignamos un valor a la referencia
    cout<<"miNum: "<<miNum<<"\n";    //Imprimimos la variable
    cout<<"rNewNum: "<<rNewNum<<"\n";    //Imprimos la referencia

    getch();    //Copia la pantalla
}
```

## referenciar Objetos

Al igual que las variables se pueden referenciar, los objetos de una clase también.

```
#include <<iostream.h> //Librería para cout y cin
#include <<conio.h>      //Librería para clrscr() y getch()

class Estudiante //Creamos nuestra clase
{
public: //Miembros públicos de la clase
    Estudiante(int code,int edad); //Constructor
    ~Estudiante() {} //Destructor
    int gCode() {return vCode;} //Método para obtener el código
    int gEdad() {return vEdad;} //Método para obtener la edad
private: //Miembros privados de la clase
    int vCode;
    int vEdad;
};

Estudiante::Estudiante(int code,int edad) //Definición del constructor
{
    vCode = code;
    vEdad = edad;
}

Estudiante::~~Estudiante() //Definición del destructor
{
}
```

```

int main()
{
    Estudiante Alvaro(108,26); //Creamos un objeto de la clase, pasándolo como
    //parámetros al constructor, el código y la edad
    Estudiante &rAlv = Alvaro; //Creamos una referencia al objeto que hemos
    //creado

    cout<<"El código de Alvaro es: "<<Alvaro.gCode(); //Imprimos el código
    cout<<"\ny su edad es: "<<rAlv.gEdad(); //Imprimos la edad

    getch(); //Copiamos la pantalla
}

```

## Retornando múltiples valores

Como sabemos, una función solo puede retornar un valor, pero que pasa si necesitamos retornar más de uno, talvez dos o tres... Con las referencias podemos hacerlo.

```

#include <iostream.h> //Librería para cout y cin
#include <conio.h>     //Librería para clrscr() y getch()

int Potencias(int, int &, int &); //Definimos la cabecera de la función

int main()
{
    int numero,cuadrado,cubo;

    cout<<"Ingresa un número positivo: ";
    cin>>numero; //Leemos el valor ingresado por teclado
}

```

```
Potencias(numero, cuadrado, cubo); //Llamamos al función

cout<<"numero: "<<numero<<"\n"; //Imprimimos los valores
cout<<"cuadrado: "<<cuadrado<<"\n";
cout<<"cubo: "<<cubo<<"\n";

getch(); //Copiamos la pantalla
}

int Potencias(int numero, int &rCuadrado, int &rCubo)
//Dos de los tres parámetros, son referencias
{
rCuadrado = numero*numero;
rCubo = numero*numero*numero;
}
```

### Cuando usar Referencias y cuando Punteros

Las referencias no pueden ser reasignadas, tampoco pueden ser nulas. Por lo general, recomiendo utilizar Referencias, porque son más sencillas y más seguras, pero si cumple alguna de las dos limitaciones es necesario utilizar punteros.

## Capítulo 10 - Herencia

### ¿Qué es la Herencia?

La herencia puede describirse como un concepto de la vida real, por ejemplo, de los de los animales, podemos sacar a los mamíferos, de los mamíferos a los gatos, y de los gatos a los gatos salvajes de las estepas nor africanas y de estos últimos a mi gato "Cheko". Podemos decir que "Cheko" descende de toda esta línea de clasificaciones y por lo tanto debe tener también sus características, tanto de los gatos como de los mamíferos. En las Herencias, las clases, heredan atributos y funciones de otras clases que son superiores a ellas.

### Ejemplo de Herencia

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para getch() y clrscr()

class Mamiferos          //Definimos nuestra clase madre
{
public:                  //Métodos públicos de la clase
    //Constructores
    Mamiferos():vEdad(2),vPeso(4){}
    //Destructor
    ~Mamiferos(){}

    int gEdad() const {return vEdad;}    //Funciones InLine
    void sEdad(int Edad) {vEdad = Edad;}
    int gPeso() const {return vPeso;}
    void sPeso(int Peso) {vPeso = Peso;}

    void Hablar() const {cout<<"Mamifero hablando\n";}

protected:            //Variables protegidas
    int vEdad;
    int vPeso;
```

```

};

class Gato:public Mamiferos //La clase Gato, hereda de la clase Mamíferos
{
public: //Métodos públicos de la clase
    Gato() {} //Constructor
    ~Gato() {} //Destructor

    void Dormir() const {cout<<"Gato durmiendo...\n";}
};

int main()
{
    clrscr(); //Borramos la pantalla
    Gato Cheko; //Creamos nuestro objetos "Cheko" de la clase "Gato"
    Cheko.Hablar(); //Llamamos a un método de la clase Mamíferos
    Cheko.Dormir(); //Llamamos a un método de la clase Gato
    cout<<"Cheko tiene "<<Cheko.gEdad()<<" años y pesa "<<Cheko.gPeso();
    cout<<" kilos.";

    getch(); //Copiamos la pantalla
}

```

### Private contra Protected

Los miembros de datos de una clase que son declarados como Private, no pueden ser accesados por las clases derivadas (En Herencia), por lo cual se utiliza el comando Protected:, esto hace que los miembros de datos continúen privados pero sean visibles para las clases derivadas.

## Métodos Virtuales

Los métodos virtuales, sirven para poder tener métodos con nombre iguales, tanto en la clase madre como en la clases derivada, sin que estos entren en conflicto, el compilador sabe a cual llamar dependiendo del contexto,

también permiten la utilización de punteros, que declaren una clase derivada como nueva instancia de la clase madre.

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()

class Mamiferos          //Definimos nuestra clase madre
{
public:    //Métodos públicos
    //Constructor
    Mamiferos():vEdad(2) {cout<<"Constructor Mamiferos...\n";}
    //Destructor Virtual
    virtual ~Mamiferos() {cout<<"Destructor Mamiferos...\n";}
    void Mover() const {cout<<"Mamiferos avanzan un paso\n";}
    //Método virtual
    virtual void Hablar() const {cout<<"Mamiferos hablando\n";}
protected:    //Variable protegida
    int vEdad;
};

class Gato:public Mamiferos //La clase gato hereda de la clase Mamiferos
{
public:    //Métodos públicos
    //Constructor
    Gato() {cout<<"Constructor Gato...\n";}
    //Destructor
    ~Gato() {cout<<"Destructor Gato...\n";}
    void Hablar() const {cout<<"Miau!\n";}
    void Mover() const {cout<<"Gato avanza 3 pasos\n";}
```

```
};

int main()
{
    Mamiferos *pGato = new Gato; //Creamos un puntero de una clase Mamifero
    //a una clase gato
    pGato->Mover();      //Llamamos al método. (Llama al de Mamíferos)
    pGato->Hablar();     //Llamamos al método. (Llama al de Gato, porque es
    //virtual)
    getch();             //Copia la pantalla
}
```



## Capítulo 11 - Arreglos o Matrices

### ¿Qué son los Arreglos o Matrices?

Los Arreglos o Matrices, pueden definirse con un conjunto de elementos, ordenados por un índice que van del 0 al n-1. Se representan por: Numeros[5], donde los índices van del 0 al 4. También como Array[3] = {1,2,3}, donde los índices van del 0 al 3.

### Ejemplo de Arreglos o Matrices

```
#include <iostream.h>           //Librería para cout y cin
#include <conio.h>               //Librería para clrscr() y getch()

int main()
{
    int Array[5];               /*Declaramos nuestras variables. Una arreglo se declara como
                                cualquier variable, solamente que debemos agregarle los [], indicando cuantos
                                números o caracteres puede almacenar.*/

    for(i = 0; i <5; i++)//Para i que vale 0, mientras i sea menor que 5, aumentar i
    //en 1
    {
        cout<<"Ingresa valores para el Arreglo: ";
        cin>>Array[i];        //Leemos en la matriz, los valores ingresados por
                                //teclado, guardando cada valor en un índice separado.
    }

    for(i = 0; i < 5; i++)//Para i que vale 0, mientras i sea menor que 5, aumentar i
    //en 1
    {
        cout<<i<<": "<<Array[i]<<"\n";
    }//Imprimimos los valores de la matriz, uno por uno
    getch();
}
```

## Inicializando Arreglos o Matrices

Un Arreglo o Matriz, puede inicializarse tanto con variables como por caracteres, estando cada uno de ellos, separados por comas.

```
int Array[5] = {1,2,3,4,5}; //Matriz de caracteres, con 5 elementos

int Array2[5] = {1,2,3}; //Matriz de caracteres con 5 elementos, pero solo 3
//declarados
```

## Declarando Arreglos o Matrices

Los Arreglos o Matrices, pueden tener cualquier nombre, pero no el mismo nombre que otra variable que ya haya sido declarada.

## Arreglo o Matriz de Objetos

Al igual que las variables o caracteres, los objetos también pueden ser almacenados en un arreglo o matriz.

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para getch() y clrscr()

class Gato                //Declaramos nuestra clase
{
public:                   //Miembros públicos de la clase
    Gato() {vEdad = 2; vPeso = 4;}    //Constructor con valores
    ~Gato() {}    //Destructor

    int gEdad() const {return vEdad;} //Funciones InLine
    int gPeso() const {return vPeso;}
    void sEdad(int Edad) {vEdad = Edad;}
private:                 //Miembros privados de la clase
    int vEdad;
    int vPeso;
};
```

```

int main()
{
    Gato Gatitos[5];    //Creamos un arreglo o matriz con nuestra clase
    int i;
    for(i = 0; i < 5; i++) //Para i que tiene el valor de 0, mientras sea
//menor que 5, aumentar el valor de i en 1.
        Gatitos[i].sEdad(2*1+1); //Asignamos un edad a cada uno

    for(i = 0; i < 5; i++) //Para i que tiene el valor de 0, mientras sea menor
//que 5, aumentar el valor de i en 1.
    {
        cout<<"\nGatito# "<<i+1<<" tiene: "<<Gatitos[i].gEdad()<<" años";
        //Imprimimos la edad de cada gatito.
    }
    getch(); //Copiamos la pantalla
}

```

### Arreglos o Matrices Multidimensionales

Para entender esto, pensemos en un tablero de Ajedrez, tiene 8 filas por 8 columnas, un total de 64 casilleros, esto podría representarse de dos maneras.

```

int Chess[64];           int Chess[8][8];

```

```

#include <iostream.h>    //Librería para cout y cin
#include <conio.h>       //Librería para clrscr() y getch()

int main()
{
    int Array[2][2];    //Declaramos una matriz o arreglo bidimensional
    int i,j;
    for(i = 0; i < 2; i++) //Creamos un for para las filas

```

```

        for(j = 0; j < 2; j++) //y otro para las columnas
        {
            cout<<"Ingresa valores a la matriz: ";
            cin>>Array[i][j]; //Guardamos los valores ingresados
//por teclado
        }
    for(i = 0; i < 2; i++) //Para leer los valores, volvemos a crear un for
//para las filas
        for(j = 0; j < 2; j++) //y otro para las columnas
        {
            cout<<"Los valores son["<<i<<"]"<<["<<j<<"]:"
            <<Array[i][j]<<"\n"; //Imprimimos los valores
        }
    getch(); //Copiamos la pantalla
}

```

### Arreglos o Matrices de Punteros

Estos son utilizados para poder almacenar los objetos en el área libre de memoria.

```

#include <iostream.h> //Librería para cout y cin
#include <conio.h> //Librería para clrscr() y getch()

class Gato //Declaramos nuestra clase
{
public: //Miembros públicos de la clase
    Gato() {vEdad = 2; vPeso = 4;} //Constructor con valores
    ~Gato() {} //Destructor

    int gEdad() const {return vEdad;} //Métodos InLine
    int gPeso() const {return vPeso;}
    void sEdad(int Edad) {vEdad = Edad;}
private: //Miembros privados de la clase

```

```

        int vEdad;
        int vPeso;
    };

int main()
{
    Gato * Gatitos[20]; //Declaramos un puntero a un arreglo o matriz de nuestra
    //clase
    int i;
    Gato * pCheko; //Declaramos un puntero a un objeto de nuestra clase
    for(i = 0; i < 20; i++) //Para i que tiene el valor de 0, mientras i sea menor que
    //20, le agregamos 1 al valor de i
    {
        pCheko = new Gato; //Creamos un nuevo objeto en el área libre de
        //memoria
        pCheko->sEdad(2*i+1); //Llamamos a nuestra método para asignar
        //un valor a nuestro objeto
        Gatitos[i] = pCheko; //Asignamos el valor del objeto a nuestro arreglo
        //o matriz
    }

    for(i = 0; i < 20; i++)//Para i que tiene el valor de 0, mientras i sea menor que
    //20, le agregamos 1 al valor de i
    {
        cout<<"Cheko# "<<i + 1<<": "<<pCheko->gEdad();
        //Imprimimos las edades de todos los gatitos en nuestro arreglo o
        //matriz
    }
    delete pCheko //Eliminamos el objeto de la memoria
    getch(); //Copiamos la pantalla
}

```

## Arreglos o Matrices de Caracteres

En C++, una palabra o cadena de caracteres, es realidad un arreglo o matriz de caracteres, terminando con el símbolo de final de línea o '\0'. Nosotros podemos declarar nuestra propia matriz o arreglo de caracteres.

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()

int main()
{
    char cadena[20];      //Declaramos una matriz o arreglo de tipo cadena
    cout<<"Ingresa la cadena de caracteres: ";
    cin>>cadena;          //Guardamos los valores ingresados por teclado
    cout<<"\nAquí está la cadena de caracteres: "<<cadena; //Imprimimos
    getch();              //Copiamos la pantalla
}
```

Cuando utilizamos un arreglo o matriz de caracteres, sin importar que tan grande sea este arreglo o matriz, si escribimos una palabra o frase que contenga espacios, solamente se tomará en cuenta la frase hasta antes de que comience el espacio. Es decir, si yo escribo esto:

```
Char Array[20];
```

Y escribo por teclado "Hola Mundo",

La matriz simplemente contendrá "Hola", puesto el espacio es un carácter de fin. Pero no se preocupen, mas adelante sabremos como almacenar nombres largos sin tener que preocuparnos de los espacios -;)

## Capítulo 12 - Polimorfismo

### ¿Qué es el Polimorfismo?

Cuando hablamos de herencia, dijimos, que podíamos tomar una clase y derivar otra de ella, haciendo que la clase derivada tomara las funciones y atributos de la clase base o primaria. Pues bien, que pasa si tenemos dos clases primaria y queremos que nuestra nueva clase derivada, derive de ambas, es decir, herede los atributos y los métodos de las clases de las cuales derivó. Por ejemplo, podemos tener la clase Caballo y la clase Ave, el primero galopa y el segundo vuela, pero supongamos que queremos crear una clase que derive de los dos, por ejemplo, un Pegaso, que puede cabalgar y volar, en Herencia, esto no sería posible, y de serlo, nos obligaría a crear varias instancias del mismo método, como en el caso de Volar(), que tendría que estar tanto en Ave como en Pegaso, o tan solo podría hacer una de las dos cosas, pues bien, con el Polimorfismo, podemos manejar que esas dos clases bases se fusionen para la clase derivada que vamos a crear.

```
#include <iostream.h> //Librería para cout y cin
#include <conio.h>      //Librería para clrscr() y getch()

class Caballo          //Declaramos nuestra clase
{
public:                //Miembros públicos de la clase
    Caballo() {}       //Constructor
    ~Caballo() {}      //Destructor
    virtual void Galopar() const {cout<<"El caballo galopa 2 pasos";}
    //Declaramos un método virtual que puede ser declarado nuevamente
//en otra clase
};

class Ave              //Declaramos otra clase
{
public:                //Miembros públicos de la clase
```

```

    Ave() {} //Constructor
    virtual ~Ave() {} //Destructor virtual
    virtual void Volar() const {cout<<"El Ave Vuela 2 metros...";}
    //Declaramos un método virtual que puede ser declarado nuevamente
//en otra clase
};

class Pegaso:public Caballo,public Ave
/*Declaramos una clase que hereda características de nuestras dos primeras
clases*/
{
public:    //Miembros públicos de la clase
    Pegaso() {} //Constructor
    virtual ~Pegaso() {} //Destructor virtual
    virtual void Galopar() const {cout<<"El pegaso Galopa 2 pasos";}
    //Declaramos un método virtual que puede ser declarado nuevamente
//en otra clase
    virtual void Volar() const {cout<<"El pegaso Vuela 2 metros...";}
    //Declaramos un método virtual que puede ser declarado nuevamente
//en otra clase
};

int main()
{
    clrscr();
    int choice;
    do //Ejecutamos un grupo de acciones
    {
        cout<<"(1)Caballo\n"<<"(2)Ave\n"<<"(3)Pegaso\n";
        cin>>choice; //Leemos el valor ingresado por teclado
    }while(choice<1 || choice>5);
    //Mientras que la respuesta esté entre 1 y 4
    if(choice == 1) //Si la respuesta es 1

```



```

{
    Caballo * Horse = new Caballo; //Declaramos un puntero a un objeto
    //de nuestra clase caballo
    Horse->Galopar(); //Llamamos al método virtual Galopar()
    delete Horse; //Eliminamos el objeto de la memoria
}
else if(choice == 2) //Si la respuesta es 2
{
    Ave * Bird = new Ave; //Declaramos un puntero a un objeto //de
    //nuestra clase caballo
    Bird->Volar(); //Llamamos al método virtual Volar()
    delete Bird; //Eliminamos el objeto de la memoria
}
else if(choice == 3) //Si la respuesta es 3
{
    Pegaso * Pegasus = new Pegaso; //Declaramos un puntero a un
    //objeto de nuestra clase caballo
    Pegasus->Galopar(); //Llamamos al método virtual Galopar()
    cout<<"\n";
    Pegasus->Volar(); //Llamamos al método virtual Volar()
    delete Pegasus; //Eliminamos el objeto de la memoria
}
getch(); //Copiamos la pantalla
}

```

### Conclusión

Como podemos ver, Pegaso hereda tanto de Caballo como de Ave, y por lo tanto puede galopar y volar. Una prueba interesante para comprobar esto, puede ser comentar los dos métodos de Pegaso, con lo cual se ve de manera más clara que hereda ambos métodos.

También, observamos la utilización de los métodos virtuales, "Virtual", que nos permiten declarar el mismo método para varias clases y luego, al hacerlas heredar sus atributos a una nueva clase, esta, sabe a que método llamar dependiendo de la clase que la está llamando. Es por eso, que cuando creamos nuestro objeto Pegaso, este puede volar y cabalgar, pero no con los métodos del caballo y el ave, sino con los suyos propios, a pesar de que los métodos de ambos tengan el mismo nombre.

## Capítulo 13 - Clases especiales

### Miembros de Datos Estáticos

Los miembros de datos estáticos, son aquellos, que permanecen a lo largo de todo el proceso de las instancias de la clase, están en contacto tanto con los datos globales como con los objetos de la clases. En realidad pertenecen a las clases y existen una por clase.

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()

class Gato                //Declaramos nuestra clase
{
public:                   //Datos públicos de la clase
    Gato() {CuantosGatos++;} //Constructor
    virtual ~Gato() {CuantosGatos--;} //Destructor Virtual
    static int CuantosGatos; //Variable estática
};

int Gato::CuantosGatos = 0; //Inicializamos nuestra variable estática

int main()
{
    const int MaxGatos = 5; //Variable constante
    int i;
    Gato * Gatitos[MaxGatos]; //Creamos un puntero un arreglo o matriz
    //de nuestra clase
    for(i=0;i<MaxGatos;i++) //Hacemos un for, para la máxima cantidad
    //de gatos que tenemos
        Gatitos[i] = new Gato; //Creamos un nuevo objeto por cada
    //una de las vueltas del for
```

```
        for(i=0;i<MaxGatos;i++)//Hacemos un for, para la máxima cantidad //de  
gatos que tenemos  
    {  
        cout<<"\nQuedan: ";  
        cout<<Gato::CuantosGatos; //Imprimimos la variable estática  
        cout<<" diciendo Miauu!!!";  
        delete Gatitos[i];    //Borramos el arreglo o matriz creado  
        Gatitos[i] = 0;    //Por seguridad, llenamos con 0 el espacio en memoria  
    }  
    getch();  
}
```

## Capítulo 14 - Cadenas de Caracteres

### Miembros de Cin

#### get() sin parámetros

Este comando retorna el valor del caracter encontrado hasta que encuentre el final del archivo. No se pueden concatenar varias instancias de caracteres en el get() sin parámetros.

```
#include <iostream.h>           //Librería para cout y cin
#include <conio.h>               //Librería para clrscr() y getch()

int main()
{
    char ch;    //Declaramos una variable tipo Char, que guarda un caracter
    ch = "Hola";
    while((ch=cin.get())!=EOF)    //EOF, End of File.
    //cin.get() va a tomar letra por letra de la palabra ingresada
    {
        cout<<"ch: "<<ch;
    }
    getch();    //Copiamos la pantalla
}
```

#### get() con parámetros

Cuando se utiliza el get() con parámetros, si es posible concatenar las instancias de los caracteres.

```
#include <iostream.h>           //Librería para cout y cin
#include <conio.h>               //Librería para clrscr() y getch()

int main()
{
```

```

char a,b,c;    //Declaramos 3 variables de tipo char

cout<<"Ingresa tres letras: ";
cin.get(a).get(b).get(c);    //Tomamos cada letra ingresada por teclado
cout<<"a: \n"<<a<<" b: \n"<<b<<" c: "<<c;    //Imprimimos los valores
getch();    //Copiamos la pantalla
}

```

### **peek() ,putback(), ignore()**

peek() se utiliza para saber cual es el caracter que viene a continuación y putback(), reemplaza el caracter siguiente por algún otro caracterer que definamos. ignore(), ignora un caracter determinado, tomandolo como inicio e ignorando en adelante de acuerdo al número especificado.

```

#include <iostream.h>    //Librería para cout y cin
#include <conio.h>    //Librería para clrscr() y getch()

int main()
{
    char ch;    //Declaramos una variable tipo Char, que guarda un caracter
    cout<<"Ingresa una cadena de caracteres: ";
    while(cin.get(ch))    //Mientras haya algun caracter por obtener
    {
        if(ch=='a')    //Si encontramos la letra "a"
            cin.putback('$');    //La reemplazamos por "$"
        else    //Sino encontramos la letra "a"
            cout<<ch;    //Imprimimos la letra que corresponde
        while(cin.peek()=='e')    //Mientras leamos la letra "e"
            cin.ignore(1,'e');    //La dejamos fuera
    }
    getch();
}

```

## Miembros de Cout

### width() y fill()

El objetivo de width(), es dejar espacio, de acuerdo al valor que le entreguemos, algo así como varios "\t". Funciona en relación con fill(). El objetivo de fill(), es rellenar espacios vacíos con algún carácter que determinemos.

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()

int main()
{
    cout<<"Ejemplo: ";
    cout.width(15);      //Establecemos la cantidad de espacios en blanco antes de
    //imprimir la cadena
    cout.fill('$');      //Llenamos los espacios vacíos con "$"
    cout<<"";
    getch();
}
```

## Flags

### setf()

El comando setf(), se utiliza para poner flags o características al texto que va a escribirse. Ya sea oct, hex, setfill(), etc.

```
#include <iostream.h>    //Librería para cout y cin
#include <conio.h>        //Librería para clrscr() y getch()
#include <iomanip.h>      //Librería para setf()

int main()
{
    const int num = 138;    //Declaramos una variable constante
    cout<<"El número es: "<<num; //Imprimimos el número
```

```
cout<<"El número es: "<<hex<<num; //Lo imprimimos con hexadecimal
cout<<"El número es: ";
cout.width(10); //Dejamos 10 espacios en blanco
cout.setf(ios::left); //Alineamos la respuesta hacia la izquierda
cout<<oct<<num; //Imprimimos el valor en octal

getch(); //Copiamos la pantalla
}
```

## Lista de Modificadores

### No necesitan iomanip.h

- **flush**: libera el buffer
- **endl**: agrega una nueva linea y libera el buffer
- **oct**: pone la base a octal
- **dec**: pone la base a decimal
- **hex**: pone la base a hexadecimal

## Lista de Modificadores

### Necesitan iomanip.h

- **setbase(base)**: pone la base(0 = decimal, 8 = octal, 10 = decimal, 16 = hex)
- **setw(width)**: pone el mínimo tamaño del width
- **setfill(ch)**: el caracter de relleno a utilizar con width
- **setprecision(p)**: para números con coma flotante
- **setiosflag(f)**: pone una o más flags del ios
- **resetiosflag(f)**: resetea una o más flags del ios

## Entrada y Salida de Archivos

### Abriendo archivos para Lectura y Escritura

Para poder manipular archivos, es necesario crear un objeto "ofstream" y luego asociarlo con algún archivo existente. Para ello, debes incluir el archivo .



```
ofstream fout("prueba.cpp") //Para escritura
```

```
ifstream fin("prueba.cpp") //Para Lectura
```

### Ejemplo

```
#include <conio.h>    //Librería para clrscr() y getch()
#include <stdio.h>    //Librería para gets();
#include <fstream.h> //Librería para fout y fin

void ImprimirN();
void Ingresar();

char nombre[20];
float sueldo,salario,sueldoneto;
int dias,i,j;

int main()
{
    clrscr();        //Limpiamos la pantalla
    for(i = 0;i < 2;i++) //Llamamos 2 veces a nuestro método
        Ingresar(); //Ingresamos los datos
    for(j = 0;j < 2;j++) //Llamamos 2 veces a nuestro método
        ImprimirN(); //Imprimimos los datos
    getch(); //Copiamos la pantalla
}

void Ingresar()      //Definimos nuestro método
{
    cout<<"\nNombre: ";
    gets(nombre);    //Leemos todos los caracteres, incluyendo espacios
    cout<<"\nSueldo: ";
```

```

cin>>sueldo;
cout<<"\nSalario: ";
cin>>salario;
cout<<"\nDías que trabajo: ";
cin>>dias;
cout<<"\nSueldo Neto: ";
cin>>sueldoneto;

// "c:\\trabajad.txt" es la ubicación y nombre del archivo
// como se crea cuando se ejecuta el programa por primera
// vez, se le puede dar una ruta distinta y el nombre debe
// ser de máximo, ocho caracteres.
ofstream fout("c:\\trabajad.txt", ios::app); // Abre el archivo para escritura
// El ios::app, hace que cada nuevo dato se agregue al final del archivo
fout<<"\nRelacion de Trabajadores "; // Escribe en el archivo
fout<<"\nNombre: "<<nombre<<"\nSueldo: "<<sueldo<<"\nSalario: ";
fout<<salario<<"\nDías que trabajo: "<<dias<<"\nSueldo Neto: ";
fout<<sueldoneto<<"\n-----";
fout.close(); // Cierra el archivo
clrscr(); // Vuelve a limpiar la pantalla
}

void ImprimirN()
{
    int num;
    ifstream fin("c:\\trabajad.txt"); // Abre el archivo para lectura
    char ch;
    while(fin.get(ch)) // Mientras hayan datos, los leemos
        cout<<ch; // y los imprimimos
    ofstream fix("c:\\trabajad.txt"); // Escribimos los datos en el archivo
    fix<<ch; // para evitar su repetición
    fin.close(); // Cierra el archivo
}

```

## Comportamiento de Ofstream

Pueden utilizarse ios para modificar el comportamiento del ofstream.

- **ios::app** Agrega los datos al final del archivo
- **ios::ate** Pone el cursor al final del archivo pero te permite escribir donde sea
- **ios::trunc** Trunca el archivo.
- **ios::nocreate** Si el archivo no existe, ocurre un error
- **ios::noreplace** Si el archivo ya existe, ocurre un error

## Capítulo 15 - Gráficos BGI

Los gráficos BGI, son gráficos para el modo DOS, que se pueden ejecutar gracias a la librería BGI (Borland Graphics Interface).

Utilizar esta librería nativa del Borland C++, nos evita tener que estar utilizando interrupciones al DOS escritas en ASM, que en mi experiencia personal.....Jamás me funcionaron -;)

Además, basta con que incluyamos las librerías en la misma carpeta que nuestra aplicación y quien lo reciba, podrá ver los gráficos sin ningún problema.

Para poder utilizar los gráficos BGI en nuestras aplicaciones, debemos incluir la librería [graphics.h](#).

Veamos un pequeño ejemplo:

```
#include <conio.h>      //Librería para getch
#include <graphics.h>   //Librería para gráficos BGI

void main()
{
    int gdriver = VGA, gmode = VGAHI; //Parametros de inicialización
    initgraph (&gdriver, &gmode, "c:\\borlan~1\\bgi");
    //El tercer parámetro debe ser la ruta exacta en formato DOS de la librería BGI
    setcolor(RED);      //Establecemos el color de la figura que vamos a crear
    circle(300,250,50); //Creamos un circulo
    setcolor(BLUE);     //Establecemos el color de la figura que vamos a crear
    circle(300,250,20); //Creamos un circulo
    getch();            //Copiamos la pantalla
    closegraph();       //Cerramos el modo gráfico
}
```

Analizando un poco este código, nos encontramos con algunos comandos interesantes, que vale la pena explicar. Por ejemplo, el comando `setcolor()`,

es el que le va a dar un color a nuestra imagen o figura. Recuerden que el nombre del color, debe ser escrito siempre en mayúsculas para que funciones sin errores. El otro comando, es `circle()`, que como nos hemos dado cuenta al momento de ejecutar la aplicación, crea un círculo. Sus tres parámetros representan (Coordena X, Coordenada Y, Radio). Es decir, especificamos su ubicación en el plano, y además cual es el Radio que va a tener, que como sabemos, a mayor Radio, mayor tamaño del círculo.

Por último, no debemos olvidarnos nunca del comando `closegraph()`, pues es el quien cierra la sesión en modo gráfico y nos permite continuar en modo DOS de así requerirlo.

Veamos ahora una pequeña lista de las figuras o imágenes que podemos crear con la librería BGI:

```
cleardevice()      //Limpia la pantalla, reemplaza a clrscr() en BGI
setbkcolor()       //Establece el color de fondo de la pantalla
setcolor(COLOR)    //Establece el color de una imagen o figura
bar(int,int,int,int) //Dibuja una barra, → Izquierda,Arriba,Derecha,Abajo
bar3D              //Dibuja una barra en formato 3D, con profundidad.
                  //→ Izquierda,Arriba,Derecha,Abajo,profundidad,tope
circle(int,int,int) //Dibuja un círculo, →X, Y, Radio
line(int,int,int,int) //Dibuja una línea, →X1, Y1, X2, Y2
rectangle(int,int,int,int) //Dibuja un rectángulo →
Izquierda,Arriba,Derecha,Abajo
ellipse(int,int,int,int,int,int) //Dibuja una elipse → X, Y, Ángulo Inicial,
Ángulo Final, Radio de X, Radio de Y.
```

Ahora veamos cuales son los colores que tenemos disponibles:

|               |                   |
|---------------|-------------------|
| BLACK → Negro | RED → Rojo        |
| BLUE → Azul   | MAGENTA → Púrpura |
| GREEN → Verde | BROWN → Marrón    |

|   |                          |
|---|--------------------------|
| CYAN → Matiz entre azul y verde                 | YELLOW → Amarillo        |
| LIGHTGRAY → Gris Claro                          | DARKGRAY → Gris Oscuro   |
| LIGHTBLUE → Azul Claro                          | LIGHTGREEN → Verde Claro |
| LIGHTCYAN → Matiz entre azul y verde, claro.... |                          |
| LIGHTRED → Rojo Claro                           | WHITE → Blanco           |
| LIGHTMAGENTA → Púrpura Claro                    |                          |

Ahora, supongamos que queremos escribir el nombre de nuestro juego o programa...pero utilizando los vistosos gráficos BGI. Aquí tenemos un pequeño ejemplo:

```
#include <conio.h>      //Librería para getch
#include <graphics.h>    //Librería para gráficos BGI

void main()
{
    int gdriver = VGA, gmode = VGAHI; //Parametros de inicialización
    initgraph (&gdriver, &gmode, "c:\\borlan~1\\bgi");
    //El tercer parámetro debe ser la ruta exacta en formato DOS de la librería BGI
    settextstyle(GOTHIC_FONT,HORIZ_DIR,24);
    //Definimos el tipo de letra, la alineación y el tamaño
    outtextxy(20,20,"Hola Mundo");
    //Especificamos las coordenadas X, Y e imprimimos el mensaje
    getch();    //Copiamos la pantalla
    closegraph(); //Cerramos el modo gráfico
}
```

A continuación, una pequeña tabla con los tipos de letra o fuentes que tenemos disponibles:

|              |              |
|--------------|--------------|
| TRIPLEX_FONT | SMALL_FONT   |
| GOTHIC_FONT  | DEFAULT_FONT |

## SANS\_SERIF\_FONT

Ahora, mostraremos las alineaciones, es decir, si el texto se va a mostrar de manera horizontal (Izquierda a Derecha) o vertical (Arriba a Abajo):

HORIZ\_DIR                      VERT\_DIR

Siempre es importante tomar en cuenta la alineación que le damos a los textos:

LEFT\_TEXT → Alineado hacia la izquierda

CENTER\_TEXT → Centrado

RIGHT\_TEXT → Alineado hacia la derecha

Ahora para finalizar, veamos un ejemplo un poco más complejo, de gráficos y coordenadas, el llamado "OJO DE MOSCA":

```
#include <conio.h>      //Librería para getch()
#include <graphics.h>    //Librería para gráficos BGI

void draw_circle();      //Métodos para dibujar
void draw_rectangles();  //los círculos, triángulos
void draw_triangles();   //rectángulos y líneas.
void draw_lines();

void main()
{
    int gdriver = VGA,gmode = VGAHI; //Establecemos los drivers.
    initgraph(&gdriver,&gmode,"c:\\archiv~1\\progra~2\\borlan~1\\bgi");
        //Abrimos la interrupción para el grafico.
        //Debemos establecer la ruta en la cual se encuentra nuestra
        //carpeta BGI.
    setcolor(WHITE); //Establecemos el color de la figura geométrica.
    draw_circle();    //Llamamos al función que dibuja los círculos.
    setcolor(BLUE);
    draw_rectangles();
```

```

setcolor(GREEN);
draw_triangles();
setcolor(RED);
draw_lines();
getch();    //Copiamos la pantalla.
closegraph(); //Cerramos la interrupción para el grafico.
}

void draw_circle()
{
circle(320,250,225); //Dibuja un circulo (x,y,radio).
}

void draw_rectangles()
{
rectangle(150,100,490,400);
line(100,250,320,30); //1er cuadrante (x1,y1,x2,y2).
line(320,470,540,250); //4to cuadrante
line(320,30,540,250); //2do cuadrante
line(100,250,320,470); //3er cuadrante
}

void draw_triangles()
{
//Primer triángulo
line(190,430,450,430);
line(190,430,320,30);
line(450,430,320,30);

//Segundo triángulo
line(190,70,450,70);
line(190,70,320,470);
line(450,70,320,470);

```



```
//Tercer triángulo
line(130,130,130,370);
line(130,130,540,250);
line(130,370,540,250);

//Cuarto triángulo
line(510,130,510,370);
line(510,130,100,250);
line(510,370,100,250);
}

void draw_lines()
{
//Cruz
line(100,250,540,250);
line(320,30,320,470);

//Diagonales
line(150,100,490,400);
line(490,100,150,400);
}
```

Ahora, como recomendación final, les digo que lean y revisen la librería [graphics.h](http://graphics.h), ya que tienes muchísimas funciones que no he podido incluir por razones obvias.....Entendiendo estos comandos básicos y revisando los que quedan en la librería, van a poder manejar el BGI a la perfección.

## Comentarios Finales

Espero que este tutorial les sea útil como una pequeña introducción al lenguaje de programación C++. Obviamente, queda mucho camino por recorrer, uno no puede abarcar todas las posibilidades que posee un lenguaje, en un tutorial, ni en diez libros. Lo importante ahora, es que busque otros tutoriales con los que puedan complementarse y bajen códigos y lo revisen hasta comprenderlos a la perfección. Demás está decir que cuentan conmigo para cualquier duda que tengan.

Pueden acceder a mi página web:

SinglePath games design (<http://www.iespana.es/singlepath>)

Escribirme un mail:

[singlepathpe@yahoo.es](mailto:singlepathpe@yahoo.es)

O ingresar a:

Xpertia (<http://www.xpertia.com>)

Nombre de experto: **blag**

# Bibliografía y Agradecimientos

## Bibliografía:

- "Teach Yourself C++ in 21 Days" de Sams Publishing.
- <http://www.planet-source-code.com>
- <http://www.lawebdelprogramador.com>

## Agradecimientos:

- Jesse Liberty de Liberty Associates, autor de "Teach Yourself C++ in 21 Days".
- [www.Planet-Source-Code.com](http://www.Planet-Source-Code.com)
- [www.lawebdelprogramador.com](http://www.lawebdelprogramador.com)
- [www.xpertia.com](http://www.xpertia.com)
- [www.informit.com](http://www.informit.com)
- [www.teleportmedia.com](http://www.teleportmedia.com)
- Mi novia Milly, por todo su apoyo.

## 18.- El Fin

