Universidad Autónoma de Entre Ríos
Facultad de Ciencia y Tecnología
Sede: Oro Verde

FUNDAMENTOS DE PROGRAMACIÓN

RESUMEN DE CONTENIDOS N°6

INTRODUCCIÓN AL LENGUAJE C++

RESUMEN DE CONTENIDOS N° 6

Introducción al Lenguaje C++

Breve historia de C++

En 1967 Martin Richards creó un lenguaje de programación BCPL para escribir sistemas operativos y compiladores y Ken Thompson creó el lenguaje B basándose en el BCPL. 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 era 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 National Standards 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 de Laboratorios Bell, comenzó a experimentar con versiones mejoradas de C (C con clases) con la única finalidad de escribir programas de simulación orientados a eventos.
Stroustrup llamó a su nuevo lenguaje C++. Este compilador fue creciendo con renovadas características que lo hicieron muy original, manteniendo la compatibilidad con su antecesor C.
C++ incorpora clases y funciones virtuales 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 con nombre, funciones virtuales puras, etc. Alex Stepanov y Andrew Koenig idearon la biblioteca de plantillas standard (STL), la cual le da a C++ una potencia única entre los lenguajes de alto nivel.

Debido a la enorme difusión del 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.

Hoy día, C++ posee una notable inserción en el mundo de las computadoras y es uno de los lenguajes clásicos de programación: tanto de sistemas operativos y compiladores (software de base) como de aplicaciones.

¡No hay mejor forma de aprender la sintaxis de un lenguaje de programación que programando!

Concepto de abstracción

Abstracción, abstracto, dos términos muy comunes en la jerga de la informática.
Pero, ¿qué es abstracción?. Para explicarlo, utilizaremos la siguiente definición:

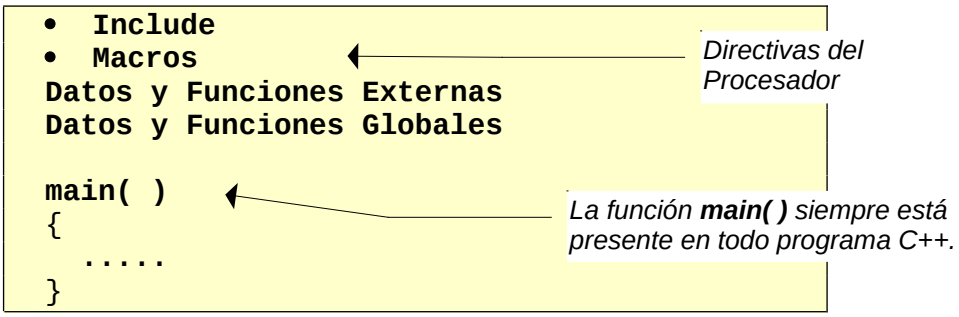
Abstracción, del latín *abstractio*, concepto vinculado al verbo abstraer (separar las propiedades de un objeto a través de una operación mental, dejar de prestar atención al mundo sensible para centrarse en un pensamiento). La abstracción, por lo tanto, es alguna de estas acciones o sus efectos.

"*Dejar de prestar atención al mundo sensible para centrarse en un pensamiento*", no es sólo una frase bonita, es todo un proceso evolutivo, que debemos asimilar, quienes nos embarcamos en la programación y el mundo de los sistemas informáticos.

En lo referente a programación, cuando se habla de "abstracción", se hace referencia a "lo que hace" un elemento, una estructura, una sentencia, y no en el cómo lo logra. Este concepto, es conocido como característica de "Caja Negra".

Estructura de un Programa C++

La estructura de un programa ANSI/ISO C++ es la siguiente:



En general, todo programa C++ se compone de funciones. **Main ( )** es la función principal siempre presente, donde las acciones o instrucciones del programa se plantean dentro del bloque delimitado por las llaves de inicio (**{**) y de fin (**}**) de esta función.

Como agregar Comentarios y documentación interna

Hay dos maneras:

```
  /* Todo lo que esta entre las barras es un comentario y puede abarcar
varias líneas  */

  // Esto es un comentario que finaliza al terminar la línea
```

Archivos de encabezado de la Biblioteca estándar de C++

La Biblioteca estándar de C++, está dividida en muchas porciones, cada una con su propio archivo de encabezado. Los archivos de encabezado contienen, entre otras cosas, definiciones de varios tipos de clases y funciones, así como las constantes que necesitan las funciones. Un archivo de encabezado "instruye" al compilador acerca de cómo interconectarse con los componentes de la biblioteca y los componentes escritos por el usuario.
A continuación, se listan algunos archivos de encabezado comunes de la Biblioteca estándar de C++, la mayoría de los cuales se verán más adelante.

<iostream> Contiene prototipos de función para las funciones de salida y entrada estándar de C++.

<iomanip> Contiene prototipos de función para los manipuladores de flujo que dan formato a flujos de datos.

<cmath> Contiene prototipos de función para las funciones de la biblioteca de matemáticas.

<cstdlib> Contiene prototipos de función para las conversiones de números a texto, de texto a números, asignación de memoria, números aleatorios y varias otras funciones utilitarias.

<cctype> Contiene prototipos de función para las funciones que evalúan caracteres con base en ciertas propiedades (por ejemplo, si el carácter es un digito o un signo de puntuación), y prototipos de funciones que se pueden utilizar para convertir letras minúsculas a letras mayúsculas y viceversa.

<cstring> Contiene prototipos de funciones para las funciones de procesamiento de cadenas estilo C.

<typeinfo> Contiene clases para la identificación de tipos en tiempo de ejecución.

<fstream> Contiene prototipos de funciones para las funciones que realizan operaciones de entrada desde archivos en disco, y operaciones de salida hacia archivos en disco.

<string> Contiene la definición de la clase string de la Biblioteca estándar de C++.

<sstream> Contiene prototipos de función para las funciones que realizan operaciones de entrada a partir de cadenas en memoria, y operaciones de salida hacia cadenas en memoria.

<ctime> Contiene prototipos de función y tipos para manipular la hora y la fecha.

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 se emplean para representar elementos importantes 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 estándares. Pero en muchas situaciones es el programador el que debe proponer el identificador de un elemento de programa; para hacerlo en C++, deben tenerse presente 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 estándares)

ANSI/ISO C++ posee el siguiente conjunto de identificadores estándares 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*
Match	*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 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 Estándares de C++

El compilador de C++ reconoce tres tipos de datos estándares: enteros, punto flotante y carácter; a los que se agregan el tipo lógico y el Nulo.

Enteros: Los diferentes tipos de datos Enteros sirven para declarar números enteros dentro de los límites de cada uno de sus tamaños.

Enteros		
Tipo	**Rango**	**Tamaño (bytes)**
Char	--128 .. 127	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: Los diferentes tipos de datos Reales sirven para declarar números en formato de coma flotante, dentro de los límites de cada uno de sus tamaños. En síntesis, es para números con decimales.

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: El tipo char es un tipo básico alfanumérico, es decir que puede contener un carácter, un dígito numérico o un signo de puntuación, contendrá un único carácter del código ASCII. Hay que notar que en C un carácter es tratado como un número y esto es entendible si se recuerda que, según este código ASCII, por ejemplo, al número 65 le corresponde el carácter 'A' o al número 49 el '1'.
Este tipo de variables es apto para almacenar números pequeños, como la cantidad de hijos que tiene una persona, o letras, como la inicial de un nombre de pila.

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

Lógico: El tipo de datos bool sirve para declarar variables que sólo pueden tomar dos valores **true** (verdadero) o **false** (falso)

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

Nulo: El tipo de datos **void** es un tipo especial que indica la ausencia de tipo. Se usa, por ejemplo, para indicar el tipo del valor de retorno en funciones que no devuelven ningún valor.

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. Algunos ejemplos son los siguientes:

Tipo de constante literal	Ejemplos	Comentario
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
Cadenas	"Facultad"	Secuencia de caracteres que forman una cadena

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*

Al igual que en otros lenguajes como Pascal y Ada, 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, sólo 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 un tipo de dato enumerado, esto permite más adelante declarar una variable y asociarla al nombre del grupo. Esta variable podrá tomar alguno de los valores listados en al definición 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

Se estudió 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.).

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

Declaración y Definición: 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’
```

Ámbito de validez de una variable

Las variables también pueden clasificarse de acuerdo a su ámbito, es decir, la parte del programa en la que la variable es reconocida y puede ser utilizada. De acuerdo con su ámbito, las variables pueden ser locales o globales.
Las variables declaradas dentro de una función, y recordar que *main* también es una función, sólo serán accesibles para esa función, desde el punto en que se declaran hasta el final. Esas variables son variables locales o de ámbito local de esa función.
Al igual que ocurre con las variables locales de bucle, en las de función, las variables se crean al iniciar la función y se destruyen al terminar.
Las variables declaradas fuera de las funciones, serán accesibles desde todas las funciones definidas después de la declaración. Diremos que esas variables son globales o de ámbito global.
El ámbito temporal de estas variables es también global: se crean junto con el programa, y se destruyen cuando el programa concluye.
Una variable global declarada después de la definición de una función no será accesible desde esa función, por eso, normalmente se declaran las variables globales antes de definir las funciones.

El ámbito de validez de una variable es el 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>

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;
}
```

*Declaración y definición de **a***

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

*Salida: **Z 20***

*Causará error de compilación: ya que **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 sólo tiene el fin de mostrar el concepto de ámbito y alcance de las variables en C++.

Entrada y Salida (Input/Output)

Flujos de Entrada y Salida

Un flujo de Entrada/Salida o *I/O stream* es una 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 se dispone 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 debe incluirse este archivo en la cabecera del programa para que el compilador enlace las rutinas de definición necesarias e interprete las llamadas a estos flujos. Si no se utilizan archivos, el dispositivo predefinido para entrada y salida será el monitor de video.

Obsérvese el ejemplo del recuadro anterior donde la primer línea del código fuente indica la inclusión de este archivo**: `#include <iostream>`.** La directiva **#include** será estudiada más adelante, y permitirá enlazar código de otros archivos fuente C++ junto al programa que se esté desarrollando.

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>

int main(void)
{
  cout <<"Comando de flujo de salida en C++" ;
}
```

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

```
#include <iostream>

int main(void)
{
   int edad, anio_nac;
   cout << "Escriba su edad:" ;
   cin >> edad;
   anio_nac = 2010 – edad;
   cout << "\n";
   cout << "Ud. ha nacido en  " << anio_nac;
}
```

Caracter especial o secuencia de escape para producir un avance de línea.

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 con la instrucción "*include"*.

Ejemplo práctico: Estudiar y Analizar la salida de este programa. Investigar el efecto de los manipuladores utilizados.

```
using namespace std;
#include <iostream>
#include <iomanip>

int main(void)
{ cout << "Lenguajes de Programación";
  cout << endl<<endl;
  cout << setfill('.');
  cout << "1. Cobol" << setw(20)<< "pág. 1"<<endl;
  cout << "2. Fortran" << setw(20)<< "pág. 2"<<endl;
  cout << "3. Basic" << setw(20)<< "pág. 3"<<endl;
  cout << "4. Pascal" << setw(20)<< "pág. 5"<<endl;
  cout << "5. ANSI/ISO C++" << setw(20)<< "pág. 8"<<endl;
  return 0;
}
```

Conversiones entre tipos

C++ permite efectuar diversas conversiones entre sus tipos de datos.
En esta unidad, veremos las más simples, referidas a los datos numéricos de tipo entero y flotante.

Antes de visualizar un ejemplo práctico de conversión, revisaremos dos conceptos relacionados al truncamiento y redondeo de números.

Redondeo: Transformar un número decimal, al número entero más próximo.
Ej: 120,30 –REDONDEA—120
120,80 –REDONDEA—121
120,50 –REDONDEA—121

Truncamiento: Transformar un número decimal en un número entero, considerando la parte entera del mismo, sin importar los decimales que posea.
Ej: 120,30 –REDONDEA—120
120,80 –REDONDEA—120
120,50 –REDONDEA—120

Existen lenguajes de programación que consideran estas dos funciones. C++, no posee el *truncamiento* como función exclusiva. Pero si se desea obtener el resultado que brinda esta función, deberá utilizarse la conversión entre tipos. Con esto, en C++, podemos obtener los siguientes resultados:

```
int a,b,c;
float x,y,z;

a=100;
b=20;
c=a+b; // c, alojará el valor 120

x=100.80
y=20;
z=x+y; // z, alojará el valor 120.80

/* ahora sumaremos dos valores float, y alojaremos el resultado en una
variable int*/

c=x+y; // c, alojará el valor 120. Es decir, truncó los valores float
```