ORÍGENES DEL LENGUAJE C++.

Antes de comenzar a desarrollar las estructuras y la utilización del lenguaje C++, creí conveniente realizar algunos comentarios acerca de su origen.
Debemos saber que el lenguaje **C++** tiene sus orígenes en el **C**, el cual evolucionó de dos lenguajes de programación anteriores, el **BCPL** y el **B,** el lenguaje **BCPL** fue desarrollado por Martin Richards en 1967 como un lenguaje para escribir códigos para sistemas operativos y compiladores. Ken Thompson desarrollaría en 1970, basándose en el **BCPL** el lenguaje **B,** con el cuál crearía las primeras versiones del conocido sistema operativo **UNIX** en los laboratorios Bell. Tanto el **BCPL** como el **B** eran lenguajes de programación sin tipo, es decir, que cada dato declarado ocupaba una palabra en memoria, por ello el trabajo de procesar un elemento entero o real era tarea del programador.
El lenguaje **C** fue una evolución del **B** a cargo de Dennis Ritchie en 1972, el lenguaje utilizó conceptos importantes del **B** como del **BCPL,** y se hizo popular como lenguaje de programación y desarrollo de sistemas operativos. **UNIX** está programado en **C,** y la mayoría de los sistemas operativos de la actualidad se programan en **C** o **C++**. Otra de las características principales de **C/C++** es su amplia portabilidad, de hardware y software, esto quiere decir que un programa realizado en **C/C++** podrá ejecutarse tanto en una PC 286 como en una PC de última generación, también sobre distintos sistemas operativos con muy pequeñas variaciones.
Conocidos los orígenes del **C** podremos decir que **C++**, es un **C** mejorado, es común también nombrar a **C++** como una superestructura de **C,** desarrollado por Bjarne Stroustrup en los laboratorios Bell, **C++** proporciona a los programadores capacidades para una *programación orientada a objetos,* es por ello que se a convertido en uno de los lenguajes más populares en las instituciones de nivel superior, en las universidades así como en la industria.

PROGRAMACIÓN ORIENTADA A OBJETOS.

La programación orientada a objetos da a los programadores una forma distinta de enfocar los problemas, un nuevo ***paradigma de programación.*** Debido a la creciente complejidad que sufren los programas de la actualidad, el **C++** brinda una forma simple de resolver los problemas complejos.
Las principales características de los lenguajes de programación orientados a objetos es que brindan la posibilidad de trabajar con ***encapsulación, polimorfismo y herencia.*** A continuación comentaré brevemente cada uno de estos conceptos.

Partiendo desde la base de que un objeto es una entidad lógica que "*encapsula*" los datos y el código que manipula dichos datos. La ***encapsulación*** es el mecanismo que enlaza el código y los datos, y los asegura frente a agentes externos, de esta forma un objeto ofrece un nivel de seguridad frente a otras partes del programa que se podrían modificar accidentalmente. Se debe tener en cuenta que cuando se define un objeto se está creando un nuevo tipo de dato.

Por otra parte, el ***polimorfismo*** es la característica por la cual se reduce la complejidad de un programa, permitiendo utilizar una misma interfaz de software para especificar una clase general de acciones, se suele decir sobre el concepto de ***encapsulación*** que permite "*sobre una misma interfaz, múltiples métodos*", es tarea del compilador seleccionar la acción específica a utilizar (el método), que será aplicada a cada situación.

Por último, la ***herencia*** es el proceso por el cual un objeto puede "*heredar*" las propiedades de otro. Por ejemplo, podemos decir que un rectángulo es un cuadrilátero, como también lo es un cuadrado, de esta manera la clase rectángulo hereda de la clase cuadrilátero, a la clase cuadrilátero, se la llama "*clase base",* y a la clase rectángulo*, "clase derivada".*
La ***herencia*** forma estructuras jerárquicas en forma de árboles.
De esta manera la **herencia** nos permite la reutilización del software, mediante la creación de nuevas clases, en base a las existentes, al absorber sus atributos y comportamientos, redefiniendo o mejorando las nuevas clases. La ***herencia*** ahorra tiempo en el desarrollo de un programa ya que permite, como se comentó, la reutilización del software.

CONCEPTOS PRELIMINARES, COMENZANDO A PROGRAMAR.

Un programa, es una secuencia ordenada de tareas a seguir, estas tareas o instrucciones deben ser ingresadas a la computadora y traducidas para que la misma pueda entenderlas, la tarea de traducir estas instrucciones la lleva a cabo el ***compilador***, para facilitar el ingreso de esta tarea, contamos con una interfaz de desarrollo, en la cual, además de ingresar código con la sintaxis de **C/C++,** podremos incluir algunas líneas de código que no serán interpretadas por el compilador y que nos ayudarán en la futura comprensión del programa. A estas líneas de código las llamaremos "*comentarios"***,** por ejemplo:

```
/* Esto es un comentario */

// Esto también es un comentario.
```

Es importante destacar que el lenguaje es "*Keysensitive",* es decir que se diferencian los caracteres mayúsculas de minúsculas. Por ejemplo:

```
int a; // se declara una variable entera llamada a.

int A; /* la variable declarada es A, pero en mayúsculas, se
          trata de dos variables distintas */
```

EL PRE-PROCESADOR.

La principal utilidad del preprocesador es incluir cabeceras o *headers.* La idea es que cuando se desea llamar a funciones que se construyen a partir del lenguaje, es decir de la biblioteca estándar. Se utiliza ***#include*** para incluir los archivos que contienen las funciones que definen las tareas a realizar. Por ejemplo:

```
#include <iostream> /* imprime por pantalla y lee datos desde
el teclado, mediante la salida y la entrada estándar, cin>> y
cout<<, respectivamente */

#include <conio.h> //por ejemplo cuando se desea utilizar la
                   //función getch().
```

ENTRADA Y SALIDA ESTÁNDAR.

En **C++** contamos con los objetos ***cout***<< *y* ***cin***>>**,** con sus respectivos operadores, estas instrucciones se encuentran dentro de la biblioteca ***iostream***.

Ejemplo de salida por pantalla.

```
#include <iostream>

 using namespace std; //se declara el espacio de nombres.
  int main(){              // función principal.
    cout<<"Hola Mundo";     // Aquí se realiza la salida.
    return 0;              //La función devuelve 0.
  }
```

El operador <<, llamado de inserción, le dice al sistema que imprima la variable o la cadena de caracteres que le sigue, en este caso "*Hola Mundo*".

Ejemplo de ingreso por teclado.

```
#include <iostream>

 using namespace std;
 int main(){       // función principal.
   int x;           // se declara la variable entera x.
   cout<<"Ingrese un valor"; // se muestra una leyenda
                             // "ingrese un valor".
   cin>>x;                   // se pide el ingreso de x.
   cout<<"el valor ingresado es: "<<x;  // se informa el valor
                                        // ingresado.

   return 0;           //la función retorna 0.
}
```

El operador >>, llamado de extracción, es obviamente el opuesto de <<, el operador de extracción, toma los datos de ***cin*** y los asigna a la variable ***x.***

OPERADORES DE TIPO.

Observe que en el ejemplo anterior se declaró una variable de tipo ***int,*** este tipo de dato solo aceptará el ingreso de un dato entero, dependiendo de la aplicación o el problema que deseemos resolver podremos utilizar tipos distintos, los posibles tipos de **C++** y sus rangos son los siguientes.

Tipo de dato	Tamaño en bits	Rango
char	8	-128 a 127
bool	8	true - false
int	16	-32768 a 32767
float	32	3,4E -38 a 3,4E+38
double	64	1,7E -308 a 1,7E+308

MODIFICADORES DE TIPO.

Los modificadores de tipo se utilizan para alterar el significado del tipo por defecto para que se ajuste de manera más precisa a la situación del problema a resolver. Los modificadores de tipo que podremos utilizar son los siguientes:

signed ***unsigned*** ***long*** ***short***

Los modificadores ***signed, unsigned, long, short*** se pueden aplicar a los tipos de enteros base, se puede aplicar los modificadores ***signed, unsigned*** a los del tipo char. También se puede aplicar el modificador de tipo ***long*** al tipo ***double.***
Se muestran en la siguiente tabla algunos ejemplos:

Tipo de dato modificado	Tamaño en bits	Rango
unsigned char	**8**	0 a 255
signed char	**8**	-128 a 127
unsigned int	**16**	0 a 65535
signed int	**16**	-32768 a 32767
short signed int	**16**	-32768 a 32767
unsigned long int	**32**	0 a 4294967295
signed long int	**32**	-2147483648 a 2147483647
long double	**80**	3,4E-4932 a 3,4E4932

DECLARACIÓN DE VARIABLES, LOCALES Y GLOBALES.

Sabemos que al declarar una variable estamos reservando espacio en memoria para almacenar uno a más datos del tipo declarado, ahora bien, ¿Cuál es el ámbito de la variable declarada?, esto lo determinaremos dependiendo si la variable es *global* o *local.*
Las variables locales tienen validez únicamente dentro de la función donde fueron declaradas, mientras que las variables globales la tienen en todas las funciones que se utilizan dentro del programa. Una variable local se declara dentro de la función donde se utilizará, una variable global se debe declarar fuera de cualquier función, incluso fuera de *main().*
Ejemplo de declaración de variables.

Variable global.

```
#include <iostream>
   using namespace std;
   float i=10.5;
    int main(){
      cout<<”La variable declarada i es global, su valor es”<<i;
      return 0;
     }
```

Variable local.

```
#include <iostream>
   using namespace std;
   int main(){
      int x=11;
      cout<<”La variable x es local, tiene validez únicamente
            dentro de main(),”<<“su valor es: ”<<x;
      return 0;
    }
```

DECLARACIÓN DE CONSTANTES.

Una constante es un dato declarado con un tipo determinado que no puede ser modificada a lo largo del programa.

Ejemplo:

```
const int num=14; // En este caso num valdrá 14 durante todo el
                  // programa.
```

Otra forma de declarar una constante es por medio de una macro, la instrucción ***#define*** modifica la mima mediante el preprocesador, por ejemplo:

```
#define NUM 0x378 //Aquí se define en NUM un valor en hexadecimal.
```

OPERADORES ARITMÉTICOS.

Ahora que sabemos imprimir datos en pantalla e ingresárselos a una variable, veamos las operaciones que podemos realizar entre ellos.
Los operadores aritméticos básicos utilizados en **C++** son los mismos que se utilizan en la mayoría de los lenguajes de programación, se muestra a continuación una tabla con ellos.

Operador	Acción que realiza
-	Resta, también signo - unario
+	Suma, también signo + unario
*****	Producto, multiplicación
/	Cociente, división
%	Módulo de la división (devuelve el resto entero)
--	Decremento
++	Incremento

Los operador aritméticos serán utilizados para realizar operaciones entre valores, a continuación se muestran algunos ejemplos.

```
#include <iostream>
     using namespace std;
        int main(){
          int N1,N2,suma=0,resta=0,cociente=0,producto=0;
          N1=10;
          N2=2;
          suma= N1+N2;
          resta=N1-N2;
          cociente=N1/N2;
          producto=N1*N2;
          cout<<"Resultados"<<endl;
          cout<<"Suma:"<<suma<<endl<<"Resta:"<<resta<<endl<<
          "Cociente:"<<cociente<<endl<<"Producto:"<<producto;
          return 0;
         }
```

Pre-incrementro y pos-incremento.

Los operadores **++** y **- -** realizan el incremento o decremento de un valor, pero es de considerar que:
En el caso de que los operadores se utilicen:

```
X++; // A esto se lo denomina pos-incremento.
++X; // Esto es pre-incremento.
X--; // A esto se lo denomina pos-decremento.
-- X // Esto es pre-decremento.
```

Ejemplo:

```
#include <iostream>
  using namespace std;
  int main(){
     int N1,N2;
     N1=5;
     N2=N1++;  // Es lo mismo que si se escribiera N2=N1+1.
     cout<<"El valor de N1 es: "<<N1;  // El valor de N1 es 5.
     cout<<"El valor de N2 es: "<<N2;  // El valor de N2 es 6.
     return 0;
  }
```

El operador **%** pude utilizarse para calcular el resto de una división entre enteros, por ejemplo:

```
#include <iostream>
  using namespace std;
  int main(){
     int N1,N2,resto=0,resultado=0;
     N1=11;
     N2=2;
     resultado=N1/N2;
```

```
        resto=N1%N2;
        cout<<"El resultado de la división es: "<<resultado<<" y
              el resto es: "<<resto;
        return 0;
}
```

En este caso el compilador informará 5 para el resultado y 1 para el resto debido a que estamos trabajando con números enteros.

OPERADORES RELACIONALES.

Los operadores relacionales se refieren a las relaciones que pueden tener unos valores con respecto a otros, la tabla que representa a dichos operadores es la siguiente:

Operador	Acción que realiza
>	Mayor que
>=	Mayor o igual a que
<	Menor que
<=	Menor o igual que
==	Igual
!=	Distinto

OPERADORES LÓGICOS.

La clave de la utilización de los operadores lógicos es la idea de **verdadero** o **falso,** trabajando con lógica positiva, será **verdadero**, cualquier valor distinto de 0, y será **falso** el 0. Los operadores relacionales y lógicos tienen menor precedencia que los operadores aritméticos, es decir, menor prioridad de cálculo.
Es importante destacar que en **C++** pueden realizarse operaciones lógicas tanto a nivel de **Byte** como a nivel de **Bits**, pero los operadores difieren entre ellas.
Se muestra a continuación la tabla de operadores lógicos a nivel de **Byte** y a nivel de **Bits.**

OPERADORES A NIVEL DE BYTE	
Operador	**Acción que realiza**
&&	Y lógica (AND)
||	O lógica (OR)
!	NO, negador (NOT)
OPERADORES A NIVEL DE BITS	
Operador	**Acción que realiza**
&	Y lógica (AND)
|	O lógica (OR)
^	O exclusivo (XOR)
~	Complemento a 1
>>	Desplazamiento a la derecha
<<	Desplazamiento a la izquierda

Ahora estamos listos para comenzar a programar, pero antes de esto, veremos algunos códigos, llamados de barra invertida o contrabarra, que son muy útiles a la hora de desarrollar un programa.

TABLA DE CÓDIGOS DE BARRA INVERTIDA.

Código	Significado
\b	Retroceso
\f	Alimentación de hoja
\n	Nueva línea
\r	Retorno de línea
\t	Tabulador horizontal
\"	Comillas doble
\'	caracter comilla simple
\0	caracter nulo - fin de cadena de caractéres
****	Inserta al texto la barra invertida
\v	Tabulador vertical
\a	Alarma del sistema - emite un Beep
\N	Constante octal (N es un valor octal)
\xN	Constante hexadecimal (N es un valor en hexadecimal)

PALABRAS RESERVADAS.

Cuando se escribe un programa utilizando la sintáxis de **C/C++** hay ciertas palabras que no se pueden utilizar como nombre de una variable por ser propias del lenguaje, a continuación se muestran dichas palabras:

Palabras reservadas de C y C++.

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

Palabras reservadas en de C++ estándar.

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

INSTRUCCIONES DE CONTROL DE PROGRAMA, PROGRAMACIÓN ESTRUCTURADA.

Por lo general, las instrucciones dentro de un programa se ejecutan de manera secuencial, es decir una tras otra todas las líneas de código, a esto se lo llama *"ejecución secuencial",* varias instrucciones de **C/C++** permiten al programador especificar que la siguiente instrucción en ejecutarse debe ser otra y no la siguiente en la secuencia, a esto se le llama "*transferencia de control".*
Todos los programas pueden escribirse en términos de tres estructuras, ***la estructura secuencial, la estructura de selección y la estructura de repetición,*** A continuación se desarrollarán estas estructuras.

ESCTRUCTURA CONDICIONAL O DE SELECCIÓN.

La instrucción de selección *if…else*

La instrucción ***if… else*** nos permite la posibilidad de evaluar una expresión, la cual, si es verdadera nos permitirá ejecutar un determinado proceso, o si es falsa, se podrá ejecutar un proceso distinto. La sintáxis de la instrucción ***if…else*** es la siguiente:

```
if (expresión o condición){
    instrucciones por verdadero;
}
else{
    instrucciones por falso;
}
```

Ejemplo:

Se ingresan 4 números enteros desde teclado, se realiza y se informa la sumatoria de los números positivos ingresados y el producto de los negativos. Compilador GCC.

```
#include <iostream>
 using namespace std;
 int main(){
  int N,suma=0,prod=1;
  cin>>N;
   if(N>0){ //Se detecta si el N ingresado es negativo.
    suma=suma+N; //Si la expresión es verdadera, se suma el N.
              }
   else{ // Si la expresión es falsa.
    prod=prod*N;  // Se realiza el producto de los N.
   }
  cin>>N;
   if(N>0){
     suma=suma+N;
   }
   else{
     prod=prod*N;
```

```
      }
       cin>>N;
        if(N>0){
          suma=suma+N;
        }
        else{
          prod=prod*N;
        }
         cin>>N;
            if(N>0){
              suma=suma+N;
            }
            else{
              prod=prod*N;
            }
    cout<<"La sumatoria de los positivos es: "<<suma<<endl; /* Se
     informa la Sumatoria.*/
    cout<<"El producto de los negativos es: "<<prod; /* Se informa
     el  producto.*/
    return 0;
   }
```

Otro ejemplo:

Se ingresan 3 números enteros desde el teclado, informar cuantos de ellos se encuentran en el rango 100-200. Compilador GCC.

```
      #include <iostream>
        using namespace std;
        int main(){
           int N, contarango=0;
           cin>>N;
             if((N>=100)&&(N<=200)){
                 contarango++;
             }
           cin>>N;
             if((N>=100)&&(N<=200)){
                 contarango++;
             }
           cin>>N;
             if((N>=100)&&(N<=200)){
                 contarango+
             }
          cout<<"Los números que se encuentran dentro del rango
                 son:"<<contarango;
          return 0;
         }
```

Observar en el ejemplo anterior que se establece una operación lógica AND entre las dos condiciones, es decir **N > 100 y N < 200** como ambas condiciones terminan siendo una,

vinculadas por la operación AND, las dos van entre paréntesis. Podemos dejar en claro que todas las condiciones en **C/C++** se escriben entre paréntesis.

Instrucción de selección *if…else if*

Una manera de poder evaluar múltiples expresiones es mediante la instrucción ***if…else if,*** la sintáxis de esta instrucción es la siguiente:

```
if (condición){
   instrucciones por verdadero;
}
else if(nueva condición){
   instrucciones por verdadero de la nueva condición;
}
else{
   instrucciones por falso de la nueva condición;
}
```

Instrucción de selección múltiple *switch…case*

La siguiente instrucción condicional, nos permite realizar evaluaciones múltiples, el formato general de la instrucción ***switch…case*** es la siguiente:

```
switch(expresión ){
    case constante 1:
         secuencia de instrucciones;
         break;
    case constante 2:
         secuencia de instrucciones;
         break;
    case constante n:
         secuencia de instrucciones;
         break;
    default:
         secuencia de instrucciones;
 }
```

La instrucción ***default*** (por defecto), se ejecuta si no se encuentra coincidencia con ninguna de las anteriores, ésta puede utilizarse, o no.

Ejemplo:

El siguiente programa ejecuta una calculadora simple, mediante las opciones 1, 2, 3 o 4 se puede seleccionar la operación a realizar entre dos N reales. Compilador GCC.

```
#include <iostream>
  using namespace std;
  int main(){
     int seleccion;
```

```
        float a, b, resultado=0;
        cout<<"Ingrese un N: ";
        cin>>a;
        cout<<"ingrese otro N";
        cin>>b;
        cout<<"MENU DE OPCIONES"<<endl<<endl;
        cout<<"seleccione una opción: "<<endl;
        cout<<"  1---------------- Sumar"<<endl;
        cout<<"  2---------------- Restar"<<endl;
        cout<<"  3---------------- Dividir"<<endl;
        cout<<"  4---------------- Multiplicar"<<endl;
        cin>>seleccion;
         switch(seleccion){
            case 1:
                  resultado=a+b;
                  break;
            case 2:
                  resultado=a-b;
                  break;
            case 3:
                  resultado=a/b;
                  break;
            case 4:
                  resultado=a*b;
                  break;
         }
         cout<< "El resultado es: "<<resultado;
         return 0;
    }
```

ESTRUCTURAS DE REPETICIÓN.

Una instrucción de repetición permite al programador especificar que una acción se repita mientras una condición sea verdadera, a esta repetición se la denomina **iteración.**
En **C/C++** existen tres instrucciones de repetición, ***for, while*** y ***do…while.*** Cada una con características propias, comenzaremos nuestra explicación por la instrucción ***for.***

Instrucción de repetición *for*.

La sintáxis de esta instrucción es la siguiente:

```
for( inicio; condición; incremento){
     secuencia de instrucciones;
}
```

Inicio: Aquí se puede especificar el comienzo de la instrucción de repetición.
Condición: Esta es la condición de ejecución, es decir, que el ciclo se ejecutará mientras la condición sea verdadera.
Incremento: Este ítem, hace referencia al incremento, o decremento que sufrirá la variable utilizada en el ciclo de repetición.

Ejemplo:

El siguiente programa permite visualizar en pantalla un conteo de 0 a 9 debiéndose pulsar una tecla para ver el incremento. Compilador GCC.

```
#include <iostream>
#include <conio.h>
  using namespace std;
  int main(){
     int i;
     for(i=0; i<10; i=i+1){ /* inicio i=0; condición mientras
                               sea < a 10; incremento 1 */
       cout<<i;
       getch(); /*espera la pulsación de una tecla y no
                  devuelve su valor*/
       system("cls");
      }
    return 0;
  }
```

Programa que calcula el promedio de números pares, de una lista de 10, ingresados desde el teclado. Compilador GCC.

```
#include <iostream>
    using namespace std;
    int main(){
     int num, cuentapar=0, sumapar=0;
     float promedio=0;
      for(int i=1; i<=10; i++){ /*Se ejecuta el ciclo con 10
                                  iteraciones*/
       cout<<"Ingrese el "<< i <<"primer número: ";
       cin>>num;
         if((num % 2) == 0){ // Se buscan los N pares.
          cuentapar = cuentapar+1; //Si el N es par se lo cuenta.
          sumapar = sumapar + num; // Si el N es par se lo suma.
        }
      }
    promedio =sumapar/cuentapar;  /*Se calcula el promedio de los
                                    N pares */
    cout<<"El promedio de los números pares es: "<< promedio.
    return 0;
   }
```

Instrucción de repetición *while.*

A diferencia de la instrucción de repetición ***for,*** el ***while*** permite la ejecución del ciclo mientras se cumpla una condición, se utiliza cuando se desea iterar mientras una expresión sea verdadera, nótese la diferencia con el ***for*** ya que el mismo se utiliza cuando tenemos las condiciones de inicio y finalización del proceso a realizar. Un ejemplo de utilización del ciclo ***while*** estaría dado por el siguiente problema.

La sintáxis de la instrucción ***while*** es la siguiente:

```
while (condición){
     secuencia de instrucciones;
}
```

Se pide ingresar el número de mes en curso, sabiendo que el año solo cuenta con 12 meses debemos validar el ingreso dentro de ese rango. Compilador GCC.

```
# include <iostream>
   using namespace std;
   int main(){
      int mes;
      cout<<"Ingrese el número de mes en curso:  "
      cin>>mes;
        while ((mes<1)||(mes>12)){ /* se valida el rango para
                                      el mes entre 1 y 12*/
         cout<<"Dato para el mes no válido, ingrese otro: ";
         cin>>mes; /* Si el dato es erróneo se pide el
                   reingreso*/
        }
      cout <<"Mes ingresado: "<<mes;
      return 0;
  }
```

La instrucción ***while*** puede interpretarse como "*hacer mientras, la condición sea verdadera*".

Otro ejemplo.

Se ingresa el sexo de N personas, (f=femenino, m=masculino). Para finalizar los ingresos "x"; informar porcentaje de personas de sexo masculino. Compilador GCC.

```
#include <iostream>
   int main(){
      char sexo;
      int cuentamasculino=0,cuentaingresos=0;
      float porcentaje=0;
        while (sexo!='x'){ /*Hacer mientras sexo sea distinto
                           de x */
         cout<<"Ingresar sexo [f/m], para salir [x]:  ";
         cin>>sexo;
          while ((sexo!='f' )||(sexo !='m')){ /* hacer
                  mientras sexo sea distinto de f o m */
           cout<<"Ingreso no válido, f=femenino, m=masculino: ";
           cin>>sexo; // Se pide el reingreso del sexo.
          }
            if(sexo=='m'){ // Se buscan los de sexo masculino
              cuentamasculino=cuentamasculino+1; /*Se cuentan
                                          los masculinos */
          }
```

```
            cuentaingresos++; // Se cuentan los ingresos.
          }

          porcentaje=(cuentamasculino*100)/cuentaingresos;
          cout<<"El porcentaje de masculinos es: "<<porcentaje;
          return 0;
        }
```

Instrucción de repetición *do…while.*

La sintáxis de la instrucción ***do…while*** es la siguiente:

```
          do {
              secuencia de instrucciones;
          } while (condición);
```

Puede observarse en base a la sintáxis de la instrucción de repetición ***do…while*** que a diferencia de la instrucción ***while*** la condición se analiza al final, por lo tanto, se ejecutará siempre por lo menos una vez, esto es muy útil a la hora de programar menú de opciones.

Por ejemplo:

Se ingresan números desde el teclado mientras no sean menores a 100. Compilador GCC.

```
#include <iostream>
   int main(){
       int num;
          do{
            cout<<"Ingrese números mayores a 100: ";
            cin>>num;
          }while (num > 100); /* Se analiza la  condición
                                 número > 100 */
          return 0;
   }
```

Nótese que auque se piden números mayores a 100, el ciclo se ejecutará por lo menos una vez de manera de ingresar por lo menos un número auque el mismo sea menor a 100.

BIBLIOGRAFÍA CONSULTADA

Kernighan, Brian W. Ritchie Dennis M. (1991). *"El lenguaje de Programación C"*. Pearson Educación. México.

Deitel & Deitel. (2004). *"Cómo Programar en C/C++ y Java*". Pearson. Prentice Hall. México.

Gottfried Byron. (2005). *"Programación en C – Serie Schaum".* McGraw Hill. España.

Schildt Hebert. (1995). *"Borland C++ Manual de Referencia"*. McGraw-Hill. España.

Stroustrup Bjarne. (2002). *"El Lenguaje de Programación C++".* Pearson Educación. México.

Walter Savitch. (2007). "*Resolución de Problemas con C++*". Pearson. Addison Wesley. México