UNIVERSIDAD NACIONAL DEL LITORAL
FACULTAD DE INGENIERÍA Y CIENCIAS HÍDRICAS

Tecnicatura en diseño
y programación de Videojuegos

UNLVIRTUAL

Introducción a la programación

Unidad Temática Número 2

Introducción al Lenguaje ANSI/ISO C++

Objetivo: Acercar al Alumno a los Conceptos de C++

Temas: Compiladores e Intérpretes, Código Fuente, Código Objeto, Enlazador, IDE, RAD, Estructura de un Programa C++, Elementos de un Programa C++, Indentificadores, Literales (constantes), Tipos de Datos Estándar, Notación y Definición de Constantes, Declaración e Inicialización de Variable, Entrada y Salida.

Introducción

Retomemos a nuestros ejemplo intentando guiar a la estación de colectivos. En la unidad I vimos que antes de dar las instrucciones a alguien hay que conocer de antemano el proceso y si no lo conocemos tenemos que plantear una estrategia para resolverlo.

Ahora llegó el momento de decirle a nuestro amigo como llegar, o sea darle a conocer nuestro algoritmo que resuelve su problema que era:

sigue derecho 2 cuadras, dobla a la izquierda y sigue adelante otras 4 cuadras, ahí busca una pequeña cortada que te deja en la puerta de la Terminal

Pero resulta que nuestro amigo no sabe castellano, y como el solo entiende chino mandarín hay que hacer una traducción, pero nosotros no sabemos chino así que se necesita un traductor castellano - chino que nos haga el trabajo.

Análogamente, con las computadoras sucede lo mismo, las computadoras solo entienden un lenguaje llamado código máquina, que es el lenguaje de instrucciones del microprocesador, existen tantos lenguajes de código máquina como tipos de microprocesadores existen, en nuestro caso vendría a ser como muchos dialectos distintos de chino.

Sin embargo no encontramos un traductor castellano - chino, pero si uno ingles - chino, como para nosotros el ingles es más familiar y conocido que el chino podemos traducir nuestras instrucciones al ingles dárselas al traductor y este traducirlas al chino mandarín con lo cual nuestro amigo podrá llegar a la Terminal fácilmente.

Análogamente existe un lenguaje intermedio entre nuestro castellano (lenguaje natural) y el código máquina (lenguaje del microprocesador), en realidad existen muchos lenguajes intermedios, son los lenguajes de programación, en nuestro caso el lenguaje que vamos a aprender es el C.

¿Por qué no aprender directamente lenguaje de máquina y ahorrarnos un paso?

Porque el código máquina es muy difícil de interpretar a simple vista debido que su representación codificada se puede mostrar en números (ya sea binario, decimal o hexadecimal) o en caracteres ASCII (u otra codificación), traducir un número a su correspondiente instrucción es una tarea muy ardua para hacer a mano. Prácticamente en la actualidad nadie programa directamente en código máquina, cada microprocesador tiene al menos un lenguaje intermedio llamado Asemmbler o Ensamblador para facilitar la codificación (ver diferencia entre código máquina y ensamblador http://es.wikipedia.org/wiki/Lenguaje_ensamblador).

¿Y porque no traducir directamente el lenguaje natural a código máquina con un programa?

Porque los lenguajes naturales tienen principalmente un par de problemas:

El primer problema es el la ambigüedad semántica, está se produce cuando la misma oración puede ser interpretada con significados diferentes, por ejemplo: Sal de Argentina. Esto puede interpretarse como una orden para salir de Argentina, o puede estar refiriéndose a un objeto, la sal de mesa, proveniente del lugar o región Argentina. Para desambiguar nosotros (como humanos) usamos varías estrategias entre ellas referirnos por el contexto donde fue dicha la frase, o apelar a conocimientos previos, información que no viene acompañada dentro de la oración.

El segundo problema es la ambigüedad sintáctica, la oración se puede interpretar con significado diferente debido a la estructura de la oración, por ejemplo: Las niñas y las ancianas enfermas fueron al sanatorio. Aquí no se sabe si las niñas y las ancianas están enfermas o solo las ancianas.

Si para nosotros es complicado resolver las ambigüedades, más para un programa que no tiene acceso al contexto del lenguaje natural.

Por eso mismo los lenguajes de programación tienen la característica que no son ambiguos de ninguna forma, y no requiere de información de contexto ni conocimientos previos.

Entonces el esquema de trabajo sería el siguiente:

1. *Tenemos un problema*
2. *Pensamos una solución*
3. *Creamos un algoritmo*
4. *Lo escribimos en un lenguaje de Programación*
5. *Usamos un programa que nos traduzca a código máquina*

Resumiendo, escribir un programa (algoritmo) en un lenguaje de programación es mucho más sencillo que hacerlo en código máquina para nosotros los humanos, y traducir un programa desde un lenguaje de programación en forma automática a código máquina es sencillo para otro programa ya que no tiene que resolver los problemas de ambigüedad del lenguaje natural.

Compiladores e Intérpretes

Existen a grandes rasgos, dos clases de programas traductores, los llamados compiladores y los llamados interpretes, por extensión a los lenguajes de programación que se basan en una de estas dos clases se los llama lenguajes compilados o lenguajes interpretados según la clase.

Un interprete funciona como los traductores de las sesiones de Naciones Unidas o algún evento o congreso internacional, a medida que una frase es dicha el interprete inmediatamente la traduce, en nuestro caso cada instrucción de un algoritmo es traducida una a la vez y luego ejecutada por el microprocesador.

En cambio un compilador es como un traductor de un libro, primero traduce todo el programa a código máquina y luego lo ejecuta el microprocesador.

Las diferencias:

Un compilador tarda mucho más tiempo desde que comienza el trabajo de traducción hasta que se ejecuta que un interprete, un interprete iniciará la ejecución casi de inmediato pero luego, en tiempo total de ejecución del programa compilado será mucho menor que el interpretado ya que no pierde tiempo en la traducción.

Otra diferencia es que por cada vez que se ejecuta un programa interpretado, se tiene que volver a hacer la traducción del mismo instrucción por instrucción, en cambio un programa compilado solo se traduce una vez.

El lenguaje C es un lenguaje compilado, esto significa que para ver el resultado del programa primero debe traducirse en forma completa, en la jerga se dice compilar y es la palabra que usaremos de aquí en adelante.

Código Fuente, Código Objeto

La manera de escribir un programa, sobre todo en C, es en un archivo de texto, ese código escrito en lenguaje C (u otro lenguaje de programación) se lo llama código fuente.

El código fuente es lo que el compilador va a compilar, el resultado de la compilación es otro archivo, el cual contiene lo que se llama código objeto, que es propiamente el programa en código máquina.

Enlazador o linkeador

Sin embargo aún no puede ejecutarse, ya que por una razón de metodología de programación, varias personas podrían escribir diversas partes del programa, que luego hay que juntar, el último paso antes de poder ejecutar un programa es enlazar todos los códigos objetos del mismo en un solo archivo ejecutable. Para eso se utiliza otro programa que se llama Enlazador o Linkeador.

IDE

Al principio, la forma de escribir programas era con un simple procesador de textos donde se guardaba el archivo de textos con el fuente, luego se iba al compilador y se compilaba el archivo de texto y se generaba un archivo objeto, luego se iba al enlazador y se generaba un archivo ejecutable, y por último se ejecutaba el programa y se veía si funcionaba.

Luego a alguien se le ocurrió que porque hacer todos los pasos por separado, porque no hacer un programa que edite, compile y enlace, para no salir y entrar en cada programa por separado. Entonces surgieron los IDEs que es un Entorno de Desarrollo Integrado (del ingles IDE Integrated Developed Enviroment).

A medida que los IDEs han ido evolucionando, han integrado más cosas, como por ejemplo un seguimiento de errores, una ejecución del programa dentro del IDE en varias modalidades para depuración (encontrar y reparar) de errores, coloreo de sintaxis, autocompletar instrucciones, etc.

Un ejemplo de IDE para C es el ZinjaI que vamos a utilizar en el curso.
http://zinjai.sourceforge.net/

RAD

Siguiendo la evolución de los IDEs aparece las herramientas RAD, Desarrollo Rápido de Aplicaciones (del ingles RAD Rapid Application Developed), estás herramientas son IDEs especializados para la confección de programas en forma rápida, para ello, acotan a un tipo de programa y proveen herramientas que generan bloques de programación comunes al tipo. Generalmente traen herramientas para la generación de ventanas y elementos comunes a las ventanas de diálogo, conexión a base de datos, componentes para Internet, etc.

Un ejemplo de **RAD** es Visual Studio que se va a usar durante la carrera.

A esta altura recomendamos tener instalado el IDE ZinjaI para correr este ejemplo de juego.

Ejemplo de programa en C++

Disparo de Cañon *(Archivo Fuente número 1)*

```
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <cstdio>
#include <ctime>
using namespace std;
#define G 9.8 //aceleracion de la gravedad 9,8 m/s
#define PI 3.14162

int main(int argc, char *argv[]) {
        float a; //variable que contendra el angulo del disparo
```

```
        float d; //variable que contendra la distancia
        const float V0=20; //velocidad inicial 20 m/s
        int x;
        srand(time(NULL)); //se obtiene una distancia al azar
        x=rand()%201+100;
        cout<<"Intenta dar en el Blanco"<<endl<<"Para ello tienes cinco balas"<<endl;
        cout<<"Te indicare que tan lejos del blanco has hecho impacto"<<endl;
        cout<<"El blanco se encuentra a "<<x<<" metros"<<endl<<endl;
        cout<<"introduce el angulo de disparo 1: ";
        cin>>a;
        d=pow(V0,2)*sin(2*a*PI/180); //calculo de la distancia
        cout<<"\nLa bala cayo a "<<d<<" metros.\n";
        cout<<"introduce el angulo de disparo 2: ";
        cin>>a;
        d=pow(V0,2)*sin(2*a*PI/180); //calculo de la distancia
        cout<<"\nLa bala cayo a "<<d<<" metros.\n";
        cout<<"introduce el angulo de disparo 3: ";
        cin>>a;
        d=pow(V0,2)*sin(2*a*PI/180); //calculo de la distancia
        cout<<"\nLa bala cayo a "<<d<<" metros.\n";
        cout<<"introduce el angulo de disparo 4: ";
        cin>>a;
        d=pow(V0,2)*sin(2*a*PI/180); //calculo de la distancia
        cout<<"\nLa bala cayo a "<<d<<" metros.\n";
        cout<<"introduce el angulo de disparo 5: ";
        cin>>a;
        d=pow(V0,2)*sin(2*a*PI/180); //calculo de la distancia
        cout<<"\nLa bala cayo a "<<d<<" metros.\n";
        system("PAUSE"); //espera la pulsación de una tecla
        return 0;
}
```

Estructura de un Programa C++

La estructura de un programa C++ es la siguiente:

Directivas del Procesador

· Incluye
· Macros
Datos y Funciones Externas
Datos y Funciones Globales

*La función **main()** siempre está presente en todo programa C++.*

```
main()
{
.....
}
```

En general, todo programa C++ se compone de funciones. Una función, como idea general, se presenta como un subalgoritmo que forma parte del algoritmo principal, el cual permite resolver una tarea específica.

Main () es la función principal que siempre está presente porque el programa comienza su ejecución en ella. Las acciones o instrucciones del programa se plantean dentro del bloque delimitado por las llaves de inicio (**{**) y de fin (**}**) de esta función.

Comentarios

Dentro del código fuente es posible agregar indicaciones para que otros programadores o uno mismo recuerden o sigan o aclaren aspectos del programa.

Los comentarios son ignorados por el compilador porque son de uso exclusivo para humanos. Hay dos formas de coloca un comentario en C++:

Con una doble barra // el compilador interpreta que todo lo que venga después hasta el final de línea es un comentario

Encerrando el comentario entre /* y */ en este caso puede abarcar varias líneas.

Elementos (Tokens) de un Programa C++

Todo programa C++ se construye a base de tokens o elementos básicos de léxico. Existen cinco clases de tokens:

- Identificadores
- palabras reservadas
- literales
- operadores
- separadores.

Describiremos brevemente estos elementos básicos de C++.

Identificadores

Los identificadores son los nombres que empleamos para representar a importantes elementos de un programa, tal como una constante, una variable, una función, un tipo de dato, o un programa. Algunos identificadores corresponden a elementos predefinidos de C++ y se denominan identificadores estándar. Pero en muchas situaciones es el programador el que debe proponer el identificador de un elemento de programa; para hacerlo en C++, recuerde las siguientes reglas:

· Utilizar como primer caracter una letra
· Continuar con letras, dígitos o guión bajo (_)
· No utilizar palabras reservadas de C++
· C++ considera diferentes las mayúsculas de las minúsculas

Ejemplos de identificadores válidos:

x
y23
suma
Resultado_del_Calculo

Ejemplos de identificadores no válidos:

4to
char
el producto
tasa&porcentaje

Nota: La elección adecuada de los identificadores favorece la comprensión del programa. No se deberían usar identificadores muy largos ni demasiado cortos. Es recomendable que sugieran un significado.

La diferencia entre constante y variable es que las constantes no cambian de valor en la ejecución de todo el programa, la variable puede cambiar de valor.
Palabras Reservadas (Identificadores standard)

ANSI/ISO C++ posee el siguiente conjunto de identificadores standard que constituyen palabras reservadas del lenguaje y no pueden emplearse con otro fin:

asm	*continue*	*float*	*new*	*signed*	*try*
auto	*default*	*for*	*operator*	*sizeof*	*typedef*
break	*delete*	*friend*	*private*	*stact*	*union*
case	*do*	*goto*	*protected*	*struct*	*unsigned*
catch	*double*	*if*	*public*	*switch*	*virtual*
char	*else*	*inline*	*register*	*template*	*void*
class	*enum*	*int*	*return*	*this*	*volatile*
const	*extern*	*long*	*short*	*trhow*	*while*

Literales (constantes)

Constituyen valores con significado propio y único. Por ejemplo

2.3
'a'
102
"programa"
OXF2B .

El último ejemplo corresponde a una constante en formato hexadecimal.

Operadores

Constituyen elementos del léxico de C++ que permiten *conectar* operandos provocando un cálculo (computación) determinado.
Algunos de ellos son: + - * / = ¡ < > == [ ] : ; % { }

Separadores

C++ considera dentro de este grupo a los espacios en blanco, avances de línea, retornos de carro y tabulaciones

Tipos de Datos Estándar de C++

El compilador de C++ recononce tres tipos de datos estándar: enteros, punto flotante y caracter.

	Enteros	
Tipo	**Rango**	**Tamaño (bytes)**
char	--127 .. 128	1
unsigned char	0 .. 255	1
short	-32768 .. 32767	2
unsigned short	0.. 65535	2
int	-2.147.483.648 .. 2.147.483.647	4
unsigned int	0.. 4.294.967.295	4
long	-2.147.483.648 .. 2.147.483.647	4
unsigned long	0.. 4.294.967.295	4

Reales (punto flotante)		
Tipo	**Rango**	**Tamaño (bytes)**
float	3.4 x 10-38 .. 3.4 x 1038	4
double	1.7 x 10 -308.. 1.7 x 10 308	8
long double	3.4 x 10 -4932 .. 3.4 x 10 4932	10

Caracter		
Tipo	**Rango**	**Tamaño (bytes)**
char	-128 .. 127	1
unsigned char	0 .. 255	1

Lógico		
Tipo	**Rango**	**Tamaño (bytes)**
bool	false,true	1

Nulo		
Tipo	**Rango**	**Tamaño (bytes)**
void	----	0

Notación y Definición de Constantes en C++

C++ admite 4 tipos de constantes diferentes: *literales, definidas, declaradas y enumeradas.*

Constantes literales

Tienen una notación y sintaxis determinada de acuerdo al tipo de dato que se desee expresar. Veamos algunos ejemplos:

Tipo de constante literal	**Ejemplos**	
Entera decimal	123 -5	Secuencia de dígitos decimales con o sin signo
Entera octal	0455	Comienzan siempre con cero
Entera hexadecimal	0XF4A	Comienzan siempre con 0X
Real o punto flotante	192.45 .76 -1.3e+4	Se emplea el punto decimal y/o notación científica
Char	‘A’ ‘\n’ ‘\f’	Caracteres del código ASCII Secuencia de escape para nueva línea Secuencia de escape para nueva página
string	“Facultad”	

Constantes definidas

Ciertas constantes pueden referenciarse en C++ a través de un nombre simbólico empleando la directiva #define.

```
#define valor 100
#define Pi 3.14159
#define proximalinea '\n'
```

C++ empleará los valores 100, 3.1459 y "\n" cuando encuentre en el programa los identificadores valor, Pi y proximalinea

Constantes declaradas: *const y volatile*

Es posible declarar en C++ constantes con nombres o identificadores a través del calificador *const*, indicando además el tipo asociado a la constante.
El calificador *const* en realidad tiene el efecto de una declaración de variable, solo que el valor asignado al identificador simbólico no puede alterarse. Si se emplea el calificador *volatile* el valor puede ser modificado en el programa o por el hardware y software del sistema.

```
const int n = 200 ;
const char letra = "B";
const char cadena[] = "Programacion"
volatile int m = 35
```

Constantes enumeradas

Son valores definidos por el programador y agrupados bajo un nombre. Este nombre constituye entonces Esto permite más adelante declara una variable asociándola al nombre del grupo.

```
enum meses { ene, feb, mar, abr, may, jun, jul, ago, set, oct, nov, dic } /* lista de constantes enumeradas */

meses mes = abr  /* declaración de la variable mes del tipo meses e inicializada con el valor abr */
```

Declaración e Inicialización de variables

Variable: posición de memoria donde se almacena un valor y que es representada por un nombre o identificador.

Una variable sirve para recordar valores dentro del programa que pueden ir cambiando a medida que este progresa en su ejecución.

En C++ toda variable debe tener asociado un tipo, un tipo le indica a C++ que clase de valores va a manejar esa variable y su rango, lo cual se hace al declararse o inicializarse la variable en el programa. La declaración puede hacerse en cualquier lugar del programa, pero antes de que la variable se use.

Declaración

En C++ se declara un variable indicando un tipo y luego el nombre o identificador de la variable.

```
int x; // declaración de la variable x de tipo entera
```

Definición: definir una variable en C++ implica asignar un valor, almacenándolo en el espacio de memoria correspondiente a la variable.

```
x = 27; // inicialización de la variable x con el valor 27
```

Es posible declarar y definir (inicializar) una variable en una misma acción.

```
float y = -2.35;  */ declaracion y definicion de y como float y su inicializacion con el dato –2.35 */
char letra = ‘A’;  // declaracion de letra e inicializacion con el valor ‘A’
extern int z;  /* declaracion de z pero sin reservar espacio para almacenamiento es decir sin definirla en el programa */
```

Ámbito de validez de una variable

El alcance o ámbito de validez de una variable se limita al bloque del programa en donde fue declarada. Si se requiere una variable global que pueda ser empleada en cualquier bloque debería declarase fuera de la función principal *main*

Analicemos el ejemplo siguiente: *(Archivo Fuente número 2)*

```
#include <iostream>
using namespace std;
int main(void)
{
   int a=54;
  { // inicio del bloque anidado
    int b = 20;
    char a = 'Z' ;
    cout << a<<" "<<b<<"\n";
  } // fin del bloque anidado
  cout <<a<<" "<<b<<"\n" ;
  return 0;
}
```

En el ejemplo del recuadro la variable a fue inicialmente declarada y definida en el bloque principal de la función main() como entera y con un valor inicial de 54. Al ser declarada nuevamente en el bloque anidado pero de tipo char permite definir su alcance o ámbito dentro de este bloque prevaleciendo sobre la anterior declaración que usa el mismo nombre. Es decir que el primer flujo de salida cout del programa permitirá obtener Z y 20.

El segundo flujo de salida producirá un error de compilación, pues la variable b no fue definida en ese bloque.

Nota: no es una buena práctica de programación emplear identificadores duplicados de variables en un programa. El ejemplo sólo tiene el fin de mostrar el concepto de ámbito y alcance de las variables en C++.

Entrada y Salida

Flujos de Entrada y Salida

Un flujo de Entrada/Salida o *I/O stream* es un secuencia de caracteres que se envían (fluyen) desde o hacia un dispositivo. En la I/O estándar, C++ utiliza cout para enviar caracteres a la pantalla; y cin para tomar caracteres desde el teclado. También disponemos de otros dos flujos cerr y clog para manejo de errores.

Los flujos cin, cout, cerr y clog, son clases predefinidas de C++, las cuales se hallan en el archivo iostream.h. Esto significa que se debe incluir este archivo en la cabecera de su programa para que el compilador enlace las rutinas de definición necesarias e interprete las llamadas a estos flujos. Si no utilizamos archivos, el dispositivo predefinido para entrada y salida será el monitor de video.

Observe el ejemplo del recuadro anterior donde la primer línea del código fuente se indica la inclusión de este archivo:

```
#include <iostream>
```

La directiva #include es parte de las instrucciones del preprocesador del compilador, al compilador de C++ se le puede ajustar el comportamiento de compilado, todas las directivas comienzan con #, #include es la más usada de todas (junto con #define) y su funcionamiento es incluir o agregar a nuestro programa una librería de funciones.

Como vimos en párrafos anteriores los identificadores estándar son pocos, la potencia de C++ se encuentra en sus librerías, en la gran cantidad de funciones que existen para resolver gran cantidad de problemas. Para incorporar esas funciones se debe usar la directiva #include según la función en que librería se encuentre.

El flujo de salida cout requiere del *operador de inserción o salida* << (dos signos *menor que* consecutivos) para enviar la información a la pantalla. *(Archivo Fuente número 3)*

```
#include <iostream>
using namespace std;
int main(void)
{
        cout<<"Comando de flujo de salida en C++";
        return 0;
}
```

de igual forma opera el comando de flujo cin pero empleando los operadores de extracción o entrada >> (dos signos *mayor que* consecutivos)
(Archivo Fuente número 4)

```
#include <iostream>
using namespace std;
int main(void)
{
        int edad, anio_nac;
        cout << "Escriba su edad:" ;
        cin >> edad;
        anio_nac = 2010 – edad;
        cout << "\n"; //Caracter especial o secuencia de escape para producir un avance de linea.
        cout << "Ud. ha nacido en " << anio_nac;
        return 0;
}
```

Caracteres especiales y manipuladores para I/O

Es posible enviar en el flujo de salida algunos caracteres especiales o secuencias de escape que permiten lograr una salida más legible y mejorar la interfaz con el usuario. Algunos de ellos son:

Secuencia de escape	Caracter	Efecto
\a	BEL	Campana o pitido de alerta
\b	BS	Retroceso (Backspace)
\f	FF	Avance de página
\n	LF	Avance de línea
\r	CR	Retorno de carro
\t	HT	Tab horizontal
\v	VT	Tab Vertical
\	\	Barra invertida (backslash)
'	'	Apóstrofo
\"	"	Doble comilla
\?	?	Marca de interrogación

En la tabla siguiente se proponen algunos casos de caracteres especiales:

Ejemplo de código C++	Salida

int a=20;
int b= 50;
cout << "Datos: \n a = " << a << "\n b= "<< b
	Datos:
	a = 20
	b = 50

int a=20;
int b= 50;
cout<<"Datos:\n a ="<<a<<"\t b= "<< b
	Datos:
	a = 20 b = 50

cerr “\a Se ha producido un error”	(suena un pitido) Se ha producido un error

Existen además manipuladores de flujo a través de los cuales se puede filtrar la información logrando algún efecto, como efectuar un cálculo, una secuencia de escape idéntica a las de la tabla anterior o establecer un formato de salida, etc.

Manipulador	**Efecto**	**Ejemplo**
endl	Avance de línea (“\n”)	cout << “a=”<< a << endl << “b=”<< b
hex	Exhibirá el siguiente valor en formato hexadecimal	cout << hex << 1000
dec	Exhibirá el siguiente valor en formato decimal	cout << dec << x
oct	Exhibirá el siguiente valor en formato octal	cout << oct << 105
setbase( )	Establece la base para mostrar el siguiente valor	cout << setbase(8) << dato
setw( )	Determina ancho de campo para mostrar la información	cout << “Resultado:”<< setw(20)<<r
setfill( )	Establece un caracter de relleno	cout << setfill(‘.’)
setprecision( )	Determina el número de dígitos de la fracción decimal en la presentación de números reales	cout << seprecision(4) << 10.0/3.0

La tabla anterior muestra algunos de los manipuladores disponibles. La mayoría se encuentra definido en el archivo de cabecera iomanip.h por lo cual es necesario incluirlo en el encabezado del programa.

(Archivo Fuente número 5)

```
#include <iostream>
#include <iomanip>
using namespace std;
int main(void)
{
        cout << “Lenguajes de Programacion”;
        cout << endl<<ende;
        cout << setfill(‘.’);
        cout << “1. Cobol” << setw(20)<< “pag. 1”<<endl;
        cout << “2. Fortran” << setw(20)<< “pag. 2”<<endl;
        cout << “3. Basic” << setw(20)<< “pag. 3”<<endl;
        cout << “4. Pascal” << setw(20)<< “pag. 5”<<endl;
        cout << “5. ANSI/ISO C++” << setw(20)<< “pag. 8”<<endl;
        return 0;
}
```

Bibliografía:

Ing. Horacio Loyarte. *Fundamentos de Programación*. [UNL -FICH]. [2008]