Laboratorio 01

Tipos de datos, condicionales y bucles

1. Competencias

1.1. Competencias del curso

Conoce, comprende e implementa programas usando las estructuras básicas del lenguaje de programación C++.

1.2. Competencia del laboratorio

Conoce, comprende e implementa programas usando las estructuras básicas del lenguaje de programación C++.

2. Equipos y Materiales

- Un computador.
- IDE para C++.
- Compilador para C++.

3. Marco Teórico

3.1. Tipos De datos

C++ es un lenguaje de programación que hereda muchos conceptos del lenguaje C, es un lenguaje compilado y fuertemente tipado, lo que significa que en las variables con las que trabajamos hay que indicar el tipo del dato que van a guardar cuando se declaran.

Los tipos de datos básicos normalmente son los numéricos y en la siguiente tabla podemos ver estos tipos de datos simples en C++, su nombre, descripción, el tamaño definido por el tipo de dato, que proporciona la cantidad de información que podemos guardar en una variable de ese tipo de dato, y el rango de valores que permite almacenar.

Tipos de datos numéricos enteros

El tipo de dato numérico entero es un subconjunto finito de los números enteros del mundo real. Pueden ser positivos o negativos.

Tipo de Dato	Descripción	Número de bytes típico	Rango
short	Entero corto	2	-32768 a 32767
int	Entero	4	-2147483648 a +2147483647
long	Entero largo	4	-2147483648 a +2147483647
char	Carácter	1	-128 a 127

Con los tipos enteros pueden utilizarse los calificadores signed y unsigned. Estos calificadores indican si el número tiene signo o no. Si se usan solos, sin indicar el tipo de dato se asume int.

Tipo de Dato	Descripción	Número de bytes típico	Rango
signed short	Entero corto	2	-32768 a 32767
unsigned short	Entero corto sin signo	2	0 a 65535
signed int	Entero	4	-2147483648 a +2147483647
unsigned int	Entero sin signo	4	0 a 4294967295
signed long	Entero largo	4	-2147483648 a +2147483647
unsigned long	Entero largo sin signo	4	0 a 4294967295
signed char	Carácter	1	-128 a 127
unsigned char	Carácter sin signo	1	0 a 255

Tipos de datos numéricos reales

El tipo de dato numérico real es un subconjunto finito de los números reales. Pueden ser positivos o negativos.

Tipo de Dato	Descripción	Número de bytes típico	Rango
float	Real (Número en coma flotante)	4	Positivos: 3.4E-38 a 3.4E38 Negativos: -3.4E-38 a -3.4E38
double	Real doble (Número en coma flotante de doble precisión)	8	Positivos: 1.7E-308 a 1.7E308 Negativos: -1.7E-308 a -1.7E308
long double	Real doble largo	10	Positivos: 3.4E-4932 a 1.1E4932 Negativos: -3.4E-4932 a -1.1E4932

Tipo de dato lógico

Los datos de este tipo sólo pueden contener dos valores: true ó false (verdadero ó falso).

Tipo de Dato	Descripción	Número de bytes típico	Rango
bool	Dato de tipo lógico	1	0, 1

3.2. Condicionales

En ocasiones, es necesario realizar una u otra acción dependiendo de si se cumple o no una condición predeterminada. Es como si tuviéramos dos caminos diferentes por los cuales el programa puede continuar dependiendo del resultado de alguna condición previa.

Para llevar a cabo estos procesos, C++ cuenta con sentencias condicionales que se encargan de evaluar si una condición es falsa o verdadera. Este tipo de procesos condicionales, se puede llevar a cabo con las instrucciones if, if-else y switch.

Condicional if

Los condicionales if, son una estructura de control condicional, también llamadas estructuras selectivas de casos simples (porque solo definen un posible flujo), las cuales nos permiten tomar cierta decisión al interior de nuestro algoritmo, es decir, nos permiten determinar qué acciones ejecutar según cierta condición sea verdadera.

La sintaxis de un condicional if, es bastante simple e incluso creo que intuitiva.

```
if(condición a evaluar)
{
   Bloque de Instrucciones si se cumple la condición....
}
....
Bloque de Instrucciones restante DEL ALGORITMO....
....
```

Condicional if-else

Los condicionales if-else, son una estructura de control, que nos permiten tomar cierta decisión al interior de nuestro algoritmo, es decir, nos permiten determinar qué acciones tomar dada o no cierta condición.

En otras palabras, es una estructura que nos posibilita definir las acciones que se deben llevar a cabo si se cumple cierta condición y también determinar las acciones que se deben ejecutar en caso de que no se cumpla; generando así una separación o bifurcación en la ejecución del programa, ejecutando ciertas acciones u otras a partir de la evaluación de una condición dada.

La sintaxis de un condicional if-else, es en principio similar a la del condicional if, pero adicionando una nueva "estructura" que es el else, el cual indica la acción o conjunto de acciones a llevar a cabo, en caso de que la condición del if no se cumpla. Cabe resaltar que el else siempre se pone inmediatamente después del if, en caso de ser necesario, el else es incapaz de funcionar por sí solo, siempre debe ir acompañado por un if.

```
if(condición a evaluar)
{
   Bloque de Instrucciones si se cumple la condición....
}
else
{
   Bloque de Instrucciones si NO se cumple la condición....
}
```

Condicional Switch

Los condicionales Switch, son una estructura de control condicional, que permite definir múltiples casos que puede llegar a cumplir una variable cualquiera, y qué acción tomar en cualquiera de estas situaciones, incluso es posible determinar qué acción llevar a cabo en caso de no cumplir ninguna de las condiciones dadas.

La sintaxis de un condicional Switch es bastante distinta a la de un condicional típico, sin embargo, es bastante intuitiva y fácil de comprender, es solo cuestión de acostumbrarse.

```
switch(opción) //donde opción es la variable a comparar
{
   case valor1: //Bloque de instrucciones 1;
   break;
   case valor2: //Bloque de instrucciones 2;
   break;
   case valor3: //Bloque de instrucciones 3;
   break;
   //Nótese que valor 1 2 y 3 son los valores que puede tomar la opción
   //la instrucción break es necesaria, para no ejecutar todos los casos.
   default: //Bloque de instrucciones por defecto;
   //default, es el bloque que se ejecuta en caso de que no se dé ningún caso
}
```

3.3. Bucles

Un ciclo o bucle permite repetir una o varias instrucciones cuantas veces lo necesitemos, por ejemplo, si quisiéramos escribir los números del uno al cien no tendría sentido escribir cien líneas mostrando un numero en cada una, para esto y para muchísimas cosas más, es útil un ciclo, permitiéndonos hacer una misma tarea en una cantidad de líneas muy pequeña y de forma prácticamente automática.

Existen diferentes tipos de ciclos o bucles, cada uno tiene una utilidad para casos específicos y depende de nuestra habilidad y conocimientos poder determinar en qué momento es bueno usar alguno de ellos. Tenemos entonces a nuestra disposición los siguientes tipos de ciclos en C++:

- Bucle for
- Bucle while
- Bucle do-while
- Bucle anidado

Bucle o ciclo for

Los ciclos for son lo que se conoce como estructuras de control de flujo cíclicas o simplemente estructuras cíclicas, estos ciclos, como su nombre lo sugiere, nos permiten ejecutar una o varias líneas de código de forma iterativa, conociendo un valor especifico

inicial y otro valor final, además nos permiten determinar el tamaño del paso entre cada "giro" o iteración del ciclo.

En resumen, un ciclo for es una estructura de control iterativa, que nos permite ejecutar de manera repetitiva un bloque de instrucciones, conociendo previamente un valor de inicio, un tamaño de paso y un valor final para el ciclo.

La sintaxis de un ciclo for es simple en C++, en realidad en la mayoría de los lenguajes de alto nivel es incluso muy similar, de hecho, con tan solo tener bien claros los 3 componentes del ciclo for (inicio, final y tamaño de paso) tenemos prácticamente todo hecho

```
for(int i = valor inicial; i <= valor final; i = i + paso)
{
   Bloque de Instrucciones....
}
```

Bucle o ciclo while

Los ciclos while son también una estructura cíclica, que nos permite ejecutar una o varias líneas de código de manera repetitiva sin necesidad de tener un valor inicial e incluso a veces sin siquiera conocer cuando se va a dar el valor final que esperamos, los ciclos while, no dependen directamente de valores numéricos, sino de valores booleanos, es decir su ejecución depende del valor de verdad de una condición dada, verdadera o falso, nada más. De este modo los ciclos while, son mucho más efectivos para condiciones indeterminadas, que no conocemos cuando se van a dar a diferencia de los ciclos for, con los cuales se debe tener claro un principio, un final y un tamaño de paso.

La sintaxis de un ciclo while es incluso más simple y "legible" que la del ciclo for en C++, pues simplemente requerimos tener clara una condición de parada.

```
while(condición de finalización)
{
   Bloque de Instrucciones....
}
```

Bucle o ciclo do-while

Los ciclos do-while son una estructura de control cíclica, los cuales nos permiten ejecutar una o varias líneas de código de forma repetitiva sin necesidad de tener un valor inicial e incluso a veces sin siquiera conocer cuando se va a dar el valor final, hasta aquí son similares a los ciclos while, sin embargo el ciclo do-while nos permite añadir cierta

ventaja adicional y esta consiste que nos da la posibilidad de ejecutar primero el bloque de instrucciones antes de evaluar la condición necesaria, de este modo los ciclos do-while, son más efectivos para algunas situaciones específicas.

La sintaxis de un ciclo do-while es un tanto más larga que la del ciclo while en C++, sin embargo, no se hace más complicado, de hecho, con tan solo tener bien clara una condición de finalización para el ciclo tendremos prácticamente todo terminado.

```
do
{
   Bloque de Instrucciones....
}
while(condición de finalización);
```

Bucle o ciclo anidado

Los ciclos anidados no son una estructura de control por sí mismos, son un indicativo de que quizá deberíamos plantear la solución a algún problema si nos vemos obligados a usar ciclos anidados y más aún si es más de uno, sin embargo, debemos saber que a veces son indispensables.

La sintaxis es sencilla, un ciclo con otro adentro, y lo que nos haga falta, pues podemos poner varias sentencias adicionales al interior de cualquiera de los dos ciclos.

```
for(int i = valor inicial; i < valor final; i++)
{
      Bloque de Instrucciones....
   for(int j = valor inicial; j < valor final; j++)
   {
      Bloque interno de Instrucciones....
   }
   //Acá puede haber más instrucciones
}
```

4. Ejercicios

Resolver los siguientes ejercicios planteados:

1. Sumar todos los enteros pares desde 2 hasta 100.

2. Escriba un código que solicite al usuario ingresar dos números enteros y que muestre el producto de ambos.
3. Escriba un código que solicite el primer nombre de una persona, el apellido paterno y el apellido materno. Retornar su correo UNSA generado, el cual consiste de la primera letra del nombre, el apellido paterno completo, y la primera letra del apellido materno. (se agrega el dominio de la universidad al final).
4. Calcule los primeros 50 números primos y muestre el resultado en pantalla.
5. Escribir un programa que visualice en pantalla los números múltiplos de 5 comprendidos entre 1 y 100.
6. Escriba un código que solicite ingresar dos números x y y, tal que $x < y$. Muestre todos los números primos que se encuentren entre el rango de valores, de no encontrarse, mostrar el primo más cercano a x o y.
7. Elabore un programa que solicite ingresar una hora del día (HH:MM en formato de cadena), solicite un tiempo en minutos a agregar, y retorne la hora de finalización (el formato de salida debe de estar en AM o PM según corresponda).
8. Escriba un código que solicite una cantidad de minutos específica y muestre como resultado la hora y fecha resultante tomando como referencia la hora y fecha actual y restarle el tiempo indicado.
9. Elabore un código que reciba como entrada una secuencia de caracteres que contiene un numero flotante y retorne el número redondeado.
10. Elabore un código que solicite un numero entre 100 < x < 999 y muestre el resultado en binario.
11. Elabore un programa que lea n números y determine cuál es el mayor, el menor y la media de los números leídos.
12. Elabore un programa que calcule la serie de Fibonacci. La serie de Fibonacci es la sucesión de números: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... Cada número se calcula sumando los dos anteriores a él.
13. Elabore un algoritmo que lea por teclado dos números enteros y determine si uno es divisor del otro.
14. Calcula el promedio de 3 notas para n estudiantes.
15. Escribir un programa que genere la tabla de multiplicar de un número introducido por el teclado.
16. Escribir un programa que calcule la media de x cantidad números introducidos por el teclado.

17. Escribir un programa que lea 10 datos desde el teclado y sume sólo aquellos que sean negativos.
18. Escribir un programa que pida al usuario un número entero y muestre por pantalla un triángulo rectángulo como el de más abajo, de altura el número introducido.

    *
    **
    ***
    ****
    *****
19. Escribir un programa que almacene la cadena de caracteres contraseña en una variable, pregunte al usuario por la contraseña hasta que introduzca la contraseña correcta.
20. Escribir un programa que pida al usuario una palabra y luego muestre por pantalla una a una las letras de la palabra introducida empezando por la última.

5. Entregables

Al final estudiante deberá:

1. Compactar el código elaborado y subirlo al aula virtual de trabajo. Agregue sus datos personales como comentario en cada archivo de código elaborado.
2. Elaborar un documento que incluya tanto el código como capturas de pantalla de la ejecución del programa. Este documento debe de estar en formato PDF.
3. El nombre del archivo (comprimido como el documento PDF), será su LAB01_GRUPO_A/B/C_CUI_1erNOMBRE_1erAPELLIDO.

   (Ejemplo: LAB01_GRUPO_A _2022123_PEDRO_VASQUEZ).
4. Debe remitir el documento ejecutable con el siguiente formato:

   LAB01_GRUPO_A/B/C_CUI_ EJECUTABLE_1erNOMBRE_1erAPELLIDO

   (Ejemplo: LAB01_GRUPO_A_EJECUTABLE_2022123_PEDRO_VASQUEZ).

En caso de encontrarse trabajos similares, los alumnos involucrados no tendrán evaluación y serán sujetos a sanción.