

Maestría en Matemática Herramientas Computacionales

Jose Alvarenga

UNAH

PACIII2023

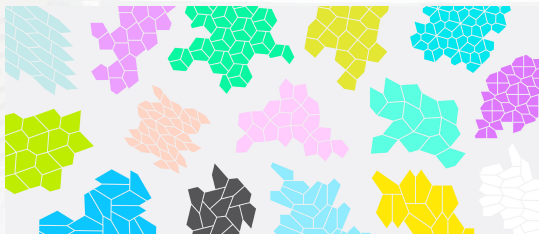
Tabla de Contenidos

- 1 Motivación
- 2 Programación Estructurada y Orientada a Objetos (C++)

Introducción I

La computación matemática en la investigación:

- 1 Resolver problemas de la ciencia mediante la simulación por ordenador. Un ejemplo de interesante es el de *Problemas de Mosaicos con copias de polígonos convexos*. En 2017 Rao (Escuela Normal Superior de Lyon) resolvió este problema.



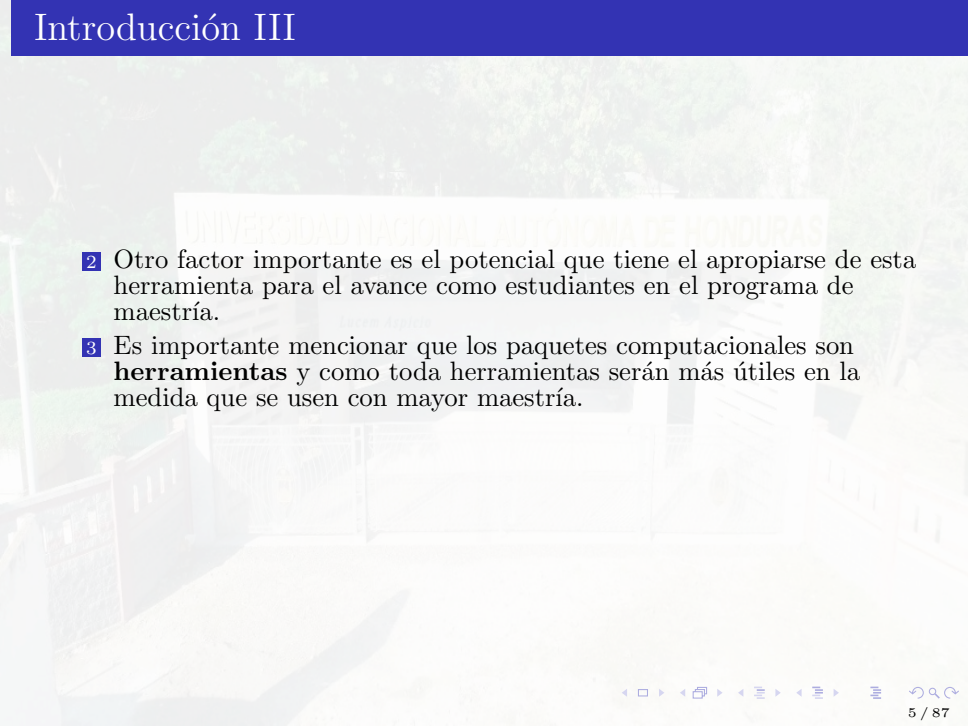
Otras aplicaciones interesantes pueden ser consideradas para estudiar el comportamiento de dinámica de sistemas. Un ejemplo interesante es el estudio del tráfico; en el siguiente enlace podemos ver la dinámica vehicular en una sección de de carretera de varios carriles [*Simulacion Trafico*](#).

Introducción II

- 2 Paquetes estadísticos. Uno de los métodos más famosos usados en paqueterías estadística son los denominados como "Montecarlo". Su origen tiene lugar en el desarrollo de la bomba atómica en los Álamos. Una pequeña descripción del origen es compartida por Nicholas Metropolis:
- Desarrollo de la computadora ENIAC.
 - Simulación de una reacción termonuclear.
 - Ulam tuvo la idea de aplicar las técnicas de muestreo en conjunto con el nuevo descubrimiento.
- 3 Cálculo simbólico. Realización de cálculos y operaciones simbólicas. Un ejemplo, ¿Cuál es el orden del grupo de Galois de $p(x) = x^5 - x + 1$? El resultado es 120; Maple te permite encontrar este valor con el comando *GroupOrder*.
- 4 Áreas: Criptografía, Optimización, Ecuaciones diferenciales, Topología, etc.

Observaciones

- 1 Parte de la importancia del curso recae en que el componente computacional es prácticamente inseparable del proceso de formación en la maestría.

- 
- 2 Otro factor importante es el potencial que tiene el apropiarse de esta herramienta para el avance como estudiantes en el programa de maestría.
 - 3 Es importante mencionar que los paquetes computacionales son **herramientas** y como toda herramientas serán más útiles en la medida que se usen con mayor maestría.

Instalación de Visual Studio y Compilador

Para instalar C++, es necesario contar con un IDE (entorno de desarrollo integrado). En resumen es necesario seguir los siguientes pasos:

- 1 Descargar la versión actual de visual studio code. Puedes encontrarla en la siguiente página [VisualStudioCode](#).
- 2 Instala el compilador de C++. Si tienes un sistema operativo windows, entonces debes instalar el compilador de C++. Puedes encontrarlo en la siguiente página [Msys2](#).
- 3 Si tienes otro sistema operativo, normalmente este ya viene acompañado por un compilador de C++.

Para más detalles consulta el siguiente enlace, de donde se hizo el resumen anterior [VideoTutorial](#).

Uso de GitHub

Para crear un repositorio y actualizarlo en GitHub siga los siguientes pasos:

- 1 Crear una carpeta que se vaya a compartir con la plataforma de GitHub.
- 2 Crear un archivo README.md; dentro de este archivo puede especificar la finalidad del repositorio que se esta creando.
- 3 Dentro de Visual Studio abra la carpeta creada.
- 4 En la opción del *Control de código fuente* se habilitará la opción *Publicar en GitHub*. Sigue los pasos que allí se mencionan.
- 5 Para hacer una actualización de cualquier archivo, ejecute las siguientes instrucciones en una terminal:

```
git status
git add .
git commit -m "Mensaje personalizado"
git push origin main
```


Una motivación del uso del lenguaje C++ se da cuando se desea construir aplicaciones grandes y complejas. Otro aspecto notable es que conceptualmente la dinámica del lenguaje es común a otros lenguajes. Los elementos comunes a la generación de un programa en C++ son los siguientes:

- 1 Código fuente: El texto que representa el programa escrito en un lenguaje de alto nivel.
- 2 Archivo fuente: El archivo que contiene al código fuente.
- 3 Editor: Programa informático para editar el código fuente. Block de notas por ejemplo.
- 4 Sufijo: Es normal que el sufijo del archivo fuente sea cpp.
- 5 Archivo ejecutable: Es el archivo que se genera a partir del código fuente, una vez que este ha sido traducido al lenguaje de máquina. Este archivo normalmente se generará con el sufijo exe con el mismo nombre del archivo fuente.

- 6 Directiva de preprocesador: Son ciertas líneas de código que inicialmente se colocan al inicio del código fuente y tienen el prefijo `#`. Normalmente se usa la directiva `"include"`. A esta directiva se le agrega el nombre de lo que se denomina un archivo de cabecera, el cual contiene un conjunto de rutinas útiles para el desarrollo de nuestro programa.
- 7 Espacio de nombres: Es un contenedor o entorno abstracto creada para contener un conjunto de bloques. Un espacio de nombres comunes es `"std"`, el cual representa a los elementos de la infraestructura estándar de C++. Tiene una función de organización.
- 8 Bloque principal: Esta parte del programa es donde ubicaremos el código que resolverá nuestro problema.

Tipo de datos primitivos y operadores I

En el aspecto numérico existen dos tipos de representaciones: Los enteros y los de coma flotante (floating point). A continuación se detallan algunas características de este tipo de datos:

- Tipo: Es una característica del número, normalmente mide la representación y el tipo de rango.
- Apóstrofe: Una forma natural de escribir enteros en C++ para un mejor lectura, por ejemplo, en lugar de 1111 se puede escribir 1'111.
- Si se precede un número con cero, se entenderá que este tiene una representación octal.
- Si se precede un número con cero y x (0X o 0x), se entenderá que este tiene una representación hexadecimal.
- Si se precede un número con cero y b (0B o 0b), se entenderá que este tiene una representación binaria.
- Notación científica: xEy , donde x es una expresión decimal y y es un entero; en este caso xEy representa $x \times 10^y$.

En C++ también podemos encontrar datos y operadores relacionados con la lógica de proposiciones:

- Los datos de tipo lógico representan a dos tipos de valores, verdadero (true) o falso (false). En C++ a estos tipos se les declara como *bool*.

Tipo de datos primitivos y operadores II

- Conjunción $\&$. Disyunción $\|$. Negación $!$. Comparación $==$.
Operadores de orden $>$, $<$, $>=$, $<=$. El operador de comparación para la diferencia $!=$.

En la siguiente tabla se muestra el funcionamiento de algunos operadores lógicos.

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$\neg p$
V	V	V	V	V	F
V	F	F	V	F	F
F	V	F	V	V	V
F	F	F	F	V	V

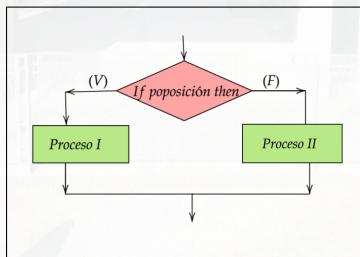
Figura: En la tabla se puede apreciar el comportamiento de los operadores lógicos de conjunción, disyunción, implicación y negación respectivamente.

Programación estructurada I

Paradigma de la programación estructurada

Según este paradigma, se establece que en cualquier algoritmo se pueden identificar los siguientes bloques de repetición:

- Bloques secuenciales.
 - Bloques de selección (if, if-else, switch).
 - Bloques de repetición (while, for, do-while, continue, break).
- El bloque estándar de selección es el denominado *bloque If-Else*.



- Es posible anidar los bloques If-Else de las siguientes formas:

Programación estructurada II

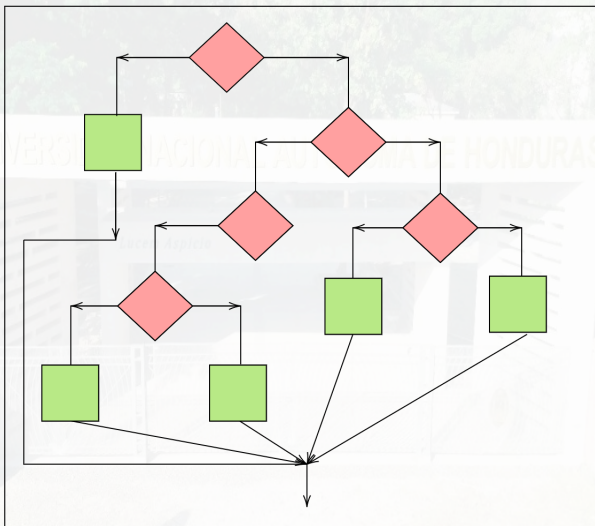


Figura: En el diagrama se observa el estilo anidado, del bloque If-Else.

Programación estructurada III

- El bloque de repetición más estándar es el bloque while. El funcionamiento del bloque while se muestra a continuación:

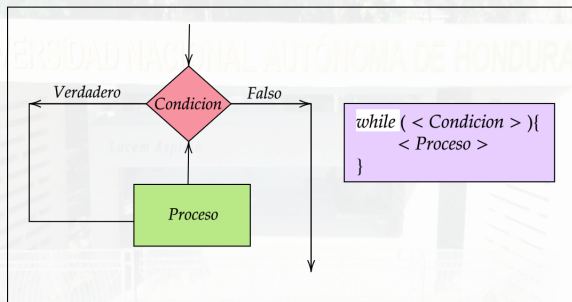
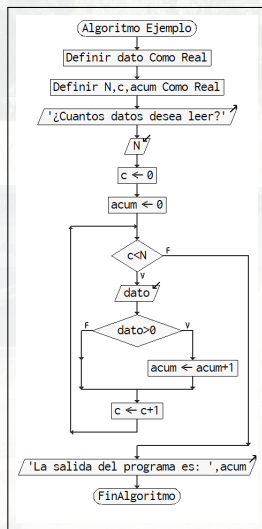


Figura: En el diagrama se observa el bloque mientras.

Ejercicio: Escriba el código en C++ correspondiente al siguiente diagrama de flujo.

Programación estructurada IV



- El bloque de repetición *for*, es una estructura de control de repetición que se manipula a través de una variable contadora.

Programación estructurada V

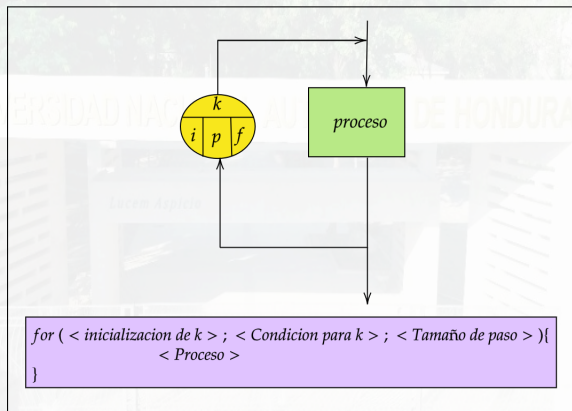
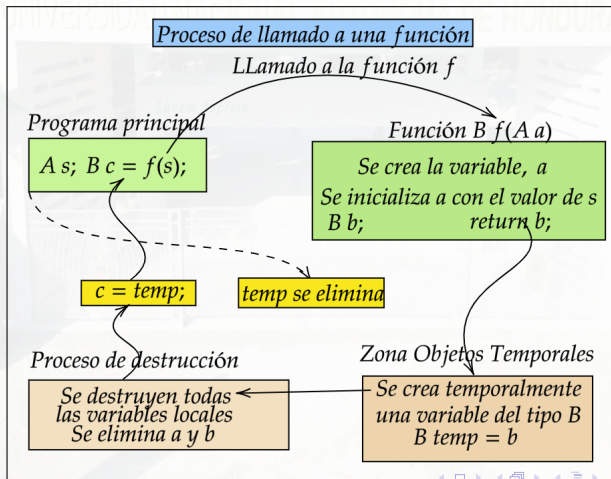


Figura: En la figura se observa el código y el diagrama de flujo de un bloque for.

Funciones I

Los elementos más importantes alrededor de una función son:

- 1 Prototipo.
- 2 Implementación.
- 3 Invocación.



Dentro del prototipo de las funciones existen dos partes notables: *el retorno de la función y los parámetros de la función*. Como una característica importante, se sabe que tanto el retorno como los parámetros se pueden mandar o recibir por referencia. El siguiente ejemplo dejará claro cuál es la diferencia alrededor de estos aspectos:

- Al invocar a $f4$ se crea una copia del valor de x . Además internamente se crea una variable "sin nombre" para dar el retorno de la función.
- Al invocar a $f3$ se hace un paso por referencia. Esto implica que no se creará una copia del argumento, en su lugar se seguirá trabajando con la variable que se pase en los parámetros.
- Al invocar a $f2$ habrá un problema; al hacer un retorno por referencia en el momento que esta función termine, la referencia se perderá y por lo tanto no existirá manera de retornar ese valor.
- En el caso de $f1$ no existirá el mismo problema dado que la referencia que se envía, continúa existiendo dentro del programa principal.

Código 1: Ejemplo de paso y retorno por referencia

```
1 #include <iostream>
2 using namespace std;
3 int & f1(int &x){
4     x+=1;return x;}
5 int & f2(int x){
6     x+=1;return x;}
7 int f3(int &x){
8     x+=1;return x;}
9 int f4(int x){
10    x+=1;return x;}
11 int main(){
12     int x=10;int y;
13     y=f4(x);
14     y=f3(x);
15     y=f2(x);
16     y=f1(x);
17     return 0;}
```

Código 2: Ejemplo de una función creada por el programador

```
1 #include <iostream>
2 using namespace std;
3 int Div(int a, int b){
4     return a/b;
5 }
6 int main(){
7     int x;
8     x=10;
9     cout<<" Invocacion : _"<<Div(x,20)<<endl;
10    return 0;}
```

Código 3: Cambiando el orden de la codificación

```
1 #include <iostream>
2 using namespace std;
3 //Definicion de prototipo
4 int Div(int , int);
5 int main(){
6     int x;
7     x=10;
8     cout<<" Invocacion : _"<<Div(x,20)<<endl;
9     return 0;}
10 int Div(int x, int y){
11     return x/y;
12 }
```

Código 4: Ejemplo de sobrecarga de funciones

```
1 #include <iostream>
2 using namespace std;
3 int Div(int, int);
4 double Div(double, int);
5 double Div(double, float);
6 int main(){
7     int x;
8     x=10;
9     cout<<" Invocacion : _"<<Div(x,20)<<endl;
10    return 0;}
11 double Div(double x, int y){
12     return x/y;}
13 int Div(int x, int y){
14     return x/y;}
15 double Div(double x, float y){
16     return x/y;}
```

La salida del código anterior es cero.

Código 5: Alcance de una variable

```
1 #include <iostream>
2 using namespace std;
3 //Variable Global
4 int GlobalX=10;
5 //Funcion sin retorno (void)
6 void f(int x);
7 int main(){
8     bool cen=true;
9     while(cen){
10         int x=10;
11         f(x); //Invocaion a f
12         cout<<GlobalX<<"\n";
13         //Variable local al
14         //bloque mientras.
15         cen=false; }
16     return 0; }
17 void f(int x){
18     int z=10;
19     //Variable local a f
20     cout<<" Argumento: _"<<x<<endl;
21     cout<<" Variable _Global: _";
22     cout<<GlobalX<<endl; }
```

La salida del codigo anterior sería:

```
Argumento: 10
Variable Global: 10
10
```


Código 6: Ejemplo paso por copia

```
1 #include <iostream>
2 using namespace std;
3 void f(int x);
4 int main(){
5     int x=20;
6     f(x);
7     cout<<"x:_"<<x<<endl;
8     return 0;}
9 void f(int x){
10    x=10;}
```

Código 7: Ejemplo referencia

```
1 #include <iostream>
2 using namespace std;
3 void f(int &x);
4 int main(){
5     int x=20;
6     f(x);
7     cout<<"x:_"<<x<<endl;
8     return 0;}
9 void f(int &x){
10    x=10;}
```

Funciones VIII

La recursividad es el proceso mediante el que una función se llama a sí misma de forma repetitiva. Considere la siguiente definición:

$$f(n) = n f(n - 1) \text{ para } n > 0$$

$$f(0) = 1$$

$$f(6) = 6 \times f(5)$$

$$= 6 \times \overbrace{5 \times f(4)}$$

$$= 6 \times 5 \times \overbrace{4 \times f(3)}$$

$$\vdots$$

$$= 6 \times 5 \times 4 \times 3 \times 2 \times \overbrace{1 \times f(0)}$$

$$= 6 \times 5 \times 4 \times 3 \times 2 \times 1 \times \overbrace{1} = 720$$

A continuación se muestra como crear la función factorial:

Código 8: Ejemplo de la función factorial

```
1 #include <iostream>
2 using namespace std;
3 int factorial(int n);
4 int main(){
5     cout<<factorial(6)<<endl;
6     return 0;}
7 int factorial(int n){
8     if(n==0)
9         return 1;
10    return n*factorial(n-1);}
```

Arreglos y cadenas I

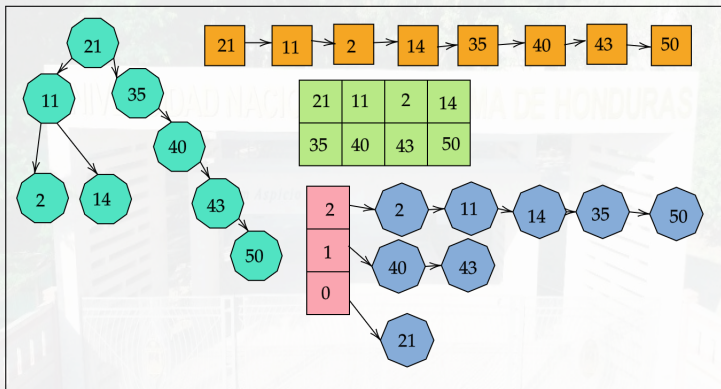
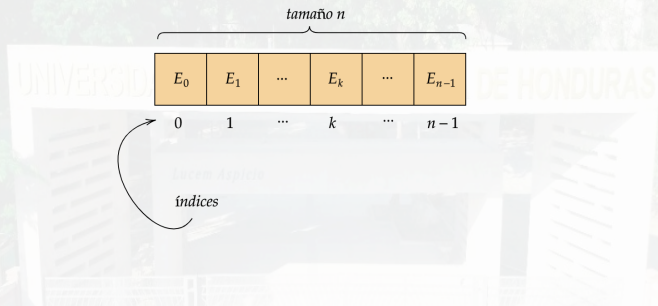


Figura: En la imagen se puede observar las diferentes formas en las que se puede empaquetar la información.

Arreglos y cadenas II

Figura: Estructura de los arreglos en C++



Declaración

<Tipo de datos><Nombre del arreglo> [<Tamaño del arreglo>];

■ `int A[100];`

■ `char B[30];`

■ `double C[10];`

■ `bool D[30];`

Inicialización 1

- `int A[]={1,2,3,4};`
`A=[1,2,3,4]`
- `int A[4]={1,2,3,4};`
`A=[1,2,3,4]`
- `int A[4]={1,2};`
- `A=[1,2,0,0]`
- `int A[4]={}`
`A=[0,0,0,0];`
- `int A[]={}`
Esto es un error.

Observación: Si no inicializas el arreglo en la declaración, el contenido del arreglo será incierto. Contendrá valores "basura" de la memoria del computador.

Lectura 1

```
int A[10];  
for(int i=0;i<10;i++)  
cin>>A[i];
```

Lectura 2

```
char A[10];  
for(int i=0;i<10;i++)  
A[i]=(char) i+50;
```

Lectura 3

```
int A[10];  
for(int i=0;i<10;i++)  
A[i]=i*i+2*i;
```


Arreglos unidimensionales en C++

Problema 1

Escriba un algoritmo que determine la MODA de un conjunto de 100 datos aleatorios, sabiendo que los datos están comprendidos en el rango de 0 a 50.

Solución

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main(){
    int Datos[100]={};
    int Indices[51]={};
    for(int i=0;i<100;i++)
        Datos[i]=rand()%51;
    for(int i=0;i<100;i++)
        Indices[Datos[i]]+=1;
```

Arreglos unidimensionales en C++

Continuación solución 1

```
int Mayor=Indices[0];
int valor=0;
for(int i=0;i<51;i++){
    if(Mayor<Indices[i]){
        Mayor=Indices[i];
        valor=i;
    }
}
for(int i=0;i<100;i++){
    cout<<Datos[i]<<" ";
}
cout<<endl;
cout<<"La MODA del conjunto de datos es: "<<valor<<endl;
cout<<"Numero de repeticiones: "<<Mayor<<endl;
return 0;}
```

Funciones con arreglos

Prototipo, arreglos en los parámetros.

- Función para imprimir un arreglo: `void Imprimir(int A[],int n).`
- Función para determinar si un valor se encuentra dentro de un arreglo: `bool Buscar(char, char [], int);`
- Función para determinar el promedio de un conjunto de valores: `float Promedio(float X[],int n);`

UNIVERSIDAD NACIONAL AUTÓNOMA DE HONDURAS

Búsqueda en arreglos

El objetivo de la **búsqueda** es encontrar un valor X en un arreglo dado A .

UNIVERSIDAD NACIONAL AUTÓNOMA DE HONDURAS

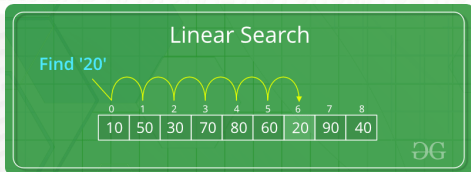
Búsqueda en arreglos

El objetivo de la **búsqueda** es encontrar un valor X en un arreglo dado A .

Observación: En algunas variaciones, se pide el índice donde está el valor encontrado. Y en el caso de que el valor no se encuentre se regresa negativo uno.

Algoritmo de búsqueda secuencial

Consiste en ir comparando desde el índice cero, el elemento buscado hasta encontrarlo.

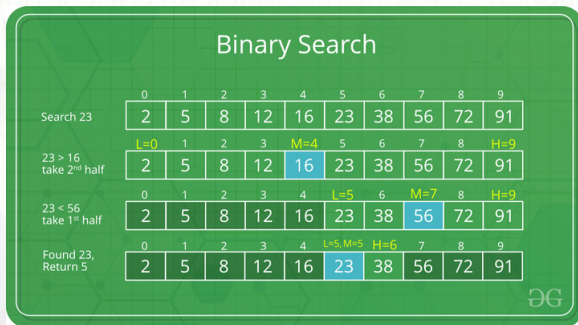


Fuente: GeeksforGeeks

```
int BusquedaSec(int x, int A[],int n){  
    for(int i=0;i<n;i++)  
        if(A[i]==x)  
            return i;  
    return -1;}
```

Algoritmo de búsqueda binaria

Este algoritmo de búsqueda presupone que nuestro arreglo ya se encuentra ordenado.

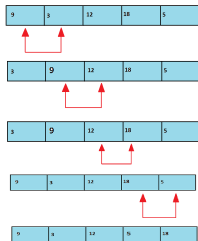


Fuente: GeeksforGeeks

Busqueda Binaria

```
int BusquedaBin(int x, int A[], int ini, int fin){  
    int med=(ini+fin)/2;  
    if(ini==fin && A[med]!=x)  
        return -1;  
    if(A[med]==x)  
        return med;  
    else{  
        if(A[med]>x)  
            return BusquedaBin(x,A,ini,med-1);  
        else  
            return BusquedaBin(x,A,med+1,fin);  
    }  
}
```

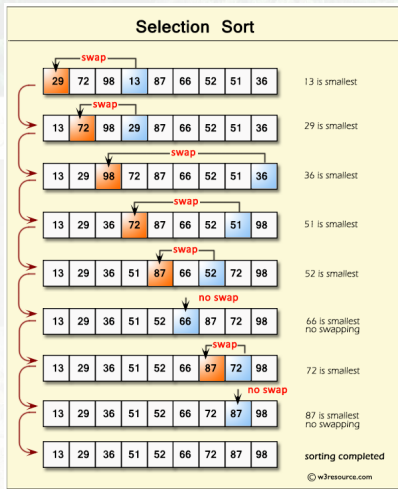
Intercambio (Swap)



Swap

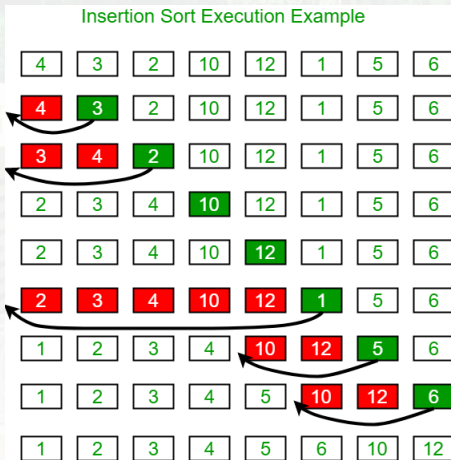
```
void Swap(int A[],int i, int j){  
    int aux=A[i];  
    A[i]=A[j];  
    A[j]=aux;}
```

Ordenamiento por selección



Fuente: w3resource

Ordenamiento por inserción



Fuente: GeeksforGeeks

Apuntador

Un puntero es una variable que almacena direcciones de memoria.

Características de los punteros

- Desreferenciación: Mecanismo por el que se recupera la información en la memoria, a través de un puntero. Analogía de las páginas en los libros.



Katerina Yushchenko (Lenguaje de programación Addrès 1955)

- Eficiencia: Permite el **paso de estructuras** a través de funciones de manera eficiente (paso por referencia)
- Memoria dinámica: Por medio de los puntero se pueden crear nuevas estructuras ajustadas a distintas necesidades de memoria.

Declaración e inicialización de un puntero

Puntero Nulo

En C++, cuando no deseas asignar una dirección a un puntero, puedes asignarle una constante conocida como "puntero nulo". Se puede definir un puntero vacío a través de la siguiente línea de código:

```
#define nulo 0
```

De manera predeterminada se puede inicializar un puntero a través de las siguientes constantes:

- NULL.
- nullptr.

Declaración e inicialización de un puntero

Puntero Nulo

En C++, cuando no deseas asignar una dirección a un puntero, puedes asignarle una constante conocida como "puntero nulo". Se puede definir un puntero vacío a través de la siguiente línea de código:

```
#define nulo 0
```

De manera predeterminada se puede inicializar un puntero a través de las siguientes constantes:

- NULL.
- nullptr.

La diferencia entre NULL y nullptr es que NULL es el entero cero y se puede interpretar como un entero; en el caso de nullptr no existe esta transformación natural.

Declaración e inicialización de un puntero

Puntero Nulo

En C++, cuando no deseas asignar una dirección a un puntero, puedes asignarle una constante conocida como "puntero nulo". Se puede definir un puntero vacío a través de la siguiente línea de código:

```
#define nulo 0
```

De manera predeterminada se puede inicializar un puntero a través de las siguientes constantes:

- NULL.
- nullptr.

La diferencia entre NULL y nullptr es que NULL es el entero cero y se puede interpretar como un entero; en el caso de nullptr no existe esta transformación natural.

A nivel de salida, con nullptr seguimos observando un cero si se le imprime.

Declaración e inicialización

<Tipo de dato al que apunta> *<Nombre del puntero>=<Dirección de memoria>

Declaración e inicialización de un puntero

A continuación se muestran algunas formas en las que se puede inicializar un arreglo:

Forma 1

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      double *x;
5      return 0;
6  }
```

Figura: En esta forma el puntero tendría información "basura".

Forma 2

```
1  #include <iostream>
2  using namespace std;
3  int main(){
4      bool *x=NULL;
5      char *z=nullptr;
6      return 0;
7  }
```

Figura: En esta forma al puntero se le asigna un puntero nulo. Si se imprimen ambos puntero se encontrará como salida un cero.

Declaración e inicialización de punteros

Operador de extracción de memoria

Por medio del operador & (ampersand) es posible extraer la dirección en memoria de cualquier variable en C++.

Forma 3

```
#include <iostream>
#include <iomanip>
using namespace std;
int main(){
    double s=12.2;
    double *x=&s;
    long double y=(intptr_t) x;
    cout<<setprecision(20)<<y<<endl;
    cout<<x<<endl;
    cout<<setprecision(20)<<setbase(16)<<y;
    return 0;
}
```

Figura: En esta forma al puntero se le asigna una dirección existente en la memoria. Además en este ejemplo particular la dirección de memoria es 0x9dd7fff838 y su valor entero es 677933742136.

Modificador const para punteros.

Operador de desreferenciación

El operador de desreferenciación es el caracter `*`(asterísco), en estos ejemplos observarás diferentes comportamientos:

Existen cuatro formas en las que un puntero puede tener acceso a la información a la que hacen referencia.

- `int *x=& z; //Acceso total a z.`
- `const int *x=& z; //Solo lectura de z.`
- `int *const x=& z; //Solo lectura de x;`
- `const int * const x=& z; // Solo lectura de x y z.`

```
int main(){
    double x=11.2;
    double y;
    double *p1=&x;
    const double *p2=&x;
    double * const p3=&y;
    cout<<"Valor de x: "<<*p1<<endl;
    *p1=x+12;
    cout<<"Valor de x despues de usar a p1: "<<*p2<<endl;
    /*p2=*p2+1; Error 1
    //p3=p2; Error 2
    return 0;
```

Desreferenciación

```
int main(){
    double x=11.2;
    double y;
    double *p1=&x;
    const double *p2=&x;
    double * const p3=&y;
    cout<<"Valor de x: "<<*p1<<endl;
    *p1=x+12;
    cout<<"Valor de x despues de usar a p1: "<<*p2<<endl;
    // *p2=*p2+1; Error 1
    // p3=p2; Error 2
    return 0;
}
```

```
Valor de x: 11.2
Valor de x despues de usar a p1: 23.2
```

Punteros, aritmética, arreglos

Considere la declaración:

```
int main(){  
    int A[5]={};  
    cout<<"Contenido de A: "<<A<<endl;  
    cout<<"Contenido de la dirección de A[0]: "<<&A[0];  
    return 0;  
}
```

Punteros, aritmética, arreglos

Considere la declaración:

```
int main(){  
    int A[5]={};  
    cout<<"Contenido de A: "<<A<<endl;  
    cout<<"Contenido de la dirección de A[0]: "<<&A[0];  
    return 0;  
}
```

```
Contenido de A: 0x71e1ff950  
Contenido de la direccion de A[0]: 0x71e1ff950  
Presione una tecla para continuar . . .
```

Este ejemplo revela dos cosas:

Punteros, aritmética, arreglos

Considere la declaración:

```
int main(){  
    int A[5]={};  
    cout<<"Contenido de A: "<<A<<endl;  
    cout<<"Contenido de la dirección de A[0]: "<<&A[0];  
    return 0;  
}
```

```
Contenido de A: 0x71e1ff950  
Contenido de la direccion de A[0]: 0x71e1ff950  
Presione una tecla para continuar . . .
```

Este ejemplo revela dos cosas:

- La variable de una arreglo es un puntero al tipo de dato que lo define.

Punteros, aritmética, arreglos

Considere la declaración:

```
int main(){  
    int A[5]={};  
    cout<<"Contenido de A: "<<A<<endl;  
    cout<<"Contenido de la dirección de A[0]: "<<&A[0];  
    return 0;  
}
```

```
Contenido de A: 0x71e1ff950  
Contenido de la direccion de A[0]: 0x71e1ff950  
Presione una tecla para continuar . . .
```

Este ejemplo revela dos cosas:

- La variable de una arreglo es un puntero al tipo de dato que lo define.
- La dirección que contiene es la del primer elemento.

Punteros, aritmética, arreglos

Considere la declaración:

```
int main(){  
    int A[5]={};  
    cout<<"Contenido de A: "<<A<<endl;  
    cout<<"Contenido de la dirección de A[0]: "<<&A[0];  
    return 0;  
}
```

```
Contenido de A: 0x71e1ff950  
Contenido de la direccion de A[0]: 0x71e1ff950  
Presione una tecla para continuar . . .
```

Este ejemplo revela dos cosas:

- La variable de un arreglo es un puntero al tipo de dato que lo define.
- La dirección que contiene es la del primer elemento.
- Este es un puntero constante, no puede cambiarse su dirección de memoria al primer elemento.

Puntero como argumento de funciones

```
1  #include <iostream>
2  using namespace std;
3  int Suma(int *A,int n);
4  int * Elementoi(int *A,int i);
5  int main(){
6      int X[6]={1,2,3};
7      cout<<Suma(X,6)<<endl;
8      int *p=Elementoi(X,4);
9      cout<<"El elemento con el indice 4 es: "<<*p<<endl;
10     return 0;}
11 int Suma(int *A,int n){
12     int S=0;
13     for(int i=0;i<n;i++)
14         S+=*(A+i);
15     return S;}
16 int * Elementoi(int *A,int i){
17     return A+i;}
```

Ejemplo uso de la memoria dinámica

```
int main(){  
    int *x=new int(12);  
    double *y=new double[10]{};  
    cout<<"Valor de la memoria dinamica creada: "<<*x;  
    Imprimir(y,10);  
    //Eliminacion de la memoria dinamica  
    delete x;  
    delete[] y;  
    return 0;  
}
```

Ejemplo uso de la memoria dinámica

```
void Imprimir(int **e, int n, int m);
int main(){
    int **e=new int*[2]{};
    *(e+0)=new int[10]{};
    e[1]=new int[5]{1,1,1,1,1};
    Imprimir(e,10,5);
    delete[] e[0];
    delete[] e[1];
    delete[] e;
    return 0;
}
```

```
void Imprimir(int **e, int n, int m){
    cout<<"Primera fila: ";
    for(int i=0;i<n;i++)
        cout<<**(e+0)<<" ";
    cout<<"\nSegunda fila: ";
    for(int i=0;i<m;i++)
        cout<<*e[1]<<" ";
}
```

Cadena

Una cadena o string es una estructura en los lenguajes, que se compone de una secuencia ordenada de caracteres de algún alfabeto.

Caracter \longrightarrow Símbolo \longrightarrow Alfabeto de lenguaje natural

Codificación de caracteres

Es el método que permite convertir un carácter de un lenguaje natural en un símbolo de otro sistema de representación:

- Código Morse
- ASCII (Código Estándar Estadounidense para el Intercambio de Información. American Standar Code for Information Interchange. Alfabeto Latino, (1963)).
ASCII \longrightarrow ASCII extendido. Cada caracter aquí representa un byte (8 bits, 256 símbolos). No es suficiente para todos los lenguajes del mundo.
- UNICODE (1991). Más de 50,000 símbolos. Universalidad, uniformidad y unicidad.
- UTF-8 (Unicode Transformation Format) Es un formato de codificación de caracteres Unicode donde se deja invariable el formato US-ASCII entre otras características.

Tabla ASCII extendida

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	#32;	Space	64	40	100	#64;	@	96	60	140	#96;	`
1	1	001	SOH (start of heading)	33	21	041	#33;	!	65	41	101	#65;	A	97	61	141	#97;	a
2	2	002	STX (start of text)	34	22	042	#34;	"	66	42	102	#66;	B	98	62	142	#98;	b
3	3	003	ETX (end of text)	35	23	043	#35;	#	67	43	103	#67;	C	99	63	143	#99;	c
4	4	004	EOF (end of transmission)	36	24	044	#36;	\$	68	44	104	#68;	D	100	64	144	#100;	d
5	5	005	ENQ (enquiry)	37	25	045	#37;	%	69	45	105	#69;	E	101	65	145	#101;	e
6	6	006	ACK (acknowledge)	38	26	046	#38;	&	70	46	106	#70;	F	102	66	146	#102;	f
7	7	007	BEL (bell)	39	27	047	#39;	'	71	47	107	#71;	G	103	67	147	#103;	g
8	8	010	BS (backspace)	40	28	050	#40;	(72	48	110	#72;	H	104	68	150	#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	#41;)	73	49	111	#73;	I	105	69	151	#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	#42;	*	74	4A	112	#74;	J	106	6A	152	#106;	j
11	B	013	VT (vertical tab)	43	2B	053	#43;	+	75	4B	113	#75;	K	107	6B	153	#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	#44;	,	76	4C	114	#76;	L	108	6C	154	#108;	l
13	D	015	CR (carriage return)	45	2D	055	#45;	-	77	4D	115	#77;	M	109	6D	155	#109;	m
14	E	016	SO (shift out)	46	2E	056	#46;	.	78	4E	116	#78;	N	110	6E	156	#110;	n
15	F	017	SI (shift in)	47	2F	057	#47;	/	79	4F	117	#79;	O	111	6F	157	#111;	o
16	10	020	DLE (data link escape)	48	30	060	#48;	0	80	50	120	#80;	P	112	70	160	#112;	p
17	11	021	DC1 (device control 1)	49	31	061	#49;	1	81	51	121	#81;	Q	113	71	161	#113;	q
18	12	022	DC2 (device control 2)	50	32	062	#50;	2	82	52	122	#82;	R	114	72	162	#114;	r
19	13	023	DC3 (device control 3)	51	33	063	#51;	3	83	53	123	#83;	S	115	73	163	#115;	s
20	14	024	DC4 (device control 4)	52	34	064	#52;	4	84	54	124	#84;	T	116	74	164	#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	#53;	5	85	55	125	#85;	U	117	75	165	#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	#54;	6	86	56	126	#86;	V	118	76	166	#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	#55;	7	87	57	127	#87;	W	119	77	167	#119;	w
24	18	030	CAN (cancel)	56	38	070	#56;	8	88	58	130	#88;	X	120	78	170	#120;	x
25	19	031	EM (end of medium)	57	39	071	#57;	9	89	59	131	#89;	Y	121	79	171	#121;	y
26	1A	032	SUB (substitute)	58	3A	072	#58;	:	90	5A	132	#90;	Z	122	7A	172	#122;	z
27	1B	033	ESC (escape)	59	3B	073	#59;	;	91	5B	133	#91;	[123	7B	173	#123;	{
28	1C	034	FS (file separator)	60	3C	074	#60;	<	92	5C	134	#92;	\	124	7C	174	#124;	
29	1D	035	GS (group separator)	61	3D	075	#61;	=	93	5D	135	#93;]	125	7D	175	#125;	}
30	1E	036	RS (record separator)	62	3E	076	#62;	>	94	5E	136	#94;	^	126	7E	176	#126;	~
31	1F	037	US (unit separator)	63	3F	077	#63;	?	95	5F	137	#95;	_	127	7F	177	#127;	DEL

Códigos ASCII extendidos

128	Ç	144	É	160	á	176	☐	192	Ł	208	Ł	224	α	240	■
129	à	145	Ê	161	â	177	☐	193	ł	209	ł	225	β	241	±
130	é	146	Ë	162	ó	178	☐	194	Ť	210	Ť	226	Γ	242	≥
131	ê	147	ô	163	ú	179		195	Ŧ	211	Ŧ	227	π	243	≤
132	ë	148	ö	164	ÿ	180	†	196	—	212	Ł	228	Σ	244	∫
133	ä	149	ò	165	Ñ	181	‡	197	†	213	Ŧ	229	σ	245	∫
134	å	150	û	166	ª	182	‡	198	‡	214	Ŧ	230	μ	246	+
135	ç	151	ù	167	º	183	¶	199	‡	215	‡	231	τ	247	×
136	ê	152	ÿ	168	¿	184	¶	200	‡	216	‡	232	φ	248	°
137	ë	153	Ö	169	ƒ	185	¶	201	Ŧ	217	Ŧ	233	⊖	249	·
138	è	154	Û	170	ƒ	186	¶	202	Ł	218	Ŧ	234	Ω	250	·
139	í	155	º	171	½	187	¶	203	Ŧ	219	■	235	δ	251	√
140	î	156	£	172	¾	188	¶	204	‡	220	■	236	∞	252	∞
141	ï	157	¥	173	¿	189	¶	205	=	221	■	237	φ	253	z
142	Ä	158	£	174	«	190	¶	206	‡	222	■	238	e	254	■
143	Å	159	ƒ	175	»	191	¶	207	Ł	223	■	239	∩	255	

Source: www.LookupTables.com

Lectura y escritura de Cadenas de arreglos

```
1  #include <iostream>
2  using namespace std;
3
4  int main(){
5      char cadena1[20];
6      char cadena2[20];
7      //Suponga que se lee "Hola mundo" en ambos casos.
8      cin.getline(cadena1,5,'a');
9      //cin.ignore(100,'\n');
10     cin.getline(cadena2,20,'\n');
11     cout<<"Contenido de la cadena 1: "<<cadena1<<endl;
12     cout<<"Contenido de la cadena 2: "<<cadena2<<endl;
13     return 0;}
```

Otras formas de inicializar una cadena

Inicializacion con punteros

```
char *Cadena=⟨cadena de caracteres⟩
```

En este estilo de inicialización la cadena es constante y por lo tanto no puede ser modificada.

Inicializacion con arreglos sin tamaño

```
char Cadena[] =⟨cadena de caracteres⟩
```

En este estilo de inicialización el tamaño del arreglo queda determinada por la longitud de la cadena (El puntero nulo también se toma en cuenta). En este caso la cadena si puede ser modificada.

Funciones de la librería String

La librería `cstring` o `string.h` proporciona un conjunto de funciones que permiten manipular cadenas de arreglos.

- La función **strlen** proporciona una forma de contar el número de caracteres de una cadena:

```
size_t strlen(const char *s);
```

- La función **strcpy** (`strcpy`) proporciona una forma de copiar una cadena en otra:

```
char *strcpy(char *s1, const char *s2);
```

En este caso copia la cadena `s2` en `s1`.

- La función **strcat** (`strncat`) proporciona una forma de concatenar una cadena en otra:

```
char *strcat(char *s1, const char *s2);
```

En este caso concatena la cadena `s2` en `s1`.

- Las funciones **strchr** y **strstr** actúan como funciones de búsqueda. La primera busca un carácter y la segunda busca una cadena; ambas retornan la dirección de la primera coincidencia.

```
char *strchr(char *s, int c);  
char *strstr(const char *s1, const char *s2);
```

Ejemplo de uso de la función strtok

Lista de palabras a partir de un parrafo

Considere el siguiente fragmento de la película *El Gladiador*:

”Mi nombre es Máximo Décimo Meridio, comandante de los ejércitos del norte, general de las Legiones Fénix, fiel servidor del verdadero emperador Marco Aurelio. Padre de un hijo asesinado, esposo de una esposa asesinada y juro que me vengaré, en esta vida o en la otra.”

Usando la función strtok determine lo siguiente:

- ¿Cuántas palabras tiene el fragmento anterior sin considerar los artículos y pronombres?

” yo, mí, conmigo, me, nosotros/as, nos, tú, usted, ti, contigo, te, vosotros/as, ustedes, os. él/ella, ello, sí, consigo, se, lo/la, le, ellos ellas, sí, consigo, los/las,un, una, unos, unas, el/los, la/las, lo”

- Convierta las iniciales de cada palabra considerada arriba a minúscula y posteriormente ordenelas de manera alfabética.

```

7      SetConsoleOutputCP( 65001 );
8      char Maximo[]="Mi nombre es Máximo Décimo Meridio, comandante\
9      de los ejércitos del norte, general de las Legiones Fénix,\
10     fiel servidor del verdadero emperador Marco Aurelio.\
11     Padre de un hijo asesinado, esposo de una esposa asesinada\
12     y juro que me vengaré, en esta vida o en la otra.";
13     char Articulos[]="yo, mí, conmigo, me, nosotros/as, nos, tú, usted
14     char Delimitadores[]=" ,.?!";
15     cout<<"La frase a analizar es: \n";
16     cout<<Maximo<<endl;
17     char *aux=strtok(Maximo,Delimitadores);
18     int cont=0;
19     while(aux!=nullptr){
20         if(strstr(Articulos,aux)==nullptr){
21             cont++;
22             if(cont==15)
23                 cout<<endl;
24             cout<<aux<<" ";}
25         aux=strtok(nullptr,Delimitadores);}
26     cout<<"\nEl número de palabras con la propiedad buscada es: "<<con
27     return 0;}

```

Problema 1

Desarrolle la siguiente función:

```
void Usuario(const char *Nombre,char *Us);
```

Esta función deberá generar un Nombre de usuario para el respectivo Nombre. La forma en que construirá el nombre de usuario sera la siguiente:

- El usuario deberá contener el nombre de pila, acompañado de las iniciales de sus apellidos en mayúsculas.
- Se le agregarán 4 dígitos al final de manera aleatoria.

Otros aspectos de C++ I

A continuación se listan algunos aspectos interesantes en el lenguaje de programación:

- 1 Tipo *enumerate*: En C++ es posible crear un tipo de enumeración; la idea consiste de en identificar un conjunto de llaves con un conjunto de numeros enteros. Por ejemplo, podrían identificarse los días de la semana con los primeros enteros no negativos de la siguiente forma:

Código 9: Ejemplo tipo enumerate

```
1 enum {enero , febrero , marzo , abril};  
2 enum dia{lunes ,martes ,miercoles ,jueves ,viernes};  
3 enum cotas{min=-1,max=1};  
4 cotas p=min;  
5 dia d=jueves;
```

En la primera línea se interpretará que enero tiene el valor de cero, febrero es uno y así. En la segunda declaración se sigue la misma lógica; sin embargo aquí se ha especificado el tipo para estas llaves. Por otro lado en la última línea se ha especificado el valor de cada llave además de especificar su tipo. En las últimas líneas p tendría el valor de -1 y d tendría el valor de 3.

- 2 Las *Referencias* en C++ son usadas normalmente para dos propósitos:

Otros aspectos de C++ II

- Retornar valores en los argumentos de una función.
- Sobrecarga de operadores.

En el siguiente código verás el primer aspecto.

Código 10: Ejemplo Referencias

```
1 #include <iostream>
2 void f(double &x){
3 x=12.2;}
4 int main(){
5     double x=10;
6     f(x);
7     cout<<x;
8     return 0;
9 }
```

En el código anterior se consigue que al final se imprima en pantalla el valor de 12.2.

- 3** Operadores de *conversion*: En C++ existen cuatro formas de convertir datos en otros. Por ejemplo se puede transformar un tipo entero en uno de tipo double o al revés. Estas cuatro formas son las siguientes:

- *static_cast<tipo nuevo>(expresión)*: Se usa para la conversión entre tipos relacionados.

- *dynamic_cast<tipo nuevo>(expresión)*: Se usa en el contexto de la herencia de clases. Por ejemplo puede convertir una clase padre en una hijo. Si no es posible la conversión retorna un puntero nulo.
- *const_cast<tipo nuevo>(expresión)*: Se usa para modificar un calificador constante de una variable. Considere el siguiente ejemplo tomado de la página de [GeeksforGeeks](#).

Código 11: Ejemplo const_cast

```
1  const int number = 5;  
2  const int* ptr = &number;  
3  int* nonConstPtr = const_cast<int*>(ptr);
```

En el ejemplo anterior, si se hubiera intentado asignar ptr directamente a nonConstPtr el compilador hubiese lanzado un error; para solucionarlo es necesario hacer la conversión.

- *reinterpret_cast<tipo nuevo>(expresión)*: Se usa para convertir cualquier puntero en otro sin hacer ninguna verificación.

- 4 Espacios de nombres; *namespace*: Esto representa un mecanismo por el cual se puede agrupar un conjunto de expresiones que normalmente tienen algún sentido que los conecta.

En el ejemplo se puede apreciar como se crea un espacio de nombres y como se accede a los elementos de un espacio de nombres:

Código 12: Ejemplo Namespace

```
1 #include <iostream>
2 namespace Tiempo{
3     enum dia{lunes , martes , miercoles , jueves , viernes };
4     enum cotas{min=-1,max=1};
5     const int anio=2023;
6 }
7 using namespace std;
8 int main(){
9     cout<<Tiempo::jueves<<endl;
10    Tiempo::dia primero=Tiempo::lunes;
11    cout<<"Anio actual:_"<<Tiempo::anio<<endl;
12    return 0;}
```

Creación de Archivos de Cabecera en Visual Studio I

Para manejar varios archivos de cabecera se pueden seguir los siguientes pasos:

- 1 Se crea un archivo con extensión h, supongamos que se almacenará como "EjemploCabecera.h":

```
1 #ifndef NombreArchivoMayusuculas_H
2 #define NombreArchivoMayusuculas_H
3 //Codigo de estructura de la clase
4 class Ejemplo{
5 //Codigo
6 };
7 #endif
```

- 2 Se crea un archivo con extensión cpp del mismo nombre que el archivo de cabecera. En este apartado se tendrá todas las implementaciones de las funciones del archivo de cabecera.

Código 13: "EjemploCabecera.cpp"

```
1 //Librerias necesarias para el funcionamiento
2 #include "EjemploCabecera.h";
3 // de las funciones de la clase.
4 //Codigo
```

- 3 Luego en el programa principal se deberá incluir el archivo de cabecera:

Código 14: "main.cpp"

```
1 //Librerias necesarias para el funcionamiento
2 #include "EjemploCabecera.h";
3 //Codigo del programa principal
```

- 4 Finalmente en la ventana de comandos se creara un "Ejecutable.exe":

```
>g++ -g -c EjemploCabecera.cpp main.cpp
>g++ -g -o Ejecutable EjemploCabecera.o main.o
>./Ejecutable
```

Programación orientada a objetos, clases I

La estructura general de una clase es la siguiente:

```
1 class Clase_Modelo{  
2 //Zona de parametros miembros  
3 Public :  
4 //Zona de funciones miembros  
5 Clase_Modelo();  
6 //Clase_Modelo(Lista de parametros)  
7 };
```

Función Constructor

Esta tiene la finalidad de proveer un mecanismo para la inicialización de un objeto de la clase en cuestión.

Observaciones alrededor de la función constructor:

- En el caso de los constructores se puede aprovechar el concepto de *sobrecarga de funciones*.
- Para crear una copia en el constructor de un objeto de la misma clase, se pueden crear las funciones que retornan los parámetros privados de la clase. Puede revisar el código de muestra en [Enlace a GitHub](#)

Programación orientada a objetos, clases II

- Si se acompaña el modificador *const* a la función que retornará los parámetros de una clase, entonces esto asegura que no se cometa el error de modificar los parámetros privados de la clase.

Herencia I

La propiedad de herencia en el contexto de la programación orientadas a objetos consiste en la posibilidad de crear una jerarquía entre dos o más clases.

El principal objetivo consiste en eliminar la redundancia de datos y funciones entre clases que están relacionadas por algún orden de jerarquía.

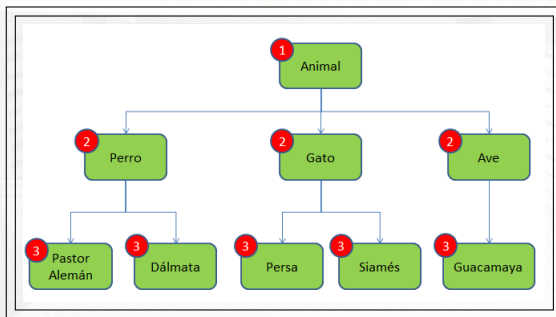


Figura: La imagen se encuentra en la siguiente dirección *Dirección Web*

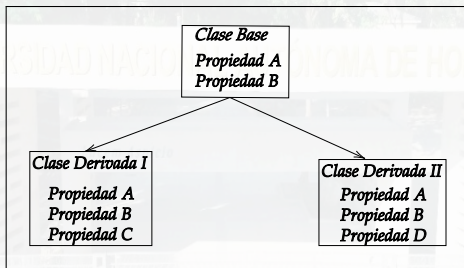


Figura: En la figura se puede observar como las clases derivadas contienen sus propias características y las de su clase derivada.

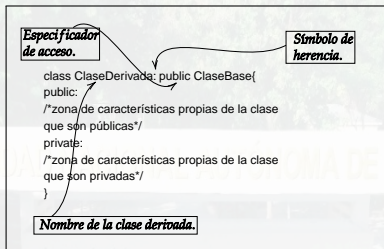


Figura: Plantilla para la escritura de una clase derivada en C++.

Existe un tercer especificador de acceso; el tipo protegido. En la siguiente tabla se muestra el nivel de acceso de cada tipo:

Miembro	M. de la misma C.	C. Amiga	Clase derivada	F. no miembro
public	Si	Si	Si	Si
protected	Si	Si	Si	No
private	Si	Si	No	No

Cuadro: Como se puede apreciar, la principal diferencia cuando usamos el tipo protegido se da en el caso de la herencia.

Dependiendo del *especificador de acceso* los miembros de la clase base vistos desde la clase derivada tendrán un tipo diferente de visibilidad. La siguiente tabla

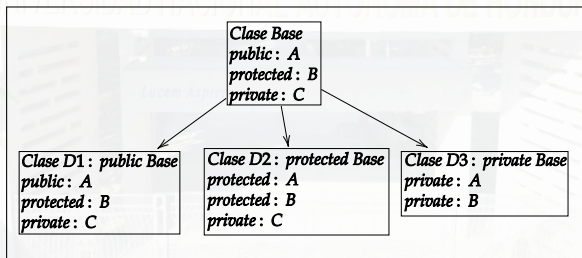


Figura: En la imagen se pueden apreciar como cambia la visibilidad de los miembros de la clase base en sus clase derivadas.

Plantilla para codigos

```
1 #include <iostream>
2 using namespace std;
3 int main(){
4     return 0;
5 }
```

Esta es una salida.



Stroustrup, Bjarne (2013). *El lenguaje de programación C++ Cuarta edición.*