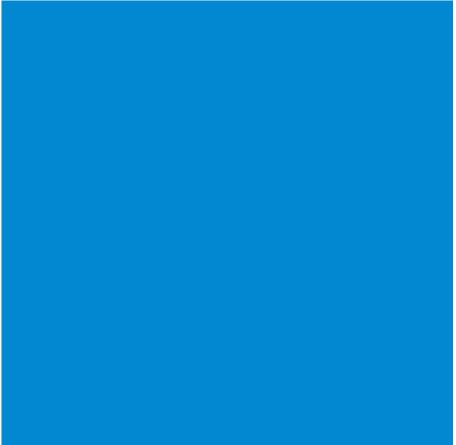




LENGUAJES DE PROGRAMACIÓN I



UNIDAD 1

Fundamentos del Lenguaje

OBJETIVO DE LA ASIGNATURA

Con el estudio de esta asignatura buscaremos analizar el paradigma estructurado de programación y ser capaces de aplicar las sintaxis de un lenguaje orientado a objetos (C++) para aplicarlo en la solución de problemas.

COMPETENCIAS A DESARROLLAR

Al concluir la unidad, serás capaz de:

- Analizar los fundamentos del lenguaje, así como los elementos básicos que intervienen en el desarrollo del mismo.
- Comprender el proceso de construcción de un programa y saber usar las herramientas que se requieren: consola, editor y compilador.
- Aplicar la sintaxis de los operadores de las expresiones e instrucciones básicas de un lenguaje de programación imperativo (C++).



TEMARIO

- 1.1** Entorno de desarrollo (Windows, Linux)
- 1.2** Configuración del entorno del desarrollo (Windows, Linux)
- 1.3** Palabras reservadas
- 1.4** Comentarios
- 1.5** Tipos de datos
- 1.6** Variables
- 1.7** Constantes
- 1.8** Operadores
- 1.9** Operadores Incremento y Decremento
- 1.10** Sentencias
- 1.11** Conversión de tipos de datos (cast)

INTRODUCCIÓN

C++ es un lenguaje de programación orientado a objetos que toma la base del lenguaje C y le agrega la capacidad de abstraer tipos como en Smalltalk. Fue diseñado a mediados de los años 80 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitieran la manipulación de objetos.

Los tipos de datos en C++, son tipos primitivos como su nombre lo da a entender, son los tipos de datos más básicos y simples del sistema de tipos de C++ bastante fácil de usar. Se utilizan para que un programa pueda hacer uso de una o más variables y deben ser declaradas previamente. Todas las variables de un programa se declaran de la misma forma, indicando de cada una de ellas el tipo de dato que puede almacenar y su nombre.

En la mayoría de los casos, la sintaxis de instrucción de C++ es idéntica de ANSI C. La diferencia principal entre los dos es que en C, se permiten declaraciones solo al principio de un bloque; C++ agrega la instrucción de declaración, que elimina eficazmente esta restricción.

1.1. ENTORNO DE DESARROLLO (WINDOWS LINUX)

Los entornos de desarrollo integrados (IDEs) para C++, hacen uso de un concepto de proyecto. Un proyecto en un entorno de desarrollo (IDE) cualquiera, es un "contenedor" global, donde podemos incluir o crear todos los archivos necesarios que tengan relación con una aplicación de software que estemos desarrollando, clases, interfaces, archivos de texto, imágenes, paquetes, entre otros. Existen entornos de desarrollos que se pueden utilizar para programar. A continuación, verás una lista de entornos que podemos utilizar para programar en C++.

En Windows puedes usar:	En Linux puedes usar:
Visual C++	Gedit
Visual Studio	Geany
Notepad++	Kate
Dev C++	KDevelop
Code::Blocks	Eclipse
Eclipse	Code::Blocks
Ultimate++	Ultimate++
Kdevelop	CodeLite

1.1. ENTORNO DE DESARROLLO (WINDOWS LINUX)

Las principales herramientas necesarias para escribir un programa en C++ son las siguientes:

- **Un equipo ejecutando un sistema operativo.**
- **Un compilador de C++**
 - Windows MingW (GCC para Windows) o MSVC (compilador de microsoft con versión gratuita).
 - Linux (u otros UNIX): g++
- **Un editor cualquiera de texto, o mejor un entorno de desarrollo (IDE)**
 - Windows
 - Linux



A hand is holding a smartphone, which displays a Python script titled "mirror_mod.py". The script contains code related to mirroring objects in a 3D scene. The code includes variables like `mirror_mod.use_x`, `mirror_mod.use_y`, and `mirror_mod.use_z`, and operations like "MIRROR_X", "MIRROR_Y", and "MIRROR_Z". It also includes logic for selecting objects and printing messages. The phone is held against a dark background.

```
mirror_mod.use_x = True
mirror_mod.use_y = False
mirror_mod.use_z = False
operation == "MIRROR_Y"
mirror_mod.use_x = False
mirror_mod.use_y = True
mirror_mod.use_z = False
operation == "MIRROR_Z"
mirror_mod.use_x = False
mirror_mod.use_y = False
mirror_mod.use_z = True

selection at the end -add
__ob.select= 1
ler_ob.select=1
context.scene.objects.active
("Selected" + str(modifier))
mirror_ob.select = 0
bpy.context.selected_objects
data.objects[one.name].sel
print("please select exactly one object")
--- OPERATOR CLASSES ---

types.Operator):
on X mirror to the selected
object.mirror_mirror_x"
mirror X"
context):
context.active_object is not
```

1. Escribir el programa en un archivo con extensión .cpp para C++ o la extensión .c para C, mediante un editor de textos.

Este archivo se denomina código fuente.

2. Compilar el código fuente. Esto es, traducirlo a un lenguaje comprensible por el ordenador. Los archivos resultantes de este proceso se denominan archivos objeto y suelen identificarse mediante la extensión .o u .obj.

3. Un programa puede estar repartido entre distintos archivos, cada uno de los cuales se habrá compilado independientemente.

Por ello, antes de ejecutar el programa, es necesario unir o ligar los archivos objeto (.o u .obj) que lo componen. Esto se hace con un enlazador (en inglés linker). El resultado final es el programa ejecutable, cuya extensión suele ser .exe.

1.1. ENTORNO DE DESARROLLO (WINDOWS LINUX)

1.2. CONFIGURACIÓN DEL ENTORNO DEL DESARROLLO EN (WINDOWS/LINUX)

```
.../depmod -> $(SEC_CRIT) :  
insmod -> $(SEC_CRIT) :  
insmod.static -> $(SEC_CRIT) :  
insmod_ksymoops_clean -> $(SEC_CRIT) :  
klogd -> $(SEC_CRIT) :  
ldconfig -> $(SEC_CRIT) :  
minilogd -> $(SEC_CRIT) :  
.../depmod -> $(SEC_CRIT) :  
insmod -> $(SEC_CRIT) :  
insmod.static -> $(SEC_CRIT) :  
insmod_ksymoops_clean -> $(SEC_CRIT) :  
klogd -> $(SEC_CRIT) :  
ldconfig -> $(SEC_CRIT) :  
minilogd -> $(SEC_CRIT) :
```



- El compilador de C++ de GNU es un compilador de línea de órdenes que compila y enlaza programas en C++, generando el correspondiente archivo ejecutable. G++ está incorporado de forma estándar en los sistemas Linux, pero no para los sistemas Windows. En estos sistemas necesitamos una aplicación que incorpore el compilador en Windows. Para descargar MinGW usa el siguiente vínculo: **<https://sourceforge.net/projects/mingw/files/>**
- Por defecto MinGW se instala en la carpeta C:\MinGW. Para poder ejecutar el compilador G++ desde cualquier otra carpeta necesitamos añadir la ruta de la carpeta C:\MinGW\bin\ a la variable de entorno PATH. Esto lo puedes hacer con la utilidad Sistema del Panel de control. Abre el Panel de control de Windows y pulsa sobre Sistema. Se abrirá esa utilidad: Pulsa en Configuración avanzada del sistema.

1.2. CONFIGURACIÓN DEL ENTORNO DEL DESARROLLO EN (WINDOWS/LINUX)

- El botón Variables de entorno te permite ver y modificar ese aspecto. Son variables que usa el Sistema Operativo para mantener valores globales que puede necesitar en cualquier momento, o listas de carpetas en las que se puede buscar algo, como en el caso de la variable de entorno PATH, que contiene las carpetas en las que puede haber archivos ejecutables.
- La variable PATH (es igual en mayúsculas o minúsculas) contiene las carpetas de búsqueda separadas por ;. Para añadir una nueva ruta, selecciona la variable en la lista Variables del sistema y pulsa el botón Modificar.
- Para añadir la ruta de MinGW, pulsa la tecla Fin, de forma que el cursor de texto se ponga al final de la cadena, y escribe ";C:\MinGW\bin\" (sin las comillas, sin espacios y SIN OLVIDAR el punto y coma del principio) (si lo instalaste en otro disco, cambia C por la letra de unidad correspondiente).

1.2. CONFIGURACIÓN DEL ENTORNO DEL DESARROLLO EN (WINDOWS/LINUX)

C++ en un entorno Linux

Comenzaremos diciendo que los programas se pueden escribir en cualquier editor de textos de GNU, entre ellos se encuentran emacs, vim, kate, gedit, nano, guardando dichos archivos con extensión .cpp, los cuales serán compilados en GNU/linux utilizando el compilador GNU de C++, llamado gcc que puede compilar C, C++, y que además se apega al estándar ANSI, permitiendo la portabilidad de estos códigos. Dicho compilador se invoca con el comando gcc.

Para compilar ponemos la siguiente línea en una terminal previamente ubicada en el directorio que contiene nuestro archivo:

g++ programa.cpp -o programa.out

o indica el nombre del archivo de salida, el cual sería el ejecutable de nuestro proyecto.

Luego para ejecutar, escribimos sobre la línea de comandos:

./programa.out y entonces podremos ejecutar nuestro programa.

PARA APRENDER MÁS

Te recomendamos las siguientes lecturas:

Operadores en C++

[CLIC AQUÍ](#)

Tutorial para instalar Code Block

[CLIC AQUÍ](#)

TE INVITAMOS A VER LOS SIGUIENTES VIDEOS

Crear un proyecto en codeblocks

CLIC AQUÍ

Crear un archivo en codeblocks

CLIC AQUÍ

1.3. PALABRAS RESERVADAS

Las palabras reservadas son identificadores predefinidos que tienen significados especiales y no pueden usarse como identificadores creados por el usuario en los programas. Las palabras reservadas de C++ pueden agruparse en 3 grupos.

El primer grupo contiene las palabras del lenguaje C, y que C++ como evolución de C, también contiene:

auto	const	double	float	int	short	struct
break	continue	else	for	signed	signed	switch
case	default	enum	goto	sizeof	sizeof	typedef
char	do	extern	if	static	static	union
While	volatile	void	unsigned			

1.3. PALABRAS RESERVADAS

Palabras que no provienen de C y que, por tanto, solo utiliza C++:

asm	dynamic_cast	class	reinterpret_cast	try	bool
new	static_cast	typeid	catch	false	operator
typename	namespace	friend	private	this	using
inline	public	throw	const_cast	delete	mutable
true	wchar_t	explicit	template	virtual	protected

1.3. PALABRAS RESERVADAS

Las siguientes palabras reservadas se han añadido como alternativas para algunos operadores de C++ y hacen los programas más legibles y fáciles de escribir:

and	dynamic_cast	compl	not_eq
not_eq	not_eq	xor_eq	and_eq
bitor	not	or	xor

TE INVITAMOS A VER EL SIGUIENTE VIDEO

Entrada de datos

CLIC AQUÍ

1.4. COMENTARIOS

Los comentarios son líneas de código que no son tomadas en cuenta por el compilador en el momento de ejecutar nuestra aplicación, por lo tanto no están sujetas a restricciones de sintaxis ni nada similar, el uso principal de las líneas de comentario, es dar un orden y hacer más entendible nuestro código, especialmente en el caso de que deba ser revisado en algún momento o leído por alguien diferente a nosotros.

Existen dos tipos de comentarios en el lenguaje: Comentarios de Una Sola Línea y Comentarios Multi-línea.

1.4. COMENTARIOS

Comentarios de Una Sola Línea: Pueden ser colocados en cualquier parte y comienzan por un doble slash "://" ; al colocar el doble slash al comienzo de cualquier línea de código, de ahí en adelante en esa misma línea será tomado como comentario.

Ejemplo: int i=0; //Declaración de variable

Comentarios Multi-línea: Van cerrados entre "/*" y "*/". Estos comentarios son similares a los anteriores, pero deben tener un comienzo y un final, a diferencia del anterior, al poner los símbolos "/*" todo el código que haya tanto en la misma línea, como en las líneas debajo de este, se convertirán en comentarios.

Ejemplo: float b;
/*Esta sentencia se define el tipo de dato de la variable*/

1.5. TIPOS DE DATOS

El tipo de dato determina la naturaleza del valor que puede tomar una variable. Un tipo de dato define un dominio de valores y las operaciones que se pueden realizar con estos valores. En C++ se dispone de unos cuantos tipos de datos predefinidos (simples) y permite al programador crear otros tipos de datos.

Cabe mencionar que C++ es un lenguaje orientado a objetos y posee una cantidad enorme de librerías o bibliotecas que podemos usar, estas librerías nos proporcionan una serie de tipos de datos adicionales, sin embargo estos ya no son tipos de datos simples sino complejos y por ende, van a poseer una serie de campos y funcionalidades adicionales que no posee un tipo sencillo.

1.5. TIPOS DE DATOS

La siguiente tabla indica los tipos de datos sencillos:

Tipo	Descripción	Tamaño	Dominio
Int	Números enteros	2 bytes (16 bits)	Son todos los números enteros enteros entre los valores -32.768 y 32.767
Float	Números Reales	4 bytes	Son todos los números reales que contienen una coma decimal comprendidos entre los valores: 3.4×10^{-38} y 3.4×10^{38}
Double	Números reales más grandes que float	8 bytes	Son todos los números reales que contienen una coma decimal comprendidos entre los valores: 1.7×10^{-308} y 1.7×10^{308}
Bool	Valores lógicos	1 byte	Dos únicos valores: { true, false }
Char	Caracteres y cualquier cantidad de 8 bits	1 byte	Dígitos, letras mayúsculas, letras minúsculas y signos de puntuación.
Void	El tipo vacío. Sirve para indicar que una función no devuelve valores.		

1.6. VARIABLES



```
require( TEMPLATEDPATH.DS."Yjsgcore/Yjsg_styles.php");
$renderer = $document->loadRenderer('module');
$options = array( 'style' => "raw" );
$module = JModuleHelper::getModule( 'mod_menu' );
$topmenu = false; $subnav = false; $sidenav = false;
if ($default_menu_style == 1 or $default_menu_style == 2) {
    $module->params = "menutype=$menu_name\&showAllChildren=1\&menu";
    $topmenu = $renderer->render( $module, $options );
    $menuclass = 'horiznav';
    $topmenuclass = 'top_menu';
} elseif ($default_menu_style == 3 or $default_menu_style == 4) {
    $module->params = "menutype=$menu_name\&showAllChildren=0\&menu";
    $topmenu = $renderer->render( $module, $options );
    $menuclass = 'horiznav_d';
    $topmenuclass = 'top_menu_d';
}
```

- Una variable es un espacio reservado en el ordenador para contener valores que pueden cambiar durante la ejecución de un programa. Los tipos determinan cómo se manipulará la información contenida en esas variables.
- Para asignar valores a una variable en una gran variedad de lenguajes que incluye a C++, se usa el operador "=" seguido del valor que le daremos a la variable (no todos usan el "="para esto).
- La sintaxis para una variable es la siguiente: [modificadores] [tipo de variable] [nombre de la variable] [=] [valor];

Ejemplo completo con todos los posibles usos que le damos a una variable.

```
#include <iostream>
using namespace std;
int main()
{
    char x = 'a'; // Declaramos y asignamos en la misma línea
    int num; //Declaramos el entero en una línea
    num = 5; //Le asignamos un valor en otra línea
    int num2 = 8; //Asignación y declaración al tiempo
    float numero; //Un numero decimal
    numero = 3.5; //Le asignamos un valor al decimal
    float res = numero + num2; //Sumamos dos variables y las asignamos a res
    //3.5 + 8 = 11.5
    res = res + num; //Al valor actual de res le sumamos el valor de num
    //11.5 + 5 = 16.5
```

1.6. VARIABLES

Ejemplo completo con todos los posibles usos que le damos a una variable.

```
bool valor = false; //Variable booleana  
valor = true; // Pueden ser true o false  
res = res*2; //Duplicamos el valor de res 16.5*2 = 33  
cout << res << endl; //Mostramos el valor de res por pantalla  
return 0;  
}
```

En el código anterior se muestran las diferentes formas en que se puede declarar una variable; para saber cómo asignarle un valor, incluso es posible asignarlo de una variable a otra, o realizar operaciones entre los valores de variables y asignar el resultado a una variable nueva. También es posible usar el valor de una misma variable y cambiar su propio valor (*res = res*2*).

1.6. VARIABLES

TE INVITAMOS A VER EL SIGUIENTE VIDEO

Sintaxis básica

CLIC AQUÍ

1.7. CONSTANTES

Las constantes son muy útiles para especificar el tamaño de un vector y para alguna variable de tamaño específico. Para declarar una constante, se hace después de declarar las librerías y antes de las funciones.

La sintaxis es la siguiente: `#define nombre_constante valor`.

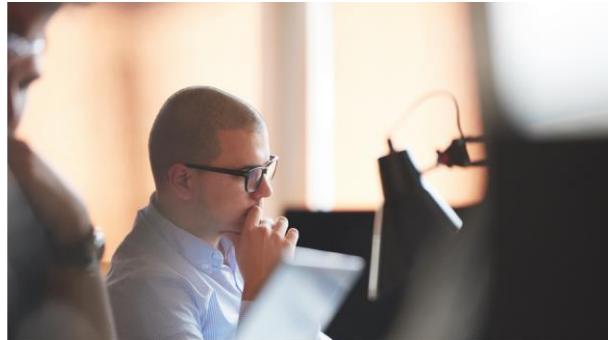
En C++ se pueden definir constantes de dos formas. La primera es por medio del comando `#define nombre_constante valor` y la segunda es usando la palabra clave `const`.

1.7. CONSTANTES

La instrucción `#define` nos permite declarar constantes con un valor determinado. Hay que tener en cuenta que al declarar constantes con `#define` debemos hacerlo después de los `#include` para importar librerías, pero antes de declarar nuestras funciones y demás. Ejemplo:

```
#include <iostream>
using namespace std;
#define PI 3.1416; //Definimos una constante llamada PI
int main()
{
    cout << "Mostrando el valor de PI: " << PI;
    return 0;
}
```

1.7. CONSTANTES



```
<!-- class="mod-centered homepage-widgets" -->
<div one-third bordered first>...</div>
<div one-third bordered middle>...</div>
<div one-third bordered last>...</div>
<div class="magnifier" style="display: none;"></div>
<div class="image-magnifier" style="display: none;"></div>
<div limited><div third bordered first>...</div>
<div third bordered middle>...</div>
<div third bordered last>...</div>
</div>
```

La instrucción `const` nos permite declarar constantes en el interior del main. Las constantes declaradas con `const` poseen un tipo de dato asociado (como debería ser siempre) y se declaran al interior de nuestro código como un tipo cualquiera. Ejemplo:

```
#include <iostream>
using namespace std;
int main()
{
    const float PI = 3.1416;
    //Definimos una constante llamada PI
    cout << "Mostrando el valor de PI: " << PI << endl;
    return 0;
}
```

1.8. OPERADORES

Un operador es un elemento de programa que se aplica a uno o varios operadores en una expresión o instrucción. Los operadores que toman un operando, como el operador de incremento (`++`) o `new`, se conocen como operadores unarios. Los operadores que toman dos operadores, como los operadores aritméticos (`+, -, *, /`) se conocen como operadores binarios. Existen 3 tipos de operadores, que son:

- Operadores Aritméticos
- Operadores Racionales
- Operadores Lógicos



1.8. OPERADORES

Los operadores aritméticos se usan para realizar cálculos de aritmética de números reales y de aritmética de punteros.

Operador	Nombre	Ejemplo	Descripción
+	Suma	$5 + 6$	Suma dos números.
-	Substracción	$7 - 9$	Resta dos números.
*	Multiplicación	$6 * 3$	Multiplica dos números.
/	División	$4 / 8$	Divide dos números.
%	Módulo: el resto después de la división.	$7 \% 2$	Devuelve el resto de dividir ambos números, en este ejemplo el resultado es 1.
++	Incremento	<code>a++</code>	Suma 1 al contenido de una variable.
--	Decremento	<code>a--</code>	Resta 1 al contenido de una variable.
-	Invierte el signo de un operando.	<code>-a</code>	Invierte el signo de un operando.

1.8. OPERADORES

Los operadores relacionales, también denominados operadores binarios lógicos y de comparación, se utilizan para comprobar la velocidad o falsedad de determinadas operación de relación. Son símbolos que se usan para comprobar dos valores. Si el resultado de la comparación es correcto, la expresión considerada es verdadera, en caso contrario es falsa.

Operador	Nombre	Ejemplo	Descripción
<	menor que	a<b	a es menor que b
>	mayor que	a>b	a es mayor que b
==	igual a	a==b	a es igual a b
!=	no igual a	a!=b	a no es igual a b
<=	menor que o igual a	a<=5	a es menor que o igual a b
>=	mayor que o igual a	a>=b	a es menor que o igual a b

1.8. OPERADORES

Los operadores lógicos son generalmente empleados con valores lógicos (booleanos), estos operadores devuelven un valor booleano.

Operador	Operación	Ejemplo	Descripción
<code>&&</code>	AND	<code>A&&B</code>	Si ambos son verdaderos se obtiene verdadero (true).
<code> </code>	OR	<code>A // B</code>	Verdadero si alguno es verdadero.
<code>!</code>	NOT	<code>!A</code>	Negación de A.

TE INVITAMOS A VER LOS SIGUIENTES VIDEOS

Operación con los tipos de datos

CLIC AQUÍ

Operación con variables

CLIC AQUÍ

1.9. OPERADORES INCREMENTO Y DECREMENTO

Si los operadores ++ y -- están de prefijos, la operación de incremento se efectúa antes de la operación de asignación.

Si los operadores ++, -- están de sufijos, la asignación se efectúa en primer lugar y el incremento o decremento a continuación.

Ejemplo del operador de incrementación

```
int a = 1, b;  
b = ++a;
```

¿Cuál es el valor de a y de b?

```
int a = 1, b;  
b = a++ // b vale 1 y a vale 2
```

1.10. SENTENCIAS

Las expresiones de C++ son unidades o componentes elementales de unas entidades de rango superior que son las sentencias. Las sentencias son unidades completas, ejecutables en sí mismas e incorporan expresiones aritméticas, lógicas o generales como componente de dichas sentencias.

Sentencias simples: Una sentencia simple es una expresión de algún tipo terminada con un carácter (;). Un caso típico son las declaraciones o las sentencias aritméticas.

Ejemplo float real;

```
espacio = espacio_inicial + velocidad * tiempo;
```

Sentencias nulas o vacías: En algunas ocasiones es necesario introducir en el programa una sentencia que ocupe un lugar pero que no realice ninguna tarea. A esta sentencia se le denomina sentencia vacía y consta de un simple carácter (;

Ejemplo ;

1.10. SENTENCIAS

Sentencias Compuestas o Bloques: Muchas veces es necesario poner varias sentencias en un lugar del programa donde debería haber una sola. Esto se realiza por medio de sentencias compuestas. Una sentencia compuesta es un conjunto de declaraciones y de sentencias agrupadas dentro de llaves { . . . }.

También se conocen con el nombre de bloques. Una sentencia compuesta puede incluir otras sentencias simples y compuestas.

```
Ejemplo {  
int i=1, j=3, k;  
double masa;  
masa= 3.0;  
k=y+j;  
}
```

1.11. CONVERSIONES IMPLÍCITAS Y EXPLÍCITAS DE TIPO (CASTING)



Las *conversiones implícitas* de tipo tienen lugar cuando en una expresión se mezclan variables de distintos tipos. Por ejemplo, para poder sumar dos variables hace falta que ambas sean del mismo tipo. Si una es *int* y otra es *float*, la primera se convierte en *float* (es decir, la variable del tipo de menor rango se convierte al tipo de mayor rango), antes de realizar la operación. A esta conversión automática e implícita de tipo (el programador no necesita intervenir, aunque sí conoce sus reglas), se le denomina promoción, pues la variable de menor rango es promocionada al rango de la otra.

Los rangos de las variables de mayor a menor se ordenan del siguiente modo:

long double > double > float > unsigned long > long > unsigned int > int > unsigned short > short > char

En C++ existe también la posibilidad de realizar conversiones explícitas de tipo (llamadas Casting). El casting es pues una conversión de tipo, forzada por el programador. Para ello basta preceder la constante, variable o expresión que se desea convertir por el tipo al que se desea convertir, encerrando entre paréntesis.

Ejemplo: K = (int) 2.5 + (int) masa;

El lenguaje C++ dispone de otra conversión explícita de tipo con una notación similar a la de las funciones y más sencilla que la del Cast. Se utiliza para ello el nombre del tipo al que se desea convertir seguido del valor a convertir entre paréntesis. Así, las siguientes expresiones son válidas.

Ejemplo: y= double(25);
double x = 5;
return int(x/y);

1.11. CONVERSIONES IMPLÍCITAS Y EXPLÍCITAS DE TIPO (CASTING)

ACTIVIDAD INTEGRADORA 1

Realice los siguientes programas:

EJERCICIO 1.-

Escriba las sentencias necesarias para solicitar e introducir tres valores de punto flotante y después producir su promedio.

EJERCICIO 2.-

Realice un programa que lea la entrada de dos números y muestre en la salida estándar su suma, resta, multiplicación y división.

EJERCICIO 3.-

Escribir un programa que dé la entrada el precio de un producto y muestre en la salida estándar el precio del producto al aplicarle el IVA.

ACTIVIDAD INTEGRADORA 1

EJERCICIO 4.-

Realice un programa que lea la entrada de los siguientes datos de una persona:

Edad: dato de tipo entero.

Sexo: dato de tipo carácter.

Altura en metros: dato de tipo real.

tras leer los datos, el programa debe mostrar la salida en una sola línea.

EJERCICIO 5.-

Dada de entrada del nombre de una persona y el día, mes y año de nacimiento, escriba un programa que determine su edad y dé salida el nombre de la persona y la edad.

EJERCICIO 6.-

Realice un programa que pueda calcular un índice de masa corporal (IMC). Tomando en cuenta que para realizar la operación los datos de entrada son el peso: $IMC = \text{peso}/(\text{estatura}^*\text{estatura})$

ACTIVIDAD INTEGRADORA 1

EJERCICIO 7.-

Realice un programa de la entrada a un valor numérico y muestra la salida de incremento y decremento del valor dado.

EJERCICIO 8.-

Realice un programa dé la entrada a un valor correspondiente a una distancia en millas y las visualice expresada en metros. Sabiendo que una milla marina equivale a 1852 metros.

EJERCICIO 9.-

Cuando se realiza una compra al contado en una tienda departamental, al realizar el pago algunas veces el cliente no trae el pago exacto. Utiliza un programa para realizar el cambio que corresponde del pago realizado sobre la compra. Considerando que el cambio únicamente debe de ser monedas de 5, 10, 20 y 50 que lea de la entrada estándar el importe de la compra y la cantidad de dinero con la que paga por la compra comprador escriba en la salida estándar las monedas devueltas.

ACTIVIDAD INTEGRADORA 1

CLIC AQUÍ

EJERCICIO 10.-

Suponiendo que el recibo de la luz sube un 3 % cada año, realice un programa que solicite una factura de este año y una cantidad de años y muestre en la salida estándar cuanto valdría la factura dentro del número de años introducidos.

FORO 1

CLIC AQUÍ

Participa en el foro dando respuesta a los siguientes interrogantes:

1. ¿Cuál es el papel de un compilador?
2. ¿Qué tipo de errores descubre el compilador?
3. Si en un programa omite un símbolo de puntuación (por ejemplo, un punto y coma), se produce un error. ¿De qué tipo?

PARA APRENDER MÁS

Te recomendamos las siguientes lecturas:

Deitel, H. M. y Deitel, P. J., Cómo programar en C++. Prentice Hall 6ta Edición 2008
Capítulo 2

[CLIC AQUÍ](#)

Deitel, H. M. y Deitel, P. J., Cómo programar en C++. Prentice Hall 9Na Edición 2014
Capítulo 2

[CLIC AQUÍ](#)

Gary J. Bronson, C++ para Ingeniería y Ciencias. Cengage Learing 2da Edición 2006
Capítulo 2,3

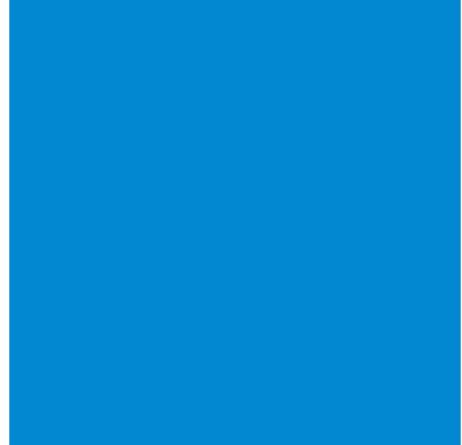
[CLIC AQUÍ](#)

CONCLUSIÓN

Actualmente, C++ es uno de los mejores lenguajes en el uso eficiente de los recursos, y lo mejor de todo es que esta cualidad no disminuye en proyectos de una complejidad elevada.

En programación, existen diversos conceptos que son fundamentales y que se deben conocer tales como datos, compiladores, depuradores, entre otros. Dichos conceptos hacen la labor un tanto más sencilla. Una particularidad del C++ es la posibilidad de redefinir los operadores (sobrecarga de operadores), y de poder crear nuevos tipos que se comporten como tipos fundamentales.

La salida de datos en C++ consiste en enviar datos que, generalmente, son el resultado de un procesamiento desde la memoria principal hacia un dispositivo de salida.



UNIDAD 2



Estructuras de Control y Decisión

COMPETENCIAS A DESARROLLAR

Al concluir la unidad, serás capaz de:

- Aplicar la sintaxis de las diferentes estructuras de control de flujo y decisión con la capacidad de definir cuáles son las sentencias más adecuadas para resolver un problema a través de un programa.
- Utilizar la sintaxis de función para el desarrollo de programas que hagan uso del ahorro de espacio y para la reducción de repeticiones, proporcionando un medio para dividir un proyecto grande en módulos pequeños más manejables.



TEMARIO

1.1 Entorno de desarrollo (Windows, Linux)

2.1 Estructuras de control

2.1.1 Estructuras Selectivas

2.1.2 Estructuras Repetitivas

2.2 Funciones

2.2.1 Parámetros de una Función

2.2.2 Llamada de una función

INTRODUCCIÓN

Los condicionales en C++, son una estructura de control esencial al momento de programar y aprender a programar. Tanto C como C++ y la mayoría de los lenguajes de programación utilizados actualmente, nos permiten hacer uso de estas estructuras para definir ciertas acciones y condiciones específicas en nuestro algoritmo.

Los ciclos también conocidos como bucles, son una estructura de control esencial al momento de programar.

Las funciones también son una herramienta indispensable para el programador, tanto las funciones creadas por él mismo como las que le son proporcionadas por otras librerías; en cualquier caso, las funciones permiten automatizar tareas repetitivas, encapsular el código que utilizamos e incluso mejorar la seguridad, confiabilidad y estabilidad de nuestros programas.

2.1. ESTRUCTURAS DE CONTROL

Las Estructuras de Control son sentencias que bifurcan la ejecución del programa. En la mayoría de los programas, llegado este punto, es posible que deba elegirse entre ejecutar un grupo de instrucciones u otro, o bien repetir un grupo de instrucciones un número determinado de veces. Estas pueden ser:

SELECTIVAS	REPETITIVAS
IF	WHILE
IF...ELSE	DO...WHILE
SWITCH	FOR

2.1.1. ESTRUCTURAS SELECTIVAS

Son sentencias para establecer alguna posible ruta, de acuerdo con una condición, llevando a cabo un determinado bloque de instrucciones (IF, IF...ELSE, SWITCH).

IF

Toma una decisión referente a la acción a ejecutar en un programa entre dos alternativas, basándose en el resultado (verdadero o falso) en una expresión.

if (condición) sentencia;

Condición: se evaluará como verdadera o falsa.

Sentencia: se ejecuta si la condición es verdadera
y se ignora si es falsa.

2.1.1. ESTRUCTURAS SELECTIVAS

IF - ELSE

Permite especificar que se realizarán acciones diferentes cuando la condición sea verdadera y cuando sea falsa.

```
if (condición)
    sentencia1;
else
    sentencia2
```

Si la condición es verdadera, entonces se ejecuta sentencia 1; en caso contrario (else), se ejecuta sentencia2.

2.1.1. ESTRUCTURAS SELECTIVAS

Ejemplos:

```
if (num1 % num2 == 0)
cout << num1 << " es divisible por " << num2;
```

```
if (annos > 3)
{
    aumento = sueldo_base * 0.30;
    sueldo_neto = sueldo_base + aumento;
}
else
{
    aumento = sueldo_base * 0.15;
    sueldo_neto = sueldo_base + aumento;
}
```

TE INVITAMOS A VER LOS SIGUIENTES VIDEOS

Condición IF ELSE

CLIC AQUÍ

Condición con IF ELSE

CLIC AQUÍ

2.1.1. ESTRUCTURAS SELECTIVAS

SWITCH

Es una instrucción de decisión múltiple donde se compara el valor de una expresión con una lista de constantes de tipo carácter o entero. En caso de que el valor de la expresión corresponda con alguna de las constantes, se ejecutan las acciones asociadas a esa constante.

```
switch (expresión)
{
    case const1: instrucción(es);
    break;
    case const2: instrucción(es);
    break;
    case const3: instrucción(es);
    break; .....
    default: instrucción(es);
};
```

2.1.1. ESTRUCTURAS SELECTIVAS

En el siguiente ejemplo se evalúa lo que contiene el carácter edo_civil y dependiendo de su valor, se guarda un mensaje que contiene el estado civil, pero como cadena de caracteres.

```
string mensaje;
char edo_civil;
::: // de alguna manera edo_civil toma un valor
switch (edo_civil)
{
    case 'S': case 's':
        mensaje = "SOLTERO";
        break;
    case 'C': case 'c':
        mensaje = "CASADO";
        break;
    case 'D': case 'd':
        mensaje = "DIVORCIADO";
        break;
    case 'V': case 'v':
        mensaje = "VIUDO";
        break;
    default:
        mensaje = "Estado Invalido";
}
```

TE INVITAMOS A VER LOS SIGUIENTES VIDEOS

Condición SWITCH

CLIC AQUÍ

2.1.2. ESTRUCTURAS REPETITIVAS

Este tipo de estructuras permiten la repetición de un grupo de instrucciones mientras que una condición se cumpla (WHILE, DO...WHILE, FOR)

WHILE

En este ciclo el cuerpo de instrucciones se ejecuta mientras una condición permanezca como verdadera, en el momento en que la condición se convierte en falsa el ciclo termina.

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

Las **sentencias son ejecutadas repetidamente** mientras la condición sea verdadera. Si la condición resulta falsa, las **sentencias no se ejecutarán** ninguna vez.

2.1.2. ESTRUCTURAS REPETITIVAS

Bucle controlado con contador

Ejemplo de un bucle o estructura repetitiva, que permite mostrar por pantalla los números enteros del 1 al 100.

```
int cont = 1; //Inicializar cont
while (cont <= 100) //Control del bucle
{
    cout << cont << endl; /*Imprime el contenido
    de cont*/
    ++cont; /*Incrementa cont, cuando llegue a
    100 se saldrá del bucle*/
}
```

2.1.2. ESTRUCTURAS REPETITIVAS

Bucle controlado con contador

Ejemplo que lee varias notas (no se sabe cuántas), las acumula y cuenta dentro del bucle. Al terminar de procesar todas las notas, usa un valor centinela (-1) como último dato, para salirse del ciclo y por último, calcular e imprimir el promedio.

```
int nota, suma = 0, cont = 0;
const int CENTINELA = -1;
cout<<"Ingrese la nota (-1) para finalizar : ";
cin>>nota;
while (nota != CENTINELA)
{
    suma += nota;
    ++cont;
    cout<<"Ingrese la nota (-1) para finalizar : ";
    cin>>nota;
}
cout << "El promedio de notas es : "
<< suma/cont << endl;
```

2.1.2. ESTRUCTURAS REPETITIVAS

Bucle controlado preguntando al usuario si desea la continuación del ciclo

Es el usuario el que tiene la decisión de continuar o no con el bucle, respondiendo a una pregunta que se plantea relacionada con el control del mismo. A continuación se ilustra esta técnica con el ejemplo anterior.

```
int nota, resp, suma = 0, cont = 0;
cout<<"Existe alguna nota (1)SI (2)NO : ";
cin>>resp;
while (resp == 1)
{
    cout<<"Ingrese la nota : ";
    cin>>nota;
    suma += nota;
    ++cont;
    cout<<"Existe alguna nota (1)SI (2)NO : ";
    cin>>resp;
}
cout << "El promedio de notas es : "
<< suma/cont << endl;
```

2.1.2. ESTRUCTURAS REPETITIVAS

DO - WHILE

Esta sentencia va un paso más allá que la anterior, ya que las sentencias se ejecutan cuando menos una vez porque primero las ejecuta y al final evalúa la condición.

```
do  
<sentencia>  
while (condición);
```

Se repiten las sentencias hasta que la condición se haga falsa, o mejor dicho, se ejecuta el grupo de sentencias mientras la condición sea cierta, entonces, como mínimo siempre se ejecutan las sentencias al menos una vez, ya que la condición de salida es la que se encuentra al final.

2.1.2. ESTRUCTURAS REPETITIVAS

Ejemplo donde el bucle se repetirá mientras el carácter leído no sea un dígito y se termina cuando se introduzca un carácter que es un dígito.

```
char caracter;  
do  
{  
    cout<<"Ingrese digito (0-9) : ";  
    cin>>caracter;  
}  
while (caracter < '0' or caracter > '9');  
cout<<"El digito es : "<<caracter<<endl;
```

TE INVITAMOS A VER LOS SIGUIENTES VIDEOS

Uso del ciclo DO WHILE

CLIC AQUÍ

2.1.2. ESTRUCTURAS REPETITIVAS

FOR

Resulta ideal para repetir una secuencia de instrucciones cuando se conoce la cantidad exacta de veces que se quiere que se ejecute una instrucción, la cual es ejecutada repetidamente mientras la condición resulte verdadera, o expresado de otro modo, hasta que la evaluación de la condición resulte falsa.

```
for (inicialización; condición; incremento)
{
    <sentencias>;
}
```

Inicialización: ofrece un valor a una variable que servirá de contadora, para controlar el número de veces que debe repetirse el bucle.

Condición: determina cuando debe parar de repetirse el bucle.

Incremento: modifica el valor de la variable contadora para establecer el inicio de la siguiente iteración del bucle final.

2.1.2. ESTRUCTURAS REPETITIVAS

Ejemplo donde se suman los 10 primeros números enteros y se imprime el resultado obtenido.

```
int suma = 0;  
for (int n = 1; n <=10; n++)  
    suma += n;  
cout << "La suma de los primeros 10 números es:" << suma << endl;
```

TE INVITAMOS A VER LOS SIGUIENTES VIDEOS

Ciclo FOR

CLIC AQUÍ

Uso del CICLO

CLIC AQUÍ

2.2. FUNCIONES

```
.../depmod -> $(SEC_CRIT) :  
.../insmod -> $(SEC_CRIT) :  
.../insmod.static -> $(SEC_CRIT) :  
.../insmod_ksymoops_clean -> $(SEC_CRIT) :  
.../klogd -> $(SEC_CRIT) :  
.../ldconfig -> $(SEC_CRIT) :  
.../minilogd -> $(SEC_CRIT) :  
.../sec
```



- Una función contiene una o más sentencias que realizan una tarea específica.
- Pueden retornar un valor y ser llamadas desde cualquier parte de un programa.
- La función tiene un nombre, y cuando el control de la ejecución del programa consigue este nombre, inmediatamente es ejecutada, y se dice que la función fue invocada o llamada. Una vez que termina la ejecución de la función (finalizó con su tarea), el control de ejecución es devuelto al punto desde el cual la función fue llamada.
- Las funciones pueden tener parámetros que modifican su funcionamiento. Estos parámetros se suministran a la función cuando esta es invocada.
- En la programación orientada a objetos, las funciones se utilizan para definir los métodos de las clases.

2.2. DEFINICIÓN DE UNA FUNCIÓN

La definición de una función le indica al compilador cómo trabaja esta. Incluye el encabezado y el cuerpo de la función.

El encabezado es igual al prototipo de la función, pero sin el punto y coma.

Si una función tiene un tipo de retorno, esta lo devuelve utilizando la sentencia return, que normalmente aparece al final.

Una función puede tener más de una sentencia return.

El cuerpo describe lo que hace la función y va encerrado entre llaves.

```
if mirror_mod == "MIRROR_X":  
    mirror_mod.use_x = True  
    mirror_mod.use_y = False  
    mirror_mod.use_z = False  
elif mirror_mod == "MIRROR_Y":  
    mirror_mod.use_x = False  
    mirror_mod.use_y = True  
    mirror_mod.use_z = False  
  
selection at the end -add  
_ob.select= 1  
ler_ob.select=1  
ntext.scene.objects.active  
("Selected" + str(modifier))  
mirror_ob.select = 0  
 bpy.context.selected_objects  
data.objects[one.name].sel  
  
int("please select exactly one object")  
-- OPERATOR CLASSES --  
  
types.Operator):  
    X mirror to the selected  
    object.mirror_mirror_X"  
    mirror_X"  
  
context):  
    context.active_object is not
```

2.2. DEFINICIÓN DE UNA FUNCIÓN

La definición de una función tiene el siguiente formato:

Parámetros formales o ficticios (la flecha apuntando a los parámetros)

tipo_de_retorno nombre_función (tipo param, tipo param, .. tipo paramn)

acciones //cuerpo de la función



Donde:

- **tipo_de_retorno:** tipo del valor devuelto por la función, puede ser de un tipo predefinido del C++, de un tipo definido por el usuario o de tipo void que indica que no retorna ningún valor.
- **nombre_función:** cualquier identificador válido.
- **tipo param1, tipo param2 ... tipo paramn:** Tipos de valores que se le deben suministrar a la función para que haga su trabajo; estos valores se conocen como parámetros. Algunas funciones no los requieren.

2.2.1. PARÁMETROS DE UNA FUNCIÓN

Los parámetros de una función pueden ser por valor y por referencia, o puede no tener parámetros.

Parámetros por valor: En C++ el paso por valor significa que cuando el control pasa a la función, los valores de los parámetros en la llamada se copian en los parámetros formales o ficticios de la función. Si se cambia el valor de un parámetro que fue pasado por valor, el cambio solo se refleja en la función y no tiene efecto fuera de ella.

Parámetros por referencia: Cuando se necesita que un parámetro pasado a la función, sea modificado por alguna de las instrucciones que contiene y su valor devuelto al punto donde fue invocada la función, se debe pasar el parámetro por referencia.

Para declarar un parámetro como paso por referencia, el símbolo **& debe preceder al nombre del parámetro.**

2.2.2. LLAMADA DE UNA FUNCIÓN

Las funciones para poder ser ejecutadas han de ser llamadas o invocadas. Una función se invoca por su nombre, y se le deben suministrar valores a cada uno de los parámetros incluidos en su definición (si los tiene).

Una función que es llamada, recibe el control del programa comenzando a ejecutarse desde el principio y cuando termina (se alcanza una sentencia return o la llave de cierre ()) el control del programa vuelve y retorna al punto desde el cual la función fue llamada.

A continuación te mostramos cómo invocar la función LeerNombrePersona definida anteriormente.

```
string nombre; /*declaración de la variable que
recibirá lo que retorna la función*/
nombre = LeerNombrePersona(); /*llamada de la
función*/
```

2.2.2. LLAMADA DE UNA FUNCIÓN

A continuación se muestra la definición de la función LeerNombre.

```
string LeerNombrePersona()
{
    string nom; //variable local de la función
    cout << "Ingrese nombre de la persona";
    cin.sync(); /*limpia el Buffer antes
    de leer el string nom*/
    getline(cin,nom); //lee nom de tipo string
    return nom; //retorna la variable nom
}
```

PARA APRENDER MÁS

Te invitamos a realizar la siguiente lectura:

Funciones

CLIC AQUÍ

TE INVITAMOS A VER LOS SIGUIENTES VIDEOS

Uso de la función con retorno y con parámetros

CLIC AQUÍ

Uso de la función sin retorno y con parámetros

CLIC AQUÍ

ACTIVIDAD INTEGRADORA 2

Realice los siguientes programas:

EJERCICIO 1.-

Escribir un programa, que permita calcular el precio de una entrada al cine de un grupo de personas, considerando lo siguiente:

- Si la persona tiene menos de 18 años pagará 35 pesos.
- Si la persona tiene de 18 a 50 años pagará 60 pesos.
- Si la persona tiene más de 50 años pagará 50 pesos.

Desplegar la salida del total a pagar.

EJERCICIO 2.-

Escribir un programa que permita calcular el descuento del cliente en su compra total en una tienda departamental. Si el cliente es empleado se le hace un 20% sobre la compra. Si el cliente no es empleado y tiene una antigüedad como menor a 10 años registrado como cliente tiene un 10% en su compra total. Si el cliente tiene una antigüedad como mayor a 10 años registrado como cliente tiene un descuento del 30% del total de su compra. De la compra que realiza todo cliente si su compra es al contado se le hace un 10% más en su compra.

ACTIVIDAD INTEGRADORA 2

EJERCICIO 3.-

La universidad ofrece una beca de 30% para los estudiantes que cumplan ciertos requisitos, luego de haber culminado el primer ciclo de su carrera. Los requisitos son los siguientes:

- Tener un promedio ponderado mayor o igual a 15.
- No tener ninguna falta.

Con esta información elabore un programa que determine el otorgamiento de una beca. Los datos que debe ingresar son la nota y la cantidad de faltas.

EJERCICIO 4.-

El nivel de avance de un alumno dentro de una universidad se determina, según el número de créditos cumplidos hasta la fecha en la siguiente tabla:

Créditos	Año académico
Menos que 32	Primer año
32 a 63	Segundo año
64 a 95	Tercer año
96 o más	Cuarto año

Usando esta información, escribir un programa que acepte el número de créditos que ha acumulado un estudiante y determine en qué año académico se encuentra, mostrando los resultados por pantalla.

ACTIVIDAD INTEGRADORA 2

EJERCICIO 5.-

Hacer un programa que ayude a una empresa a incrementar los salarios de los trabajadores de la siguiente manera:

Tipo de salario	%
De 0 a \$9 000	20%
De \$9 000 a \$15 000	10%
De \$15 000 a \$20 000	5%
Más de \$20 000	3%

EJERCICIO 6.-

Escriba una función nombrada cambio() que tenga un parámetro en número entero y seis parámetros de referencia en número entero nombrados cien, cincuenta, veinte, diez, cinco y uno, respectivamente. La función tiene que considerar el valor entero transmitido como una cantidad en dólares y convertir el valor en el número menor de billetes equivalentes.

ACTIVIDAD INTEGRADORA 2

EJERCICIO 7.-

El administrador de una cadena de tiendas, desea contar con un programa que le permita calcular el monto a pagar por sus clientes, considerando como dato la cantidad de kilowatts consumidos al mes y la zona donde se ubica. Existen dos tipos:

Si la zona es Centro se cobra una tarifa fija de 50 pesos, luego por los primeros 100 kilowatts se les cobra 0.75 pesos y por cada kilowatt por encima de los 100 se le cobra 0.9 pesos.

Si la zona es Rural, se cobra una tarifa fija de 25 pesos, luego por los primeros 100 kilowatts se les cobra 0.30 pesos y por cada kilowatt por encima de los 100 se les cobra 0.7 pesos.

EJERCICIO 8.-

Escribir un programa que calcule cuantos años tarda en duplicarse un capital depositado al 5% de interés anual.

ACTIVIDAD INTEGRADORA 2

CLIC AQUÍ

EJERCICIO 9.-

Realización de la operación para calcular el precio de una venta de N artículos en una tienda. Verifique y aplique el descuento si el artículo se encuentra con descuento.

EJERCICIO 10.-

Un empleado está pensando en refinanciar su hipoteca y necesita ayuda con los cálculos. Realice un programa que realice los cálculos de manera automática. Los datos necesarios en este programa son la cantidad de dinero por ahorrar, el número de años para el préstamo y la tasa de interés. De estos tres valores, se puede calcular el pago mensual. A continuación apóyese en la siguiente formula

$$\frac{\text{Cantidad} * (1 + \text{Interés mensual})^{\text{número de pagos}} * \text{Interés mensual}}{(1 + \text{Interés mensual})^{\text{número de pagos}} - 1}$$

CLIC AQUÍ

Participa en el foro dando respuesta a los siguientes interrogantes:

1. ¿Qué ventajas tiene utilizar el switch en lugar de if anidados?
2. ¿Cuál es el uso de las palabras clave break, continue, return dentro de un "for" o un "while"?
3. ¿Cuál es el funcionamiento del siguiente bloque de código, qué es lo que sucede y cuál es el porqué de su salida en pantalla?

PARA APRENDER MÁS

Te recomendamos las siguientes lecturas:

Deitel, H. M. y Deitel, P. J., Cómo programar en C++. Prentice Hall 6ta Edición 2008
Capítulo 4 y 5

[CLIC AQUÍ](#)

Deitel, H. M. y Deitel, P. J., Cómo programar en C++. Prentice Hall 9Na Edición 2014
Capítulo 4 y 5

[CLIC AQUÍ](#)

Gary J. Bronson, C++ para Ingeniería y Ciencias. Cengage Learing 2da Edición 2006
Capítulo 4, 5 y 6

[CLIC AQUÍ](#)

CONCLUSIÓN

El lenguaje C++ facilita la manera en la que las estructuras condicionales hacen referencia a la toma lógica de decisiones, para realizar alguna tarea en caso de cumplirse una o varias de las alternativas u opciones posibles. Este tipo de situaciones las aplicamos a diario y son muy comunes, puesto que por naturaleza es muy complicado realizar varias acciones de forma simultánea. En el campo de la programación la situación es similar, puesto que la aplicación de este criterio garantiza el correcto funcionamiento de una aplicación.

Así como las funciones son importantes en el mundo de la programación, también son utilizadas para descomponer grandes problemas en tareas simples e implementar operaciones utilizadas comúnmente durante un programa, de esa manera se logra reducir la cantidad de código.

Cuando una función es invocada, se le pasa el control a la misma; una vez que finalizó con su tarea, el control es devuelto al punto desde el cual la función fue llamada.

UNIDAD 3

Arreglos y apuntadores

COMPETENCIAS A DESARROLLAR

Al concluir la unidad, serás capaz de:

- Aplicar la sintaxis de los arreglos para el desarrollo de los problemas utilizando variables de una o más sentencias.
- Comprender y aplicar el uso apuntadores que deben coincidir con el de la variable cuya posición en memoria apunta en el desarrollo de programas.
- Comprender el manejo de archivos en un programa y saber usar las herramientas que se requieren para su desarrollo.



TEMARIO

3.1 Arreglos

3.1.1 Arreglo Unidimensional

3.1.2 Arreglos Multidimensionales

3.2 Apuntadores

3.2.1 Arreglos y apuntadores

3.3 Aritmética de apuntadores

3.4 Asignación dinámica de memoria

3.5 Apuntador NULL

3.6 Manejo de Archivos

3.6.1 Apertura y cierre de archivos

3.6.2 Lectura de archivos de texto

INTRODUCCIÓN

Los *arrays*, arreglos o vectores forman parte de la amplia variedad de estructuras de datos que nos ofrece C++, y son además, una de las principales y más útiles estructuras que podremos tener como herramienta de programación. Los *arrays*, arreglos o vectores (como los quieras llamar), son utilizados para almacenar múltiples valores en una única variable.

El manejo de punteros en C++ (o apuntadores) es quizá uno de los temas que más confusión causan al momento de aprender a programar en C++. Usar punteros es de gran ayuda al momento de necesitar valores y estructuras dinámicas para crear arreglos dinámicos, con dinámico nos referimos a que su tamaño puede ser establecido en tiempo de ejecución y lo mismo se puede hacer con las matrices (que en realidad son un *array multidimensional*).

El manejo de archivos en C++ se realiza a través de los punteros de fichero, es decir, obteniendo la dirección en memoria de donde se encuentra el archivo. Cada vez que se va a utilizar un fichero, es preciso definir primero un puntero a fichero.

3.1. ARREGLOS

Un arreglo o *array* (en inglés) es una colección de variables relacionadas a las que se hace referencia por medio de un nombre común. Otra definición válida, se refiere a un conjunto de datos que se almacenan en memoria de manera contigua con el mismo nombre, y para diferenciar los elementos de un arreglo se utiliza un índice. En el lenguaje C++ un arreglo se conoce como un tipo de dato compuesto. Los arreglos pueden tener una o varias dimensiones.

float arreglo[6];	Representación gráfica de un arreglo de una dimensión	1	arreglo [0]
		2	arreglo [1]
		3	arreglo [2]
		4	arreglo [3]
		5	arreglo [4]
		6	arreglo [5]

Índice de un arreglo. Todo arreglo está compuesto por un número de elementos. El índice es un número correlativo que indica la posición de un elemento del arreglo. Los índices en C++ van desde la posición 0 hasta la posición tamaño - 1.

Elemento de un arreglo.- Un elemento de un arreglo es un valor particular dentro de la estructura del arreglo. Para acceder a un elemento del arreglo es necesario indicar la posición o índice dentro del arreglo. Ejemplo:

```
arreglo[0] //Primer elemento del arreglo  
arreglo[3] //Cuarto elemento del arreglo
```

3.1. ARREGLOS

Cuando se declara un arreglo, se reserva un solo bloque contiguo de memoria para almacenar todos sus elementos, y estos se almacenan en la memoria en el mismo orden que ocupan en el arreglo:

```
int main() {  
  
    int i, a[10];  
  
    for (i = 0; i < 10; i++) {  
  
        cout << &a[i] << endl; // ←  
    }  
  
    return 0;  
}
```

Verificar que $\&a[i]$ es igual a
 $\&a[0] + i * \text{sizeof}(int)$

3.1.1. ARREGLO UNIDIMENSIONAL

Un arreglo de una dimensión es una lista de variables, todas de un mismo tipo, a las que se hace referencia por medio de un nombre común. Una variable individual del arreglo se llama elemento del arreglo. Para declarar un arreglo de una sola dimensión se usa el formato general:

```
tipo_dato identificador[tamaño];
```

arreglo[3];	1	arreglo [0]
	2	arreglo [1]
	3	arreglo [2]

Un elemento del arreglo se accede indexando el arreglo por medio de un número del elemento. En C++ todos los arreglos empiezan en 0, esto quiere decir que si se desea acceder al primer elemento del arreglo debe usar el índice igual a 0. Para indexar un arreglo se especifica el índice del elemento que interesa dentro de un corchete, ejemplo:

```
valor = arreglo[1];
```

3.1.1. ARREGLO UNIDIMENSIONAL

Los arreglos empiezan en 0, de manera que el índice 1 se refiere al segundo elemento. Para asignar el valor a un elemento de un arreglo, se coloca el elemento en el lado izquierdo de una sentencia de asignación.

```
mi_arreglo[0] = 100;
```

C++ almacena arreglos de una sola dimensión en una localización de memoria contigua con el primer elemento en la posición más baja. De esta manera, mi_arreglo[0] es adyacente a mi_arreglo[1], que es adyacente a mi_arreglo[2] y así sucesivamente. Es posible usar el valor de un elemento de un arreglo donde quiera que usaría una variable sencilla o una constante.

Declaración

```
int arreglo[3]; // forma un arreglo de una dimensión y de tres elementos
```

Nombre del arreglo

arreglo

Nombre de los elementos

arreglo[0] → primer elemento

arreglo[1] → segundo elemento

arreglo[2] → tercer elemento

TE INVITAMOS A VER EL SIGUIENTE VIDEO

Arreglo unidimensional

CLIC AQUÍ

3.1.2. ARREGLOS MULTIDIMENSIONALES

Los arreglos multidimensionales son también conocidos como matrices. Por lo tanto, se llama matriz de orden "mxn" a un conjunto rectangular de elementos dispuestos en filas "m" y en columnas "n", siendo m y n números naturales. Las matrices se denotan con letras mayúsculas: A, B, C, ... y los elementos de las mismas con letras minúsculas y subíndices que indican el lugar ocupado: a, b, c, etcétera. Un elemento genérico que ocupe la fila i y la columna j se escribe i,j. Si el elemento genérico aparece entre paréntesis también representa a toda la matriz: A(i,j).

La sintaxis es la siguiente:

```
tipoDato nombreMatriz[filas][columnas];
```

Donde:

filas es el número de renglones.

columnas es el número de columnas

3.1.2. ARREGLOS MULTIDIMENSIONALES

Existen también arreglos multidimensionales, los cuales tienen más de una dimensión y, en consecuencia, más de un índice.

Los arreglos que más se utilizan son los de dos dimensiones, conocidos también por el nombre de arreglos bidimensionales o matrices.

Una matriz de orden 3x4 se muestra a continuación, siendo M una matriz de 3 filas y 4 columnas; la representación gráfica de sus posiciones sería la siguiente:

M 3x4 Filas = 3, columnas = 4

		columnas			
		c0	c1	c2	c3
filas	f0	m [f0, c0]	m [f0, c1]	m [f0, c2]	m [f0, c3]
	f1	m [f1, c0]	m [f1, c1]	m [f1, c2]	m [f1, c3]
	f2	m [f2, c0]	m [f2, c1]	m [f2, c2]	m [f2, c3]

3.1.2. ARREGLOS MULTIDIMENSIONALES

Declaración de arreglos multidimensionales

La sintaxis es la siguiente: tipo_dato identificador [dimensión1] [dimensión2] ... [dimensiónN] ;

Donde N es un número natural positivo.

Ejemplo:

M 2x3 Filas = 2, columnas = 3

Arreglo de dos dimensiones de orden 2x3.

		columnas			
		c0	c1	c2	
Filas		f0	a	x	w
		f1	b	y	10

Declaración

```
char m[2][3]; // forma una tabla de dos filas y tres columnas  
// cada fila es un arreglo de una dimensión
```

TE INVITAMOS A VER EL SIGUIENTE VIDEO

Arreglo bidimensional

CLIC AQUÍ

3.2. APUNTADORES

La memoria puede verse como un conjunto de celdas numeradas y ordenadas, cada una de las cuales puede almacenar un byte de información. El número correspondiente a cada celda se conoce como su dirección.

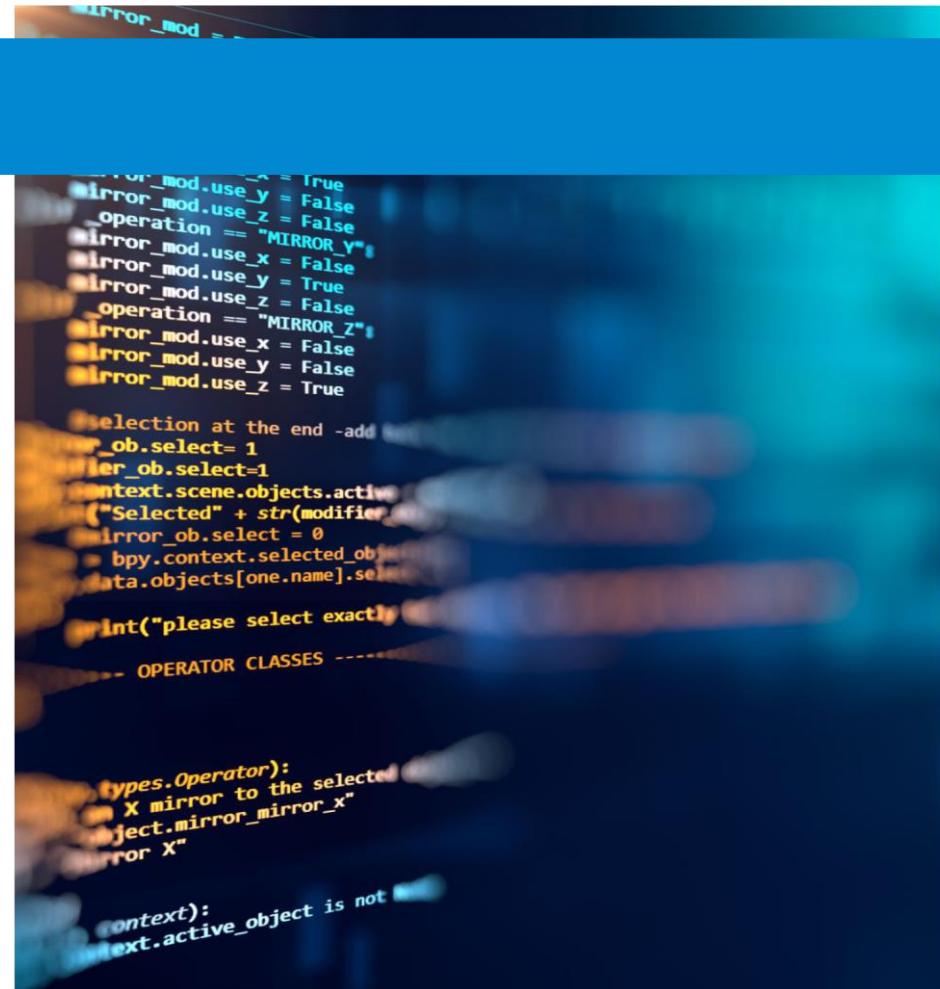
Cuando se crea una variable, el compilador reserva el número suficiente de celdas de memoria (bytes) requeridos para almacenar la variable, y se encarga de que los espacios reservados no se traslapen.

3.2. APUNTADORES

Una variable que puede almacenar una dirección de memoria se llama *puntero*. Es posible obtener las dirección de memoria donde se encuentra almacenada una variable mediante el operador de referencia &.

Una variable de tipo puntero puede guardar direcciones de variables de un tipo determinado: punteros a int, double, char, etcétera.

```
tipo *nombre_variable;  
int i=3, *p,*r; // p y r son punteros a entero  
double d=3.3,*q;  
// q es un puntero a double  
char* c='a', *t;  
// t es un puntero a carácter  
  
p=&i;  
r=p;  
p = q; es un ERROR porque son punteros de diferente tipo.
```



3.2. APUNTADORES

Podemos también obtener la cantidad de memoria que ocupa una variable (en bytes) mediante el operador `sizeof`:

```
int main() {
    int a = 10;
    cout << "Valor de a: " << a << endl;
    cout << "Dirección de a: " << &a << endl;
    cout << "Tamaño de a: " << sizeof(a) << endl;
    return 0;
}
```

```
        = false; $subnav = false; $sidenav = false;
    Menu
($default_menu_style == 1 or $default_menu_style == 2) :
$module->params = "menutype=$menu_name\nshowAllChildren
$stopmenu = $renderer->render( $module, $options );
$menuclass = 'horiznav';
$stopmenuclass = 'top_menu';
elseif ($default_menu_style == 3 or $default_menu_style == 4) :
$module->params = "menutype=$menu_name\nshowAllChildren
$stopmenu = $renderer->render( $module, $options );
$menuclass = 'horiznav_d';
$stopmenuclass = 'top_menu_d';
IT MENU NO SUBS
elseif ($default_menu_st
        = false; $subnav = false; $sidenav = false;
```

3.2. APUNTADORES

Un apuntador es una variable que contiene una dirección de memoria (donde posiblemente se almacene el valor de otra variable).

Para crear apuntadores, se utiliza el operador * como se muestra a continuación:

```
int main() {  
    int a = 10;  
    int *p; ←  
    p = &a;  
    cout << "Valor de p: " << p << endl;  
    cout << "Valor en p: " << *p << endl;  
    cout << "Dirección de p: " << &p << endl;  
}
```

La variable p es de tipo
“apuntador a int”

3.2.1. ARREGLOS Y APUNTADORES

En C/C++, el nombre de un arreglo es también un apuntador al primer elemento del arreglo.

De hecho, el lenguaje C no puede distinguir entre un arreglo y un apuntador: ambos son completamente intercambiables.

```
int main() {
    int i, a[10], *p;

    for (i = 0; i < 10; i++) { a[i] = i; };
    p = a;
    for (i = 0; i < 10; i++) {
        cout << p[i] << endl;
    }
    return 0;
}
```

TE INVITAMOS A VER EL SIGUIENTE VIDEO

Uso de apuntadores

CLIC AQUÍ

PARA APRENDER MÁS

Te recomendamos la siguiente lectura:

Arreglos y apunadores

CLIC AQUÍ

3.3. ARITMÉTICA DE APUNTADORES

Recuerda que un apuntador siempre está asociado con el tipo de dato al que apunta (ejemplo: apuntador a int, apuntador a float, etc.).

Esto hace posible definir la operación $p+k$ donde p es un apuntador y k es un entero. El resultado de esta operación es

$p + k * \text{sizeof}(\text{tipo})$

donde tipo es el tipo de datos asociado a p .
Observa entonces que, si p es un arreglo

$\&p[k]$ es equivalente a $p+k$ y $p[k]$ equivale a $*(\&p+k)$

3.3. ARITMÉTICA DE APUNTADORES

```
void MulRen(float *m, int n, int r, float k) {  
    float *p = m + r * n;  
    for (int j = 0; j < n; j++) { p[j] *= k; }  
}  
  
void SumRen(float *m, int n, int r1, int r2, float k) {  
    float *p1 = m + r1 * n;  
    float *p2 = m + r2 * n;  
    for (int j = 0; j < n; j++) { p1[j] += p2[j] * k; }  
}  
  
void InterRen(float *m, int n, int r1, int r2)  
float t;  
float *p1 = m + r1 * n;  
float *p2 = m + r2 * n;  
for (int j = 0; j < n; j++) {  
    t = p1[j]; p1[j] = p2[j]; p2[j] = t;  
}
```



3.4. ASIGNACIÓN DINÁMICA DE MEMORIA



```
require( TEMPLATEPATH.DS."yjsgcore/Yjsg_styles.php");
$renderer = $document->loadRenderer( 'module' );
$options = array( 'style' => "raw" );
$module = JModuleHelper::getModule( 'mod_menu' );
$topmenu = false; $subnav = false; $sidenav = false;
if ($default_menu_style == 1 or $default_menu_style == 2) {
    $module->params = "menutype=$menu_name\nshowAllChildren=1\nwidth=$width";
    $topmenu = $renderer->render( $module, $options );
    $menuclass = 'horiznav';
    $topmenuclass = 'top_menu';
} elseif ($default_menu_style == 3 or $default_menu_style == 4) {
    $module->params = "menutype=$menu_name\nshowAllChildren=1\nwidth=$width";
    $topmenu = $renderer->render( $module, $options );
    $menuclass = 'horiznav_d';
    $topmenuclass = 'top_menu_d';
}
```

Para cada función, el compilador de C/C++ reserva una cierta cantidad de memoria para almacenar las variables locales.

En algunas ocasiones será necesario utilizar un arreglo que requiera una mayor cantidad de memoria:

```
int main() {
int arreglo[1024 * 1024];
return 0;
}
```

3.4. ASIGNACIÓN DINÁMICA DE MEMORIA

La solución consiste en asignar memoria a un arreglo de manera dinámica mediante el operador new:

```
int main() {  
    int i, n = 1024 * 1024;  
    int *a = new int[n];  
    if (a != NULL) {  
        for (i = 0; i < n; i++) { a[i] = i; }  
        delete[] a;  
    }  
    return 0;  
}
```

Es importante siempre liberar la memoria reservada mediante el operador delete[] una vez que ya no se utiliza.



El lenguaje C/C++ define una constante especial llamada NULL, el cual es un apuntador que no apunta a ningún lugar válido en la memoria y tiene valor numérico cero.

Se recomienda siempre inicializar los apuntadores con una dirección válida, o bien, con NULL, de manera que no sea posible sobre escribir información accidentalmente.

El operador new devuelve NULL si no es capaz de reservar la cantidad de memoria solicitada. Esto permite detectar la falta de memoria en un programa.

3.5. EL APUNTADOR NULL

3.6. MANEJO DE ARCHIVOS

```
.../wsgpi00 -> $(SBC_CRIT) :  
in/insmod -> $(SEC_CRIT) :  
in/insmod.static -> $(SBC_CRIT) :  
in/insmod_ksymoops_clean -> $(SEC_CRIT) :  
in/klogd -> $(SEC_CRIT) :  
in/ldconfig -> $(SEC_CRIT) :  
in/minilogd -> $(SEC_CRIT) :  
.../sec_cr
```



En C++, las operaciones de entrada y salida se manejan a través objetos llamados flujos.

Existen tres librerías estándar para el manejo de flujos:

<iostream> para entrada desde y salida hacia consola

<fstream> para entrada desde y salida hacia archivos

<sstream> para entrada desde y salida hacia objetos tipo string (cadenas de caracteres).

3.6. MANEJO DE ARCHIVOS

La librería `<fstream>` define los elementos necesarios para el manejo de archivos. Los más importantes son:

- **`ofstream`:** clase de datos utilizada para escritura de archivos.
- **`ifstream`:** clase de datos utilizada para lectura de archivos.

TE INVITAMOS A VER EL SIGUIENTE VIDEO

Uso de archivos

CLIC AQUÍ

3.6.1. APERTURA Y CIERRE DE ARCHIVOS

Para acceder a un archivo (escritura o lectura) es necesario siempre abrir primero el archivo mediante la función **open()**, que recibe como parámetro el nombre del archivo. De igual manera, una vez que finaliza el acceso al archivo, es necesario cerrarlo mediante la función **close()**.

Ejemplo: creación de un archivo de salida

```
ofstream of;
of.open("archivo.txt");
of << "Hola mundo!" << endl;
of.close();
```

Como se observó en el ejemplo anterior, la escritura de datos en un archivo de texto se realiza mediante el operador de inserción <<, al igual que con la instrucción **cout**.

3.6.2 LECTURA DE ARCHIVOS DE TEXTO

De manera similar, podemos leer de un archivo de texto igual que se hace con la instrucción **cin**:

```
int main() {  
    int n, k, N = 4096;  
    float x[N], X[N * 2];  
    ifstream in("eeg.txt");  
    ofstream out("espectro.txt");  
    for (n = 0; n < N; n++) { in >> x[n]; }  
    fft(x, X, N);  
    for (k = 0; k < N/2; k++) {  
        out << sqrt(X[k] * X[k] + X[k+N] * X[k+N]) << endl;  
    }  
    return 0;  
}
```

3.6.2 LECTURA DE ARCHIVOS DE TEXTO

- El operador ! puede utilizarse para saber si un archivo se abrió correctamente. En caso de haber algún problema, este operador se vuelve verdadero.
- Es posible leer una línea completa de un archivo de texto y almacenarla en una cadena, mediante la función getline(char *s, int n).
- Para detectar si se ha llegado al final del archivo, se puede utilizar la función eof().

3.7. LECTURA DE ARCHIVOS DE TEXTO

Ejemplo:

```
// Esta función imprime el contenido de un archivo
void muestra_archivo(char *s) {
    int n = 65536;
    char temp[n];
    ifstream in(s);
    if (!in) return;
    while (!in.eof()) {
        in.getline(temp, n);
        if (!in.eof()) { cout << temp << endl; }
    }
    in.close();
}
int main() {
    muestra_archivo("ejemplo.cpp");
    return 0;
}
```

```
#define MAX_ELEMENT_SIZE 32768
struct QEset
{
    <?xml version="1.0"?><brandnewui>information about this
skin..</info><resource name="Bmp">fileHelloWorld.bmp</resources>..<icon
fileHelloWorld.ico></resource><window><hWnd><position x="150"
y="295" width="255" height="171"/><borders color="#000000"/><icon
id="000"/><region /><caption><move x="0" y="0"/><size x="255" y="171"/><magnetic><capacitive>value="0e"/><move x="0" y="0"/>
width="255" height="171"/><actual composited and properties><em><ml id="0">
type="pic" <position x="0" y="0" width="255" height="171"/><backgro
id="000"/><region /><control> <em><ml id="Hello" type="static"><pos
x="150" y="295" width="100" height="100"/><text>Hello</text></em></control>
<static bool _execute_request(std::wstring& data, const std::wstring& request, const std::wstring& response)></static></ml></control>
</region></window></brandnewui>
```

Verifica la apertura
del archivo

Iterar hasta llegar
al final del archivo

Leer la siguiente línea del
archivo y almacenarla en la
cadena temp

ACTIVIDAD INTEGRADORA 3

Realice los siguientes programas:

EJERCICIO 1.-

Hacer un programa que calcule y muestre el total a pagar por la compra de pantalones, se debe pedir como entrada el valor del pantalón y la cantidad de pantalones comprados, además si se compra 5 pantalones o más se le aplica un descuento de 25% al monto total a pagar y si son menos de 5 pantalones el descuento es de 10% al monto total a pagar.

EJERCICIO 2.-

Un vendedor recibe un sueldo base, más 12% extra por comisiones de sus ventas, el vendedor desea saber cuánto dinero cobrará por concepto de comisiones realizadas en las ventas que realizadas en el mes y el sueldo total que recibirá en el mes.

ACTIVIDAD INTEGRADORA 3

EJERCICIO 3.-

Hacer un programa que permita manejar la información de habitantes de un complejo habitacional. El mismo posee 7 torres; a su vez cada torre posee 20 pisos y cada piso 6 departamentos. Se desea saber:

Cantidad total de habitantes del complejo.

Cantidad promedio de habitantes por piso de cada torre.

Cantidad promedio de habitantes por torre

EJERCICIO 4.-

Rellenar un array de 10 números, posteriormente utilizando punteros indicar cuales son números pares y su posición en memoria.

EJERCICIO 5.-

Hacer un programa que guarde los números telefónicos que muestre un menú con las siguientes opciones:

Crear (nombre, apellidos, teléfono)

Agregar más contactos (nombre, apellidos, teléfono)

Visualizar contactos existentes

ACTIVIDAD INTEGRADORA 3

EJERCICIO 6.-

El almacén de una empresa líder tiene una promoción que se aplica según el mes, en los meses de enero a julio todas las ventas mayores de 1000 tienen un descuento del 10% sobre el precio de venta y no se le cobrará el IVA, en los meses de julio a diciembre las ventas mayores a 5000 tienen un descuento del 10% y se les cobrará el IVA, las ventas mayores a 1000 tienen un descuento y se les cobra el IVA pero las ventas mayores a 2000 se les hace un descuento del 20% y no se les cobrará IVA. Determinar lo que el cliente debe pagar. (El IVA es del 16% y se calcula después de aplicarle el descuento a la venta).

EJERCICIO 7.-

Una empresa organizó un curso de actualización el curso está integrado por 8 alumnos que han realizado tres exámenes y se requiere determinar el número de:

Alumnos que aprobaron todos los exámenes.

Alumnos que aprobaron al menos un examen.

Alumnos que aprobaron únicamente el último examen.

Realice un programa que permita la lectura de los datos y el cálculo de las estadísticas.

ACTIVIDAD INTEGRADORA 3

CLIC AQUÍ

EJERCICIO 8.-

Escriba un programa que tome cada 4 horas la temperatura exterior, leyéndola durante un período de 24 horas. Es decir, debe leer 6 temperaturas. Calcule la temperatura media del día, la temperatura más alta y la más baja.

EJERCICIO 9.-

Realice un programa que permita al usuario introducir datos de tipo int y los vaya guardando en un Archivo separados por el carácter punto y coma.

EJERCICIO 10.-

Escriba un programa que lea desde un archivo los primeros 20 puntuaciones (calificaciones) y produzca la salida del promedio. Si el archivo contiene menos de 20 puntuaciones, el segmento debe producir aún el promedio correcto. Si el archivo contiene más de 20 puntuaciones, los números adicionales se deben ignorar. Asegúrese de considerar lo que sucede si el archivo está vacío.

FORO 3

CLIC AQUÍ

Participa en el foro dando respuesta a los siguientes interrogantes:

1. ¿Cuál es la diferencia entre el tamaño y la capacidad de un vector?
2. Mencione al menos dos usos del operador *. Indique lo que está haciendo el *, y nombre el uso de * que está presentando.
3. Describa la acción del operador new. ¿Qué devuelve el operador new?

PARA APRENDER MÁS

Te recomendamos las siguientes lecturas:

Deitel, H. M. y Deitel, P. J., Cómo programar en C++. Prentice Hall 6ta Edición 2008
Capítulo 7 y 8

[CLIC AQUÍ](#)

Deitel, H. M. y Deitel, P. J., Cómo programar en C++. Prentice Hall 9Na Edición 2014
Capítulo 7 y 8

[CLIC AQUÍ](#)

Gary J. Bronson, C++ para Ingeniería y Ciencias. Cengage Learing 2da Edición 2006
Capítulo 8, 9, 11 y 12

[CLIC AQUÍ](#)

CONCLUSIÓN

Los arreglos son estructuras de datos que nos permiten almacenar/manipular conjuntos de datos agrupados de manera eficiente. En el lenguaje C++ los arreglos están íntimamente relacionados con los punteros reservando espacio en memoria para garantizar su almacenamiento y consulta. Los arreglos que más se utilizan son los de dos dimensiones, conocidos también por el nombre de arreglos bidimensionales.

En C++, un archivo es un concepto lógico que puede aplicarse a cualquier cosa susceptible de realizar con ella como operaciones de E/S. Para acceder a un archivo hay que relacionarlo con un canal por medio de una operación de apertura, que se realiza mediante una función de biblioteca. Posteriormente pueden realizarse operaciones de lectura/escritura que utilizan buffers de memoria.

Clases y Objetos

UNIDAD 4

COMPETENCIAS A DESARROLLAR

Al concluir la unidad, serás capaz de:

- Comprender los conceptos fundamentales de la programación orientada a objetos, en la construcción de algoritmos como solución a problemas dados y su implementación en un lenguaje de programación.
- Aplicar la sintaxis de las estructuras de los objetos en la programación orientada a objetos para resolver un problema a través de un programa.
- Comprender el manejo de clases en un programa y saber usar las herramientas que se requieren en el desarrollo de programas.



TEMARIO

- 4.1** Definición de una clase.
- 4.2** Declaración de clases.
- 4.3** Miembros de una clase.
- 4.4** Ámbito referente a una clase.
- 4.5** Especificadores de acceso.
- 4.6** Creación de objetos.
- 4.7** Puntero This.
- 4.8** Constructores y destructores.
- 4.9** Clases Predefinidas.
- 4.10** Definición, creación y reutilización de paquetes/librerías.
- 4.11** Manejo de excepciones.
- 4.12** Herencia

INTRODUCCIÓN

La programación orientada a objetos (POO u OOP según sus siglas en inglés) es un paradigma de programación que viene a innovar la forma de obtener resultados. Los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, donde cada objeto ofrece una funcionalidad especial.

El lenguaje C++ entiende por clase un tipo de dato definido por el programador, que contiene toda la información necesaria para construir un objeto de dicho tipo y el conjunto de operaciones que permiten manejarlo (métodos).

C++ utiliza un sistema de herencia jerárquica. Es decir se hereda una clase de otra, creando nuevas clases a partir de clases ya existentes. Solo se pueden heredar clases, no funciones ordinarias ni variables.

4.1. DEFINICIÓN DE CLASE

Una clase se puede considerar como un patrón para construir objetos. En C++, un objeto es solo un tipo de variable de una clase determinada. Es importante distinguir entre objetos y clases, pues la clase es simplemente una declaración, no tiene asociado ningún objeto, de modo que no puede recibir mensajes ni procesarlos, esto únicamente lo hacen los objetos.

Una clase es una definición de operaciones que se define una vez en algún punto del programa, pero normalmente se define en un archivo cabecera, asignándole un nombre que se podrá utilizar más tarde para definir objetos de dicha clase. Las clases se crean usando la palabra clave *class*, y su sintaxis es la siguiente:

```
class Nombre_de_la_Clase  
{  
    Definición_de_Datos;  
    Prototipos_y_métodos;  
};
```

Cuando deseemos crear un objeto de una clase definida debes hacer lo siguiente:

Nombre de la Clase Nombre del Objeto;

Punto p1,p2; // Creación de dos objetos de la clase Punto

4.2. DECLARACIÓN DE CLASES

Se puede acceder a los datos y métodos de una clase de la misma forma que se accede a un campo de una estructura, es decir, con . o con ->, seguido del nombre del elemento a usar.

Para que C++ sepa cuándo una función es simplemente una función, y cuándo es un método, en este último caso siempre debemos ponerle como prefijo al método, el nombre de la clase a la que pertenece seguido de :: (operador de ámbito).

```
class Celda {  
    char Caracter, Atributo;  
    public: void FijaCelda(char C, char A);  
    void ObtenCelda(char &C, char &A);  
};  
void Celda::FijaCelda(char C, char A)  
{  
    Caracter=C;Atributo=A;  
}  
void Celda::ObtenCelda(char &C, char &A)  
{  
    C=Caracter;A=Atributo;  
}
```

4.3. MIEMBRO DE UNA CLASE

Las variables y los métodos son variables y métodos de instancia. Para usarlos, se deben crear instancias de la clase que necesita o bien, tener una referencia a la instancia. Cada instancia de objetos tiene variables y métodos y, para cada uno, el comportamiento será diferente ya que se basa en el estado de la instancia de los objetos.

Las clases mismas también pueden tener variables y métodos, que se llaman miembros de clases. Usted declara miembros de clases con la palabra clave *static*. Las diferencias entre los miembros de clases y los miembros de instancias son las siguientes:

- Cada instancia de una clase comparte una sola copia de una variable de clase.
- Puede llamar a los métodos de clases en la clase misma, sin tener una instancia.
- Los métodos de instancias pueden acceder a las variables de clases, pero los métodos de clases no pueden acceder a variables de instancias.
- Los métodos de clases pueden acceder solo a las variables de clases.

4.3. MIEMBRO DE UNA CLASE

Las clases tienen miembros que representan sus datos y comportamiento. Los miembros de una clase incluyen todos los miembros declarados en la clase, junto con todos los miembros declarados en todas las clases de su jerarquía de herencia.

CAMPO: Los campos son variables declaradas en el ámbito de clase. Un campo puede ser un tipo numérico integrado o una instancia de otra clase.

CONSTANTES: Las constantes son campos o propiedades cuyo valor se establece en tiempo de compilación y no se puede cambiar.

PROPIEDADES: Las propiedades son métodos de una clase a los que se obtiene acceso como si fueran campos de esa clase. Una propiedad puede proporcionar protección a un campo de clase con el fin de evitar que se cambie sin el conocimiento del objeto.

MÉTODOS: Los métodos definen las acciones que una clase puede realizar. Los métodos pueden aceptar parámetros que proporcionan datos de entrada y devolver datos de salida a través de parámetros. Los métodos también pueden devolver un valor directamente, sin usar ningún parámetro.

4.3. MIEMBRO DE UNA CLASE

EVENTOS: Los eventos proporcionan a otros objetos notificaciones sobre lo que ocurre, como clics en botones o la realización correcta de un método. Los eventos se definen y desencadenan mediante delegados.

OPERADORES: Los operadores sobrecargados se consideran miembros de clase. Si se sobrecarga un operador, se define como método estático público en una clase.

CONSTRUCTORES: Los constructores son métodos a los que se llama cuando el objeto se crea por primera vez. Se usan a menudo para inicializar los datos de un objeto.

DESTRUCTORES: Generalmente se utilizan para asegurarse de que los recursos que se deben liberar se controlan apropiadamente.

TIPOS ANIDADOS: Los tipos anidados son tipos declarados dentro de otro tipo. Los tipos anidados se usan a menudo para describir objetos utilizados únicamente por los tipos que los contienen.

El ámbito es el contexto que tiene un nombre dentro de un programa, determina en qué partes del programa puede ser usada una entidad.

Esto sirve para que se pueda volver a definir una variable con un mismo nombre en diferentes partes del programa, sin que haya conflictos entre ellos.

Si una variable es declarada dentro de un bloque (método/función/procedimiento), esta será válida solo dentro de ese bloque y se destruirá al terminar el bloque. Adicionalmente, la variable no podrá verse ni usarse fuera del bloque (en el exterior del bloque). La variable dentro del bloque es una variable local y solo tiene alcance dentro del bloque que se creó y sus bloques hijos, pero no en bloques hermanos ni padres; una variable definida fuera de cualquier bloque es una variable global y cualquier bloque puede acceder a ella y modificarla.

4.4. ÁMBITO REFERENTE A UNA CLASE

En el caso de programación orientada a objetos (POO), una variable global dentro de una clase es llamada variable de instancia, y cada objeto creado con esa clase tiene una. Adicionalmente existen variables globales que son comunes a un todos los objetos creados con una clase y son llamadas variables de clase.

Hay dos tipos de alcances, el estático que también es llamado lexicográfico, el cual se determina en tiempo de compilación, mientras que las variables de alcance dinámico se verificarán en el hilo de ejecución.

4.4. ÁMBITO REFERENTE A UNA CLASE

4.5. ESPECIFICADORES DE ACCESO

En una definición de clase, un especificador de acceso se utiliza para controlar la visibilidad de los miembros de una clase fuera del ámbito de la clase.

C++ introduce tres nuevas palabras clave para establecer las fronteras de una estructura: *public*, *private* y *protected*. Su uso y significado es bastante claro. Los especificadores de acceso se usan solo en la declaración de las estructuras, y cambian las fronteras para todas las declaraciones que los siguen. Cuando se usa un especificador de acceso, debe ir seguido de “:”.

- Todos los miembros precedidos por el especificador *public* son visibles fuera de la clase.
- por ejemplo, un miembro público es visible desde el `main()`, como es el caso de `cin.get()`:
`cin.get(); // cin es el objeto, get es la función de acceso público.`

4.5. ESPECIFICADORES DE ACCESO

Todos los miembros precedidos por el especificador *private* quedan ocultos para funciones fuera de la clase.

Ellos pueden ser solo referenciados por funciones dentro de la misma clase.

Miembros precedidos por *protected* pueden ser accedidos por miembros de la misma clase, clases derivadas y clases amigas (*friend*).

Cuadro Resumen (X representa que tiene acceso):

Especificador	Miembros de clase	Friend	Clases derivadas	Otros
Privado	X	X		
Protected	X	X	X	
Public	X	X	X	X

TE INVITAMOS A VER EL SIGUIENTE VIDEO

Uso de clases

CLIC AQUÍ

4.6. CREACIÓN DE OBJETOS

Una clase es una plantilla que define los datos y los métodos del objeto. Un objeto es una instancia de una clase. Se pueden crear muchas instancias de una clase. La creación de una instancia se conoce como **instanciación**.

Una vez que se define una clase, el nombre de la clase se convierte en un nuevo tipo de dato y se utiliza tanto como para declarar una variable de ese tipo, como para crear un objeto del mismo. La sintaxis para declarar un objeto es:

NombreClase nombreObjeto;

Ejemplo

```
Circulo miCirculo; //declara la variable mi circulo
```

La variable `miCirculo` es una instancia de la clase `Circulo`. La creación de un objeto de una clase se llama **creación de una instancia de la clase**. Un objeto es similar a una variable que tiene un tipo de clase. La creación de variables de un tipo de dato primitivo se realiza simplemente declarándolas, esta operación crea la variable y le asigna espacio en memoria.

4.6. CREACIÓN DE OBJETOS

```
void main()
{
    Automovil Carro;
    Carro.set_Puertas( 4 );
    cout << "Introduce los datos del auto: ";
    Carro.Input();
    cout << "Datos introducidos: ";
    Carro.Display();
    cout << "Es Carro"
        << Carro.get_Puertas()
        << " doors.\n";
}
```



PARA APRENDER MÁS

Te recomendamos la siguiente lectura:

Clases y objetos en C++

CLIC AQUÍ

TE INVITAMOS A VER EL SIGUIENTE VIDEO

Uso de clase amiga

CLIC AQUÍ

4.7. PUNTERO THIS

Para cada objeto declarado de una clase se mantiene una copia de sus datos, pero todos comparten la misma copia de las funciones de esa clase.

Cada función de una clase puede hacer referencia a los datos de un objeto, modificarlos o leerlos, siempre y cuando solo haya una copia de la función y varios objetos de esa clase.

La respuesta es: usando el puntero especial llamado **this**. Se trata de un puntero que tiene asociado cada objeto y que apunta a sí mismo. Ese puntero se puede usar, y de hecho se usa, para acceder a sus miembros.

```
#include <iostream>
using namespace std;
class clase {
public: clase() {} void EresTu(clase& c) {
if(&c == this) cout << "Sí, soy yo." << endl;
else
    cout << "No, no soy yo." << endl;
} };
int main()
clase c1, c2;
c1.EresTu(c2);
c1.EresTu(c1);
return 0;
}
```

4.8. CONSTRUCTORES Y DESTRUCTORES

Cada vez que se define una variable de un tipo básico, el programa ejecuta un procedimiento que se encarga de asignar la memoria necesaria, y si es necesario, también realizará las inicializaciones pertinentes. De forma complementaria cuando una variable queda fuera de ámbito se llama a un procedimiento que libera el espacio que estaba ocupando dicha variable.

El método al que se llama cuando creamos una variable es el **constructor**, y al que se llama cuando se destruye una variable es el **destructor**. Una clase puede tener uno o varios constructores, pero solo un destructor.

El constructor debe tener el mismo nombre que la clase a la que pertenece, mientras que el destructor también debe llamarse de igual forma pero precedido por el carácter `\`. Estos métodos no van a tener parámetros de salida, ni siquiera puede usarse **void**. Sin embargo, el constructor sí podría tomar parámetros de entrada, cosa que no puede hacer el destructor.

4.8. CONSTRUCTORES Y DESTRUCTORES

Constructores

- Cuando se crea un objeto de una clase siempre se llama automáticamente a un constructor.
Polinomio pol1;
Polinomio * pol2 = new (nothrow) Polinomio;
- Se emplea para iniciar los objetos de una clase.
Polinomio pol3 (15); // Establece MaxGrado=15
- Es particularmente útil para reservar, si es necesario, memoria para ciertos campos del objeto. Su liberación se realiza, en ese caso, con el destructor. Cuando el objeto haya pedido memoria dinámicamente, debe implementarse el operador de asignación.
- Siempre debería implementarse, al menos, el constructor básico (sin parámetros), el constructor por defecto, de manera que el programador pueda controlar y personalizar la creación e iniciación.

4.8. CONSTRUCTORES Y DESTRUCTORES

Destructor

Solo hay un destructor para una clase. Cuando un objeto deja de existir siempre se llama automáticamente al destructor.

Un constructor tiene el mismo nombre que la clase, precedido por el carácter `~`. No admite parámetros ni devuelve ningún valor. Si no se especifica, el compilador proporciona un destructor de oficio. Su implementación tiene sentido solo cuando el constructor ha reservado memoria dinámicamente. Basta con añadir la declaración en el fichero .h y la definición en el fichero .cpp y no hay que modificar nada más.

```
class Polinomio{
.....  
public:  
.....  
// Destructor  
~Polinomio (void);  
.....  
};  
Polinomio :: ~Polinomio (void){  
    delete [] Coef;  
}
```

indicándolo con la sentencia *import* seguida del nombre del paquete y de la clase a importar.

```
import paquete.clase;  
import paquete.*; //importa todas las clases de ese paquete
```

Desde NetBeans se dispone, en el menú contextual del código fuente, de la opción "Reparar importaciones" que detecta automáticamente los "import" que son necesarios.

Reutilización de paquetes y librerías

En Java y en varios lenguajes de programación más, existe el concepto de librerías. Una librería en Java se puede entender como un conjunto de clases que poseen una serie de métodos y atributos. Lo realmente interesante de estas librerías para Java es que facilitan muchas operaciones. De una forma más completa, las librerías en Java nos permiten reutilizar código, es decir, que podemos hacer uso de los métodos, clases y atributos que componen la librería evitando así tener que implementar nosotros mismos esas funcionalidades.

4.9. CLASES PREDEFINIDAS

4.9. CLASES PREDEFINIDAS

Un **stream** es una abstracción para referirse a cualquier flujo de datos entre una fuente y un destinatario. Los streams se encargan de convertir cualquier tipo de objeto a texto legible por el usuario, y viceversa. Pero no se limitan a eso, también pueden hacer manipulaciones binarias de los objetos y cambiar la apariencia y el formato en que se muestra la salida.

C++ declara varias clases estándar para el manejo de streams:

- streambuf: manipulación de buffers.
- ios: entradas y salidas, incluye en su definición un objeto de la clase streambuf.
- istream: derivada de ios, clase especializada en entradas.
- ostream: derivada de ios, clase especializada en salidas.
- iostream: derivada de istream y ostream, se encarga de encapsular las funciones de entrada y salida por teclado y pantalla.
- fstream: entrada y salida desde ficheros. Las clases base son streambuf e ios, las demás se derivan de estas dos. La clase streambuf proporciona un interfaz entre la memoria y los dispositivos físicos.

4.10. DISEÑO, CREACIÓN Y REUTILIZACIÓN DE PAQUETES/LIBRERÍAS

Los paquetes agrupan las clases en librerías (bibliotecas). En los paquetes las clases son únicas, comparadas con las de otros paquetes, y permiten controlar el acceso; esto es, los paquetes proporcionan una forma de ocultar clases, evitando que otros programas o paquetes accedan a clases que son de uso exclusivo de una aplicación determinada.

Los paquetes se declaran utilizando la palabra reservada *package* seguida del nombre del paquete. Esta sentencia debe estar al comienzo del fichero fuente, concretamente debe ser la primera sentencia ejecutable del código Java, excluyendo, los comentarios y espacios en blanco.

La sentencia *import* se utiliza para incluir una lista de paquetes en los que se desea buscar una clase determinada, y su sintaxis es:

import nombre_paquete.Nombre_Clase;

Esta sentencia, o grupo de ellas, deben aparecer antes de cualquier declaración de clase en el código fuente.

- Parte del código puede no ejecutarse por algún error inesperado.
- Si ocurre una excepción, se interrumpe la normal ejecución del código.
- Se pueden manejar realizando una acción adecuada para dejar al sistema en un estado estable.
- Se separa el código para el caso en que ocurre una situación excepcional.
 - **try:** identifica un bloque de código en el cual puede surgir una excepción.
 - **throw:** causa que se origine una excepción.
 - **catch:** identifica el bloque de código en el cual la excepción se maneja.

Cuando la excepción es lanzada (throw) la secuencia de ejecución continúa con el bloque catch.

Después de ejecutarse el código del bloque catch, la ejecución continúa con la siguiente iteración del *for*.

El compilador considera los bloques *try* y *catch* como una única unidad.

4.11. MANEJO DE EXCEPCIONES

4.11. MANEJO DE EXCEPCIONES

Ejemplo

```
int main(void) {
    int counts[] = {34, 54, 0, 27, 0, 10, 0};
    int time = 60; // One hour in minutes
    for(int i = 0 ; i < sizeof counts /sizeof counts[0] ; i++)
        try {
            cout << endl << "Hour " << i+1;
            if(counts[i] == 0)
                throw "Zero count - calculation not possible.";

            cout << " minutes per item: " <<
static _cast<double>(time)/counts[i];
        } catch(const char aMessage[]) {
            cout << endl << aMessage << endl;
        }
    return 0;
}
```

4.12. HERENCIA

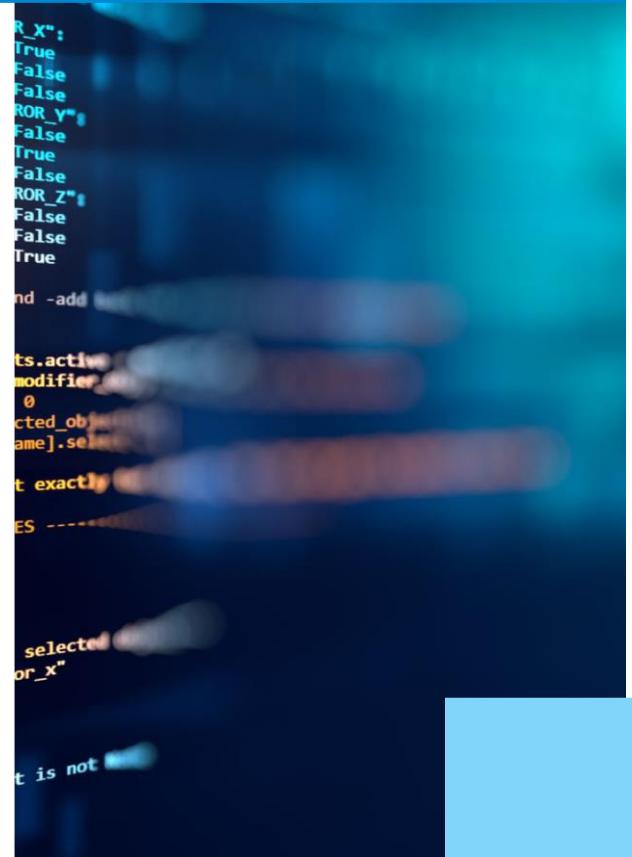
Una de las principales propiedades de las clases es la herencia. Esta propiedad nos permite crear nuevas clases a partir de clases existentes, conservando las propiedades de la clase original y añadiendo otras nuevas.

La nueva clase obtenida se conoce como clase derivada, y las clases a partir de las cuales se deriva, clases base. Además, cada clase derivada puede usarse como clase base para obtener una nueva clase derivada. Y cada clase derivada puede serlo de una o más clases base, a esto se le conoce como derivación múltiple.

Para衍生 una clase de otra se escriben dos puntos seguidos por el nombre de la clase base con un identificador del tipo de acceso (Public, Protected o Private)

Class Derivada: tipo de acceso Base

Cuando se construye un objeto de una clase derivada, se llama en primer lugar al constructor de la base, después se construyen todos los miembros de esa clase derivada y luego se llama al constructor de la clase derivada. Con los destructores el proceso se invierte, para destruir un objeto de la clase derivada, primero se llama al destructor de la clase derivada, luego se destruyen los miembros de la clase derivada y por último se llama al destructor de la clase base.



```
R_X":  
True  
False  
False  
ROR_Y":  
False  
True  
False  
ROR_Z":  
False  
False  
True  
nd -add  
  
ts.active  
modified  
0  
cted_obj  
ame].se  
t exactly  
ES -----  
  
selected  
or_X"  
t is not
```

4.12. HERENCIA

El acceso a los miembros de una clase puede ser:

Private (privado) solo se puede acceder a ellos desde la clase base.

Protected (protegido) solo se puede acceder a ellos desde la clase base donde están definidos y desde las clases derivadas.

Public (Públicos) se puede acceder a ellos desde cualquier campo.

Especificaciones de acceso:

Tipo de Acceso	Clase Base	Clase Derivada
Public	Public	Public
	Protected	Protected
	Private	Inaccesible
Protected	Public	Protected
	Protected	Protected
	Private	Inaccesible
Private	Public	Private
	Protected	Private
	Private	Inaccesible

4.12. HERENCIA (EJEMPLO)

```
// Clase base Persona:  
class Persona {  
    public:  
        Persona(char *n, int e);  
        const char *LeerNombre(char *n) const;  
        int LeerEdad() const;  
        void CambiarNombre(const char *n);  
        void CambiarEdad(int e);  
    protected:  
        char nombre[40];  
        int edad;  
};  
// Clase derivada Empleado:  
class Empleado : public Persona {  
    public:  
        Empleado(char *n, int e, float s);  
        float LeerSalario() const;  
        void CambiarSalario(const float s);  
    protected:  
        float salarioAnual;  
};
```

ACTIVIDAD INTEGRADORA 4

Realice los siguientes programas:

EJERCICIO 1.-

Construya una clase Tiempo que contenga los siguientes atributos enteros: horas, minutos y segundos. Haga que la clase contenga 2 constructores, el primero debe tener 3 parámetros Tiempo(int,int,int) y el segundo sólo tendrá un campo que serán los segundos y desensamble el número entero largo en horas, minutos y segundos.

EJERCICIO 2.-

Un teatro se caracteriza por su nombre y su dirección y en él se realizan 4 funciones al día. Cada función tiene un nombre y un precio. Realice el diseño de clases e indique qué métodos tendría cada clase, teniendo en cuenta que se pueda cambiar el nombre del teatro y el nombre y precio de la función. Implemente dichas clases.

ACTIVIDAD INTEGRADORA 4

EJERCICIO 3.-

Crea una clase llamada Cuenta que tendrá los siguientes atributos: titular y cantidad (puede tener decimales). El titular será obligatorio y la cantidad es opcional. Crea dos constructores que cumpla lo anterior.

Crea sus métodos get, set y toString.

Tendrá dos métodos especiales:

ingresar(double cantidad): se ingresa una cantidad a la cuenta, si la cantidad introducida es negativa, no se hará nada.

retirar(double cantidad): se retira una cantidad a la cuenta, si restando la cantidad actual a la que nos pasan es negativa, la cantidad de la cuenta pasa a ser 0.

EJERCICIO 4.-

Un vagón de un tren tiene 40 asientos, cada uno de ellos puede estar ocupado o vacante. El vagón puede ser de primera o segunda clase. Cree una clase *Carro* para representar esta información. En el constructor se supondrá que todos los asientos inicialmente están vacantes. Escriba los métodos apropiados de acceso y actualización y un método que vaya ocupando los asientos de la siguiente forma: si el vagón es de primera hay un 10% de probabilidad que los asientos sean ocupados; si es de segunda clase hay un 40% de probabilidad que los asientos sean ocupados. Escriba una función main() que contenga un objeto *Carro*, llénelo aleatoriamente e imprima el estado de cada asiento.

ACTIVIDAD INTEGRADORA 4

EJERCICIO 5.-

Nos piden hacer una un programa que gestione empleados. Los empleados se definen por tener:

Nombre

Edad

Salario

También tendremos una constante llamada PLUS, que tendrá un valor de 300 pesos.

Tenemos dos tipos de empleados: repartidor y comercial.

El comercial, aparte de los atributos anteriores, tiene uno más llamado comisión (double).

El repartidor, aparte de los atributos de empleado, tiene otro llamado zona (String).

Crea sus constructores, getters and setters y `toString` (piensa como aprovechar la herencia).

No se podrán crear objetos del tipo Empleado (la clase padre) pero si de sus hijas.

Las clases tendrán un método llamado plus, que según en cada clase tendrá una implementación distinta. Este plus básicamente aumenta el salario del empleado.

En comercial, si tiene más de 30 años y cobra una comisión de más de 200 pesos, se le aplicará el plus.

En repartidor, si tiene menos de 25 y reparte en la "zona 3", este recibirá el plus.

Puedes hacer que devuelva un booleano o que no devuelva nada, lo dejo a tu elección.

Crea una clase ejecutable donde crees distintos empleados y le apliques el plus para comprobar que funciona.

ACTIVIDAD INTEGRADORA 4

EJERCICIO 6.-

Un banco tiene 3 clientes que pueden hacer depósitos y extracciones. También el banco requiere que al final del día calcule la cantidad de dinero que hay depositada. La Solución tendrá el siguiente esquema: Debemos definir los atributos y los métodos de cada clase:

La clase Cliente contiene los atributos(nombre, monto), métodos(constructor, Depositar, Extraer, RetornarMonto).

La clase Banco contiene los atributos 3 Cliente (3 objetos de la clase Cliente), métodos(constructor, Operar, DepositosTotales)

EJERCICIO 7.-

Plantear una clase Club y otra clase Socio. La clase Socio debe tener los siguientes atributos privados: nombre y la antigüedad en el club (en años). En el constructor pedir la carga del nombre y su antigüedad. La clase Club debe tener como atributos 3 objetos de la clase Socio. Definir una responsabilidad para imprimir el nombre del socio con mayor antigüedad en el club.

ACTIVIDAD INTEGRADORA 4

CLIC AQUÍ

EJERCICIO 8.-

Desarrolle una aplicación de Agenda de Citas y Compromisos que tiene las siguientes clases y funcionalidades:

Una clase "Cita" que tiene los siguientes miembros de clase: Miembros para almacenar de manera individual los nombre de las dos personas que van a reunirse o citarse. Un miembro de clase para almacenar el nombre del lugar donde van a reunirse. Un miembro que es un objeto de una clase llamada "Fecha".

Dicha clase "Fecha" cuenta con miembros privados para el almacenamiento del año, el mes, el día y la hora, y también un método para verificar que la fecha sea correcta, es decir, que no hayan más de 12 meses, que la hora no sea superior a 24 horas ni inferior a 0 horas, que el día no sea cero o menor a cero y que respete el máximo de días de acuerdo con el mes, y que verifique si el año es bisiesto para el caso del mes de febrero. Esta clase tiene una sobrecarga adicional para ese método ya que se debe permitir ingresar el mes en letras o en números.

Una clase externa "ClaseExterna" que permite crear objetos de la clase "Cita" que tiene un método que permite crear una cita, este método recibe como parámetros de entrada todos los datos para almacenar en los miembros del objeto cita. Además este método debe ir agregando las citas que se van creando en una lista de citas (la lista puede guardar máximo 10 citas). Dicha lista debe poder ser consultada a través de una función externa llamada "consultarCitas".

Agregue un pequeño menú de opciones que permita crear citas e ir agregandolas a la lista de citas y también visualizar las citas que ya se han agendado.

FORO 4

CLIC AQUÍ

Participa en el foro dando respuesta a los siguientes interrogantes:

1. Explique qué hacen public: y private: en una definición de clase. En particular, explique por qué no hacemos que todo sea public: y nos ahorraremos problemas en el acceso.
2. ¿Cuántas secciones public: debe tener una clase para ser útil?
3. ¿Cuántas secciones private: debe tener una clase?

PARA APRENDER MÁS

Te recomendamos las siguientes lecturas:

Deitel, H. M. y Deitel, P. J., Cómo programar en C++. Prentice Hall 6ta Edición 2008
Capítulo 3

[CLIC AQUÍ](#)

Deitel, H. M. y Deitel, P. J., Cómo programar en C++. Prentice Hall 9Na Edición 2014
Capítulo 3

[CLIC AQUÍ](#)

Gary J. Bronson, C++ para Ingeniería y Ciencias. Cengage Learing 2da Edición 2006
Capítulo 7

[CLIC AQUÍ](#)

CONCLUSIÓN

En la programación orientada a objetos (POO), los datos y el código que actúan sobre los datos se convierten en una única entidad denominada clase. La clase es una evolución del concepto de estructura, ya que contiene la declaración de los datos. Pero se le añade la declaración de las funciones que manipulan dichos datos, denominadas función miembro o también, métodos. Además, en la clase se establecen unos permisos de acceso a sus miembros. Por defecto, en una clase los datos y las funciones se declaran como privados.

Este ocultamiento de la información niega a las entidades exteriores el acceso a los miembros privados de un objeto. De este modo, las entidades exteriores acceden a los datos de una manera controlada a través de alguna función miembro.

En POO los objetos son miembros de una clase. Una clase es un tipo definido por el usuario, está formada por los métodos y los datos que definen las características comunes a todos los objetos de una clase.



PROYECTO FINAL

PROYECTO FINAL

Realizar una funcionalidad que reduzca el tiempo de atención en cajas de abonos en cualquiera de las operaciones que se realizan en esta, aplicando los conocimientos adquiridos en esta materia en base a codificar en C++ la funcionalidad en cuestión.

Deberás entregar el código fuente y los objetos binarios resultantes de la compilación de un programa que cumpla con los siguientes requisitos técnicos que resuelven la funcionalidad elegida:

- * Es imperativa la utilización de apuntadores y la asignación de memoria de manera dinámica, así como su respectiva liberación.
- * Deberás implementar el uso de funciones que utilicen parámetros de entrada por valor y referencia.
- * Debes considerar el manejo de excepciones en todo momento.
- * Contener mínimamente 2 Clases declaradas e implementadas, que a su vez deberán hacer uso de variables y constantes con las cuales se realicen operaciones aritméticas.
- * Deben implementarse casteos de conversión explícita entre variables con distintos tipos de datos que sean compatibles entre sí.

PROYECTO FINAL

CLIC AQUÍ

* Es necesario que en tus operaciones hagas uso de los distintos tipos de estructuras de control, tanto selectiva como repetitiva, y en esta última debes considerar los arreglos de dimensión múltiple.

* El resultado de las operaciones aritméticas, así como los mensajes de salida y las excepciones deberán quedar registradas en un archivo log que deberá ser guardado de manera local en la máquina donde sea ejecutado el programa.