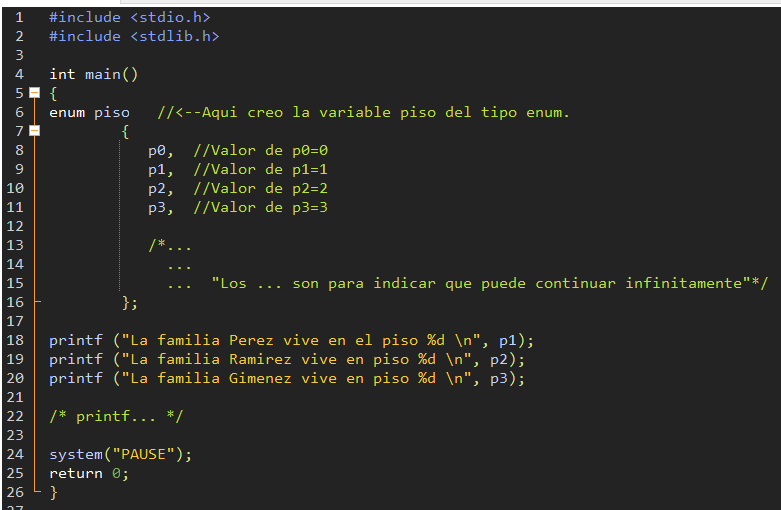
   Buenos días alumn@s, para estas alturas ya habrán estado jugando con las variables y estaremos en el punto en que abrimos podido resolver problemas con decimales y enteros, dando respuestas con dos, tres, o "n (infinitos)" números después de la coma. Podemos entender leyendo el manual que cada tipo de asignación por más que, por ejemplo, trabajemos en variables de valores decimales, no es lo mismo un valor del tipo float (4bit) que un valor del tipo double (8bit).

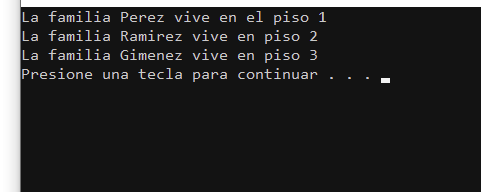
     Lo que le vamos a ver a continuación son funciones a las cuales le vamos a dar un tratamiento especial. En este caso el enunciado **enum**, este tipo de variable resulta muy interesante, tanto para realizar caracterización de valores numéricos, así como asignaciones de un valor de texto del tipo string que se encuentra ordenado.

FUNCIÓN ENUM:

     En el *ejemplo*del tema 4 la función enum, enumera los días de la semana y como verán le asigna automáticamente un orden de valores de 0 a 6 para cada día, para clarificar aún más este tipo de enunciado los invito a realizar el siguiente ejercicio:

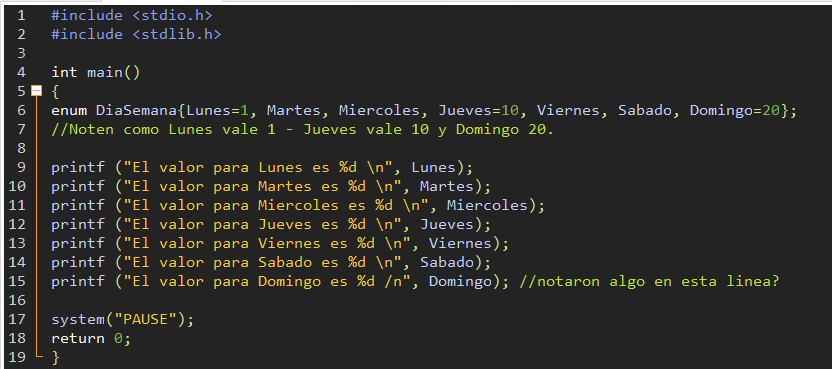


     Obtendremos el siguiente resultado:



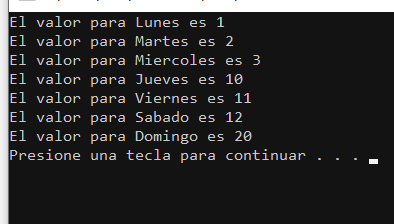
     Como se puede observar al realizar el ejercicio, el número de piso es asignado directamente con la función **enum**y lo podemos repetir las veces que sea necesario en la salida ya que dentro del main el parámetro esta asignado.

***Ejemplos clásicos de uso de la función Enum:***



     Copien este ejercicio y vean como también puedo utilizar la asignación de valores para las variables creadas en enum y a partir de ellas comenzar a contar.

     Deberíamos obtener algo similar a esto:



     Noten como nuestro enumerador dado el valor lunes = 1, cuenta a martes como 2 y así sucesivamente hasta que aparece jueves = 10, donde entonces viernes = 11 y Sábado = 12, por último, aparece el valor asignado a Domingo que es 20 y ya no hay más días para contar.

     Solo para tener una ***mayor comprensión*** en la siguiente expresión queda demostrado como se puede definir una función enum y como una vez nombrada puedo reutilizarla bajo otro nombre en el programa.

enum colores

{ azul, amarillo, rojo, verde, blanco, negro } ; //<-- Recuerden que los valores iran desde el 0 al 5

typedef enum colores t\_colores;  //<-- Aquí define colores como t\_colores para utilizarla en el programa

// Uso mis declaraciones dentro del main.

main()

{

t\_colores color;

*//*...

}

------------------------------------------------------------------------------------------------------------------------------

La declaración del tipo *t\_colores*puede realizarse también así:

typedef enum colores {azul, amarillo, rojo, verde, blanco, negro}  t\_colores;

------------------------------------------------------------------------------------------------------------------------------

o simplemente así:

typedef enum {azul, amarillo, rojo, verde, blanco, negro,}  t\_colores;

**------------------------------------------------------------------------------------------------------------------------------**

     Más adelante segun vayamos viendo funciones como if o while, esta función enum, tomará una utilidad mayor para la obtención de soluciones según lo vayan solicitando los programas.

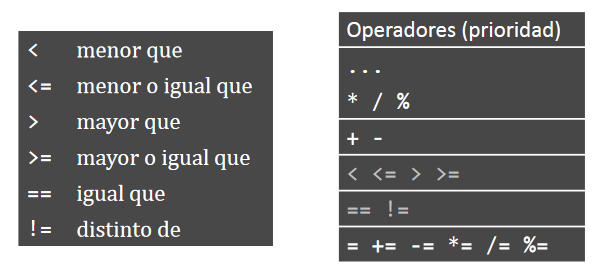
FUNCIÓN BOOL:

     Vamos a ver ahora como funcionan las expresiones del tipo **BOOLE** que nos entregan como resultado 0 o 1, lo cual nos resolverá muchísimos problemas al paso del tiempo. Este tipo de resultados nos permitirá, por ejemplo: arrojar resultados verdaderos o falsos, positivos o negativos, correctos o incorrectos, es muy útil y el resultado resuelve eventualmente situaciones que deben cumplir muchas condiciones con una sola respuesta.

     En C++, el tipo de datos bool se usa para representar datos booleanos (lógicos). Por ello, este tipo de datos está restringido a uno de dos valores: verdadero o falso. Este tipo de datos es más útil cuando un programa debe examinar una condición específica y, debido a la condición de ser verdadera o falsa, tomar un curso de acción prescrito. Por ejemplo, en una aplicación de ventas, la condición que se está examinando podría ser “la compra total es por $100 o más”. Sólo cuando esta condición es verdadera se aplica un descuento. Sin embargo, debido a que un tipo de datos booleano usa un código de almacenamiento de enteros, tiene implicaciones útiles explotadas por casi todos los programadores profesionales en C++.

     Para comenzar a ver algunas relaciones simples del tipo booleanas, podemos seguir los siguientes ejemplos:

Operadores relacionales  
Comparaciones (condiciones)  
Condición simple ::= Expresión  Operador relacional Expresión

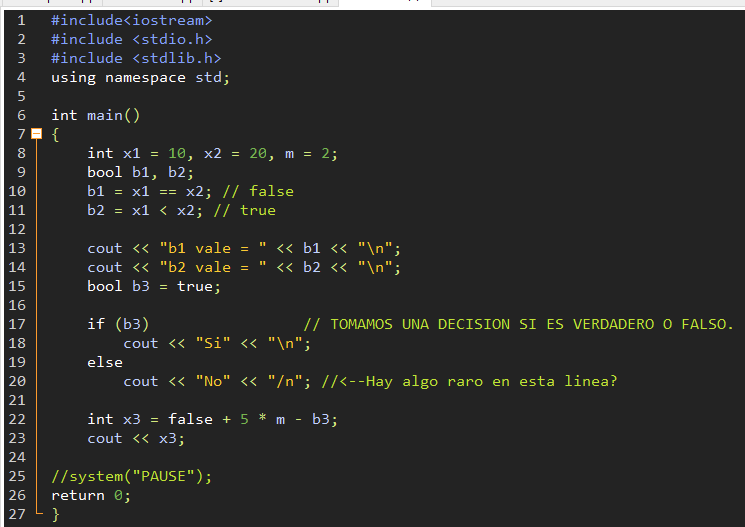
     Concordancia de tipo entre las expresiones  
Resultado:  
bool(true o  false)  
  
  


     Menor prioridad que los operadores aditivos y multiplicativos

bool resultado;  
int a = 2, b = 3, c = 4;  
resultado = a <5;                      // 2 < 5  [true]  
resultado = a \* b + c >= 12;     // 10 >= 12  [false]  
resultado = a \* (b + c) >= 12;   // 14 >= 12 [true]  
resultado = a != b;                    // 2!= 3 [true]  
resultado = a \* b >c + 5;           // 6 > 9 [false]  
resultado = a + b == c + 1;       // 5 == 5 [true]

     Más adelante el tipo BOOL nos va a servir para responder rápidamente a ecuaciones que deriven en decisiones o ciclos de bucles del estilo:

**Ejemplo 1 bool:**



     En este ejemplo vemos algunas cosas interesantes, en C++ además de la salida estándar "printf" también tenemos la posibilidad de mostrar contenido utilizando la función "std::cout", pero para no tener que anteponer "std::" cada vez que voy a utilizar "cout" al comenzar el programa voy a agregar como se ve en la línea 4 de código “**using namespace std;**” de esta forma omitiremos la necesidad de nombrar toda la sintaxis y la resumiremos directamente a **cout<<”texto”<<endl;**.

     Por otro lado, fíjense la línea 22 de código creamos la variable entera x3 “int x3 = false + 5 \* m - b3;” noten como utilizamos los valores booleanos de false (0) y de true (1), este último es el equivalente a la variable b3, para poder resolver el valor de x3. Copien todo el ejercicio, no se preocupen por el if y else que serán vistos mas adelante en la materia, y ejecuten el programa para ver los resultados, OJO hay algunas cosas para corregir, tal vez algo más que una pausa nos falte para poder ver correctamente el contenido del programa.

     Veamos ahora como una operación lógica utiliza como operando datos booleanos. La mayoría de ellas compara dos expresiones o datos booleanos e indica una posibilidad de lo que puede ocurrir cuando se juntan. El resultado es otra expresión booleana.

     En la página anterior hemos visto las operaciones condicionales, las cuales daban como resultado expresiones booleanas. Por lo tanto, también podemos poner como operandos de las operaciones lógicas el resultado de una operación condicional. Es decir, cada uno de los operando es una operación condicional, la cual se realiza en primer lugar, para después realizar las operaciones lógicas. Sin embargo, entenderemos esto más fácilmente cuando veamos cuales son las operaciones lógicas.

**Operador AND**

     Esta operación compara dos datos booleanos, de manera que el resultado será "true" (verdadero) solamente si los dos operando son "true" (verdadero); en los demás casos el resultado será "false" (falso).

     El signo para realizar la operación es && (doble signo ampersand) pero también podemos poner la palabra and. Por ejemplo, declaramos las      siguientes variables booleanas.

     bool v1=true, v2=true, f1=false, f2=false, r;

     Las variables que tenemos inicializadas las utilizaremos como operando, mientras que la variable r, sin inicializar, será donde obtengamos el resultado, operamos así:

     r = v1 && v2;

     El resultado será true. La misma operación la podemos escribir así:

     r = v1 and v2;

     Aunque es preferible ponerlo como en el primer ejemplo.

     Los posibles valores que puede tener el resultado son los siguientes:

     r = v1 && v2; *// r: true*

     r = v1 && f1; *// r: false*

     r = f1 && v1; *// r: false*

     r = f1 && f2; *// r: false*

     Aquí el resultado solo es verdadero si los dos operandos son verdaderos.

**Operador OR**

     Este operador compara dos datos booleanos, de manera que el resultado será "false" (falso) solamente si los dos operadores son falsos; en los demás casos el resultado es "true" (verdadero).

     El signo para realizar la operación consiste en la doble barra vertical || (en la mayoría de los teclados se escribe con las teclas "Alt Gr" + 1). También tenemos la opción de escribir la palabra or entre los operando.

     Los posibles valores que puede tener el resultado son los siguientes: (utilizaremos aquí las mismas variables que en el ejemplo con and)

     r = v1 || v2; *// r: true*

     r= v1 || f1; *// r: true*

     r = f1 || v1; *// r: true*

     r = f1 || f2; *// r: false*

    Aquí el resultado solo es falso si los dos operando son falsos.

**Operador NOT**

     Este es un operador unario, es decir se aplica a un solo elemento que será un dato booleano, y lo que hace es cambiarlo de valor. De manera que si el dato es verdadero el resultado es falso, y si el dato es falso, el resultado es verdadero.

     El operando es un signo de admiración puesto delante del dato o variable que contiene el dato. Siguiendo con las mismas variables de ejemplos anteriores.

     r =! v1; // *r: false*

     r =! f1; // *r: true*

     Aquí el resultado siempre es el contrario del que tenemos al principio.

**Operador XOR**

     Este operador compara dos variables booleanas, y devuelve true cuando una de las dos es verdadera true y la otra falsa false. Cuando las dos son verdaderas, o las dos son falsas devuelve false.

     El signo del operador es el acento circunflejo ^, aunque también podemos escribir la palabra xor entre los dos operandos (El acento circunflejo lo escribimos con mayúscula + acento grave, y después la barra espaciadora).

     Los posibles valores que puede tener el resultado son los siguientes: (utilizaremos aquí las mismas variables que en ejemplos anteriores)

     r = v1 xor v2; *// r: false*

     r = v1 xor f1; *// r: true*

     r = f1 xor v1; *// r: true*

     r = f1 xor f2; *// r: false*

     Aquí el resultado es falso cuando los dos operandos son iguales, y verdadero cuando son contrarios.

**Operador condicional ternario:**

     Este operador utiliza tres operandos, el primero es siempre un dato o expresión booleana. El segundo y el tercero pueden ser datos de cualquier otro tipo, siempre que los dos sean del mismo tipo. Supongamos que declaramos las siguientes variables:

     bool tr=true, fs=false;

     string vr="verdad", mt="mentira", resultado;

     Ahora utilizamos el operador condicional ternario. Lo escribiremos así:

     resultado = tr ? vr : mt;

     El primer operando es tr que es un booleano, el segundo y el tercero aquí son strings, pero pueden ser datos de cualquier otro tipo, siempre que los dos sean del mismo tipo. El resultado será el segundo operando siempre que el booleano del primero sea true, y el tercero cuando el booleano sea false. En el ejemplo anterior recogemos el resultado en la variable resultado.

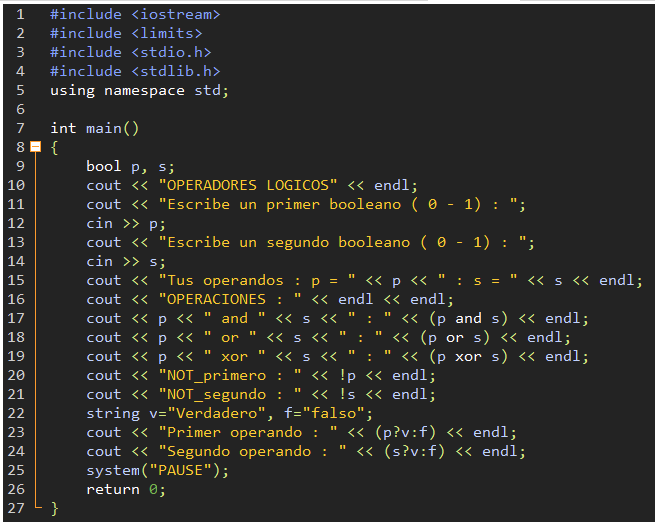
     Si seguimos el ejemplo anterior, se puede comprobar mostrando el resultado en pantalla:

     cout << resultado << endl;

     El resultado al compilar y ejecutar, dar la palabra "verdad". Si cambiamos el primer operando (variable tr) por la variable fs, veremos que el resultado es la palabra "mentira".

***Ejemplo BOOLEANO con OPERADORES:***

     Veamos un programa en el que utilizaremos valores booleanos en conjunto con los operadores lógicos.



    En este simple ejemplo vamos a poder corroborar todos los valores posibles de los operadores que hemos visto anteriormente, lo primero sera crear 2 variables "p" y "s" del tipo bool (booleano) para luego ingresar un valor 0 o 1 y asi lograr que el programa resuelva las ecuaciones mostrandonos los resultados con and/or/xor el resultado inverso a "p" y el inverso a "s" y por ultimo el valor que ha tomado nuestra variable booleana.  
    Copien el ejercicio ingresen 0 o 1 y vean los resultados obtenidos.

\*\*Para aclarar dudas y dialogar sobre como será la materia me gustaría que nos encontráramos el HOY lunes 3 de abril a las 20:00 hs. a 20:30 hs. en un meet (por favor sean puntuales):

meet.google.com/fyi-zyxz-xho

El encuentro **NO**es de carácter obligatorio, pero si servirá para que nos podamos conocer y además puedan entender la dinámica de la materia. Les dejo un saludo y seguimos en contacto. Recuerden que hay un FORO DE CONSULTAS donde pueden dejar mensajes o también puede escribirme a mi correo eshimoyama@esbabarrionorte.edu.ar.