UNL FICH Facultad de Ingeniería y Ciencias Hídricas

Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

FUNDAMENTOS DE PROGRAMACIÓN

Asignatura correspondiente al plan de estudios de la carrera de Ingeniería Informática

UNIDAD 6
INTRODUCCIÓN AL LENGUAJE ANSI/ISO C++

UNIDAD 6

Introducción al Lenguaje ANSI/ISO C++

Resumen de Conceptos

Breve historia de C++

En 1967 Martin Richards creó un lenguaje de programación llamado BCPL para escribir sistemas operativos y compiladores. Luego, Ken Thompson creó el lenguaje B basándose en el BCPL. Con B, Thompson escribió las primeras versiones de UNIX en los laboratorios Bell durante 1970. Estos 2 lenguajes eran muy *rústicos* y dejaban muchas tareas al programador.
En 1972 Denis Ritchie -también de Laboratorios Bell- escribe un lenguaje basado en BCPL y B con varias mejoras que contribuyeron a su posterior popularidad; lo llamó C. La eficiencia del C en términos de ejecución y administración de recursos lo hizo el preferido de las empresas de software que diseñaban sistemas operativos y compiladores. Una de sus principales características es su independencia del hardware, lo cual permitía inicialmente correr programas C en cualquier plataforma con mínimas modificaciones. Pero las empresas de software comenzaron a diseñar versiones de C particulares que le quitaban portabilidad a los programas. Por eso, en 1983 el American Nacional Standard Institute (ANSI) creó un comité técnico para su estandarización. La versión aprobada junto a la Organización Internacional de Estandarizacón (ISO) vio la luz en 1990 y se la conoce como ANSI C.

En 1980 Bjarne Stroustrup en Laboratorios Bell, comenzó a experimentar con versiones mejoradas de C (C con clases) con la única finalidad de escribir programas de simulación orientada a eventos. Stroustrup llamó a su nuevo lenguaje C++. Este lenguaje fue creciendo con renovadas características pero, manteniendo la compatibilidad con su antecesor C. C++ incorporó los conceptos del paradigma de programación orientada a objetos basándose en SIMULA67, tipos genéricos y excepciones de ADA, la posibilidad de declarar variables en cualquier lugar de ALGOL68, así como otras características originales que no existían antes: herencia múltiple, espacios de nombre, funciones virtuales puras, etc. Más tarde, Alex Stepanov y Andrew Koenig idearon la biblioteca de plantillas standard (STL), que le dió a C++ una potencia única entre los lenguajes de alto nivel.

Debido a la enorme difusión de C++, y -nuevamente- a las diferentes versiones que fueron apareciendo, las organizaciones ANSI e ISO se reunieron en 1990 para definir el Standard de este lenguaje, el cual fue aprobado en 1998. Actualmente, un comité ISO continúa manteniendo y haciendo evolucionar al lenguaje, en un esquema de trabajo que prevee una versión cada 3 años. La última versión con mejoras de gran impacto en la forma de programar con C++ fue C++11 (de 2011),

y la versión más actual es C++14 (de 2014). Se espera una próxima versión con importantes novedades para 2017.

Hoy en día, C++ posee una notable inserción en el mundo de las computadoras, lleva 30 años manteniéndose entre los lenguajes más populares y más utilizados, aún sostiene una fuerte presencia en el mercado y es el lenguaje preferido para desarrollos donde el rendimiento y la gestión de recursos resultan críticos, como en la programación de sistemas operativos y compiladores (software de base).

Estructura de un Programa C++

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

```
includes

funciones globales

int main( ) {
   cuerpo del programa
}
```

En general, todo programa C++ se compone de funciones (concepto que desarrollaremos en la unidad 8). **main ( )** es la función principal, siempre presente en todo programa C++. Las acciones o instrucciones del programa se plantean dentro del bloque delimitado por las llaves de inicio (**{**) y de fin (**}**) de esta función. Esta es la función que se ejecuta al inciar el programa (similar al "Proceso" que definíamos en pseudocódigo).

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.

Palabras Reservadas (Identificadores standard)

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

alignas	continue	friend	register	true
alignof	decltype	goto	reinterpret_cast	try
asm	default	if	return	typedef
auto	delete	inline	short	typeid
bool	do	int	signed	typename
break	double	long	sizeof	union
case	dynamic_cast	mutable	static	unsigned
catch	else	namespace	static_assert	using
char	enum	new	static_cast	virtual
char16_t	explicit	noexcept	struct	void
char32_t	export	nullptr	switch	volatile
class	extern	operator	template	wchar_t
const	false	private	this	while
constexpr	float	protected	thread_local	
const_cast	for	public	throw	

Literales (constantes)

Constituyen valores con significado propio y único. Por ejemplo

3.14 **1e9** **'a'** **102** **"programa"** **0xF2B** **true**.

Más adelante se detallan las reglas para introducir constantes según cada tipo de dato.

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++

Todo compilador de C++ recononce los siguientes tipos de datos estándar:

Numéricos 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

Numéricos 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	**Valores**	**Tamaño (bytes)**
bool	false,true	1

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

Objetos string

Para operar cadenas de caratcteres utilizaremos el tipo string (clase string). Cuando estudiemos punteros y objetos analizaremos el tamaño y funcionamiento de este tipo de dato. Por el momento, será utilizado como un tipo de dato más, sin que sea necesario hacer cosideraciones particulares.

Cadenas de caracteres		
Tipo	**Rango**	**Tamaño (bytes)**
string	----	----

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.

En las constantes de tipo numérico se puede utilizar notación científica (ej: **1e9**), los signos **+** y **-** (**-3**, **+2**), el punto como separador de decimales (**3.14**), y la comilla simple (**‘**) como separador de miles opcional (**1’234’567.89**). Al ingresar un real con parte entera igual a cero se puede omitir el 0 y comenzar directamente con el punto (**.54**). Opcionalmente, se pueden introducir constantes en otras bases utilizando prefijos especiales como por ejemplo hexadecimal (**0xF2B**) y binario (**0b11001**).

Para las variables lógicas (**bool** en C++), los dos posibles valores son **true** (verdadero) y **false** (falso).

Para los datos de tipo cadena se utilizan las comillas dobles (**“programa”**), mientras que para los caracteres individuales las comillas simples (**‘x’**).

Veamos algunos de los ejemplos más habituales:

Tipo de constante literal	Ejemplos	
Entera decimal	123 -5	Secuencia de dígitos decimales con o sin signo
Real o punto flotante	192.45 .76 -1.3e+4	Se emplea el punto decimal y/o notación científica.
char	‘A’ ‘\n’	Caracteres del código ASCII Secuencia de escape para nueva línea
string	“Facultad”	

Constantes declaradas: *const*

Es posible declarar en C++ constantes con nombres o identificadores a través del calificador *const*. Este calificador se utiliza en una declaración de variable, para que el valor asignado al identificador simbólico no puede alterarse.

```
const  int  n = 200 ;
const  char letra  = "B";
const  char cadena[] = "Programación"
```

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.

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

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

Definición e Inicialización de variables

Estudiamos en algorítmica computacional el concepto de variable: posición de memoria donde se almacena un valor y que es representada por un nombre o identificador.
En C++ toda variable debe tener asociado un tipo, 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 sea invocada; esto permite reservar el espacio de memoria necesario de acuerdo al tipo asociado (int, char, double, etc.).

Definición: en C++ se declara y define un variable indicando un tipo y luego el nombre o identificador de la variable.

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

Inicialización: inicializar una variable en C++ implica asignar un primer 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;  // declaración y definición de  y como float e
                  //  inicialización con el  dato  –2.35
char letra = 'A';  // declaración de letra e inicialización con el
```

// valor 'A'

Ambito 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:

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

Declaración y definición de ***a***

Definición de ***b*** *y segunda definición de* ***a*** *en el bloque anidado*

Salida: ***Z 20***

Causará error de compilación: ***b*** *no está definida en este bloque.*

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 solo tiene el fin de mostrar el concepto de ámbito y alcance de las variables en C++.

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 un archivo de salida; y **cin** para tomar caracteres desde un archivo de texto. 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**. Esto significa que Ud. debe incluir este archivo en su programa para que el compilador procese las definiciones necesarias e interprete

las llamadas a estos flujos. El dispositivo predefinido para la entrada será el teclado y para la salida el monitor.

Observe el ejemplo anterior donde la primer línea del código fuente se indica la inclusión de este archivo: **`#include <iostream>`**. Estudiaremos la directiva **#include** más adelante, la cual nos permitirá incluir código de otros archivos fuente C++ (denominados "cabeceras" o "headers") en a nuestro programa.

Además del **#include**, el ejemplo utiliza la sentencia **using namespace std**. Esto se debe a que en realidad, todas las definiciones del estandar se agrupan en un "espacio de nombres" que se llama **std**. Entonces, **cout** es en realidad **std::cout** (es decir, el **cout** definido dentro del espacio de nombres **std**). Lo mismo ocurre con **endl**, **cin**, y la mayoría de los objetos y funciones que utilizaremos en los ejemplos. La directiva **using namespace std** le indica al compilador que debe buscar los identificadores que utilizemos dentro de ese espacio de nombres automáticamente, aunque no lo explicitemos para cada uno de ellos. De esta forma, nos evita tener que repetir **std::** una y otra vez.

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

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

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

```
#include <iostream>
using namespace std;
int main() {
   int edad, anio_nac;
   cout << "Escriba su edad:" ;
   cin >> edad;
   anio_nac = 1998 - edad;
   cout << endl; // salto de linea
   cout << "Ud. ha nacido en  " << anio_nac;
}
```

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 <<"Se ha producido un error"	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** por lo cual es necesario incluirlo en el encabezado del programa.

```
#include <iostream>
#include <iomanip>
using namespace std;
int main() {
    cout << "Lenguajes de Programación" << endl<<endl;

    cout<<setfill('.')
    cout<<"1. Basic"<<setw(20)<<"pág. 1"<<endl;
    cout<<"2. Pascal"<<setw(20)<<"pág. 2"<<endl;
    cout<<"3. Go"<<setw(20)<<"pág. 3"<<endl;
    cout<<"4. Python"<< setw(20)<<"pág. 5"<<endl;
    cout<<"5. ISO C++"<< setw(20)<<"pág. 8"<<endl;
}
```

Estudie y analice la salida de este programa. Investigue el efecto de los manipuladores utilizados.