RESUMEN DE CONTENIDOS N° 6

INTRODUCCIÓN AL LENGUAJE C++

Estructura de un Programa C++

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

- **Include**
- **Macros**

Datos y Funciones Externas
Datos y Funciones Globales

```
main( )
{
  ….
  return 0;
}
```

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.

Documentación interna en C++

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++

Un archivo de encabezado “instruye” al compilador acerca de cómo interconectarse con los componentes de la biblioteca y los componentes escritos por el usuario.

<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.

<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++.

<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

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.

- 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**

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++

Enteros

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

Char - Unsigned char - Short - Unsigned short - Int - Unsigned int - Long - Unsigned long

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.
Son reales:

Float - Double - long double

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.

Lógico

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

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.

Notación y Definición de Constantes en C++

Constantes literales

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

123 -5 192.45 ‘A’ ‘\n’ “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

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";
```

Diferencia entre usar const y #define

Mediante const se declara una constante que tiene un tratamiento “similar” a una variable. Con define se indica que escribir el nombre especificado equivale a escribir el valor, con una correspondencia directa y sin tratamiento análogo al de una variable.

Por ejemplo: **const int FUTBOL5 = 5;** vs **#define FUTBOL5 5;**

En la primer declaración se indica que FUTBOL5 es una constante entera (int) que vale 5. En la segunda, cada vez que aparezca la palabra FUTBOL5, se reemplazará automáticamente por el valor 5.

Las constantes const tienen asociado un tipo de datos, mientras que las constantes define (macro) no.

Al utilizar const, el compilador puede realizar comprobaciones de seguridad de tipo, mientras que con define sólo realiza la sustitución de caracteres, no verifica y pueden ocurrir errores inesperados (efectos secundarios) en la sustitución de caracteres.

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 */***

Aquí, el compilador de C++ asigna un valor que empieza en cero a cada uno de los elementos enumerados, de esta forma, ene es igual a 0, feb a 1, mar es igual a dos y así sucesivamente hasta el elemento final. Luego de declarar un tipo de dato enumerado, se pueden crear variables de ese tipo, como cualquier otro tipo de dato, por ejemplo. (se ampliará al ver arrays)

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

En este ejemplo meses es un tipo de dato así como los tipos de datos int, char, float. Con la diferencia que este es un tipo de dato que toma 12 valores, los cuales fueron definidos como los meses del año, de manera abreviada.

Declaración e Inicialización de variables

Declaración:

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

Definición:

`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'
```

Entrada y Salida (Input/Output)

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.

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;
   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.

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

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

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;
}
```

Ámbito de validez de una variable

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;
}
```

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 –TRUNCA—120
120,80 –TRUNCA—120
120,50 –TRUNCA—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.

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
```