

Programación Orientada a Objetos, con Principios de Programación

Parcial 1
Notas de curso

Dr. Ezequiel Arceo May

21 de marzo de 2019

Índice

Índice	1
1. Programa: la representación de un problema	3
1.1. Identificadores (nombres para las variables y las funciones)	4
2. Datos I: información que yo escribo en mi programa	6
2.1. Variables (para guardar valores)	6
2.2. Valores y Tipos de datos	8
3. Funciones I: las partes de una función	11
4. Librerías	15
4.1. iostream: entrada y salida de datos	15
4.2. string: manejo de cadenas de caracteres	16
4.3. vector: manejo de colecciones de datos	16
5. Algoritmos 0: Pseudo-código y diagramas de flujo	22
5.1. Algoritmo	22
5.2. Pseudo-código	22
5.3. Diagramas de flujo	26
6. Algoritmos I: Toma de decisiones	32
6.1. if	32
6.2. if . . . else	33
6.3. Operadores lógicos	34
7. Algoritmos II: Repeticiones	40
7.1. for	40
7.2. while	42
7.3. do/while	42

8. Funciones II: Definición y Llamado	44
8.1. Funciones sin argumentos y sin valor de retorno	44
8.2. Funciones sin argumentos y con valor de retorno	47
8.3. Funciones con argumentos y sin valor de retorno	48
8.4. Funciones con argumentos y con valor de retorno	50
9. Datos II: donde es “visible/accesible” una variable/función	58
9.1. Uso de ámbitos: declaración de funciones	61

«Un language de programación es una forma de expresarnos»
— Ezequiel Arceo —

1. Programa: la representación de un problema

Usamos un language de programación para resolver un problema. En dicho language, nuestro problema lo expresamos/escribimos con **datos** y con **funciones**.

- los **datos**: representan unidades de información, entonces los datos son valores.
- las **funciones**: representan acciones, entonces las funciones son procedimientos o procesos.

Y lo que hacemos es

1. **escribir**: escribimos instrucciones (**sentencias**) en un archivo de texto con extensión apropiada, por ejemplo `programa.cpp`. A este archivo de texto con instrucciones se le llama **código fuente**.
2. **compilar**: un programa especial llamado compilador traduce las instrucciones en nuestro programa a un language que la computadora si puede entender. A este language se le conoce como **language máquina**.
3. **ejecutar**: ejecutamos el resultado de la compilación, el cual es un archivo de instrucciones que la máquina puede leer y ejecutar. A este archivo se le conoce como **ejecutable**.

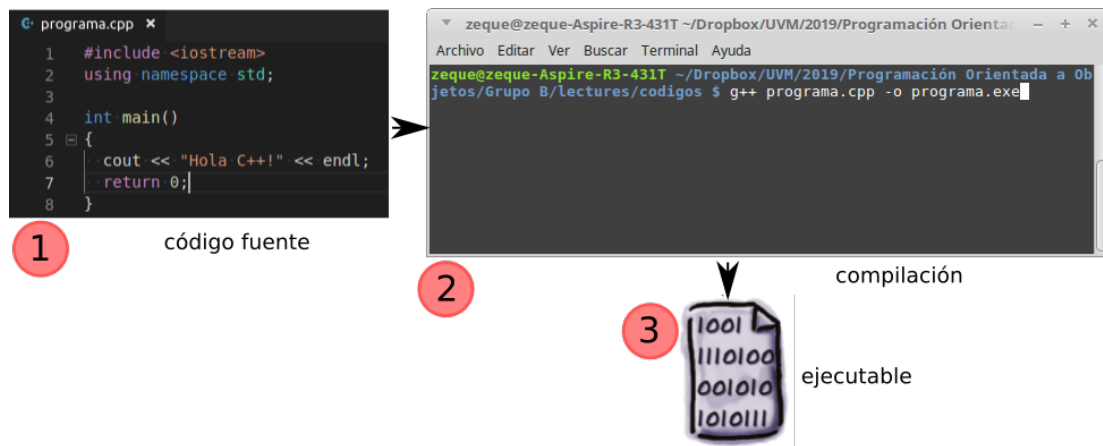


Figura 1: La vida de un programa.

1.1. Identificadores (nombres para las variables y las funciones)

Para facilitarnos el trabajo, es común que los datos y las funciones los asociemos a un nombre (**identificador**) con el cuál referirnos a ellos. Note que su nombre, el nombre de su escuela, y el nombre de su mascota son identificadores en el mundo real.

En C++ hay reglas para los identificadores que podemos usar:

- Son **identificadores válidos** las combinaciones de a) letras (sin acentos, ni la ñ), b) números y c) _ (guión bajo), si el primer caracter no es un número y no contiene espacios en blanco.

Ejemplos de **identificadores válidos**: hola, Hola, hola_mundo, numero1, mi_numero,...

- Son **identificadores inválidos** las combinaciones de a) letras (con acentos, y la ñ), b) números y c) _ (guión bajo), si el primer caracter es un número, o contiene espacios en blanco. O el identificador es una *palabra reservada* por el lenguaje (if, else, for, while, return, int, float, bool, char, etc)

Ejemplos de **identificadores inválidos**: 5hola, cabaña, hola mundo, número1, mi_número, ...

Actividad 1.

Clasifica los siguientes identificadores como válidos o inválidos marcando con una X en la casilla correspondiente

Identificador	válido	inválido
c++		
martin		
martín		
mi número		
777suerte		
el_nombre		
float		
mi-variable		
programacionOrientadaObjetos		
el+bonito		

Trabajo A1. Entrega Antes de: Jueves 7 de Marzo de 2019 a las 23:59.

Clasifica los siguientes identificadores como válidos o inválidos marcando con una X en la casilla correspondiente

Identificador	válido	inválido
nombre		
jose		
jose1		
minúmero		
otro numero		
cadena_de_texto		
hola8mundo		
cosa_		
papitasFritas		
int		

- Enviar por correo a `ezequiel_arceo@my.uvm.edu.mx`
- El asunto del correo será `P00_TrabajoA1_NOMBRE_DEL_AUTOR`
- El nombre de cada archivo adjunto será `P00_TrabajoA1_NOMBRE_DEL_AUTOR.*`

Ejemplo:

ASUNTO: `P00_TrabajoA1_JUAN_PEREZ_FERNANDEZ`

ADJUNTO: `P00_TrabajoA1_JUAN_PEREZ_FERNANDEZ.cpp`

2. Datos I: información que yo escribo en mi programa

2.1. Variables (para guardar valores)

Una **variable** es un “espacio” en donde guardamos un dato de algún tipo, y le damos un nombre (identificador) para “recordarla”.

- *Definimos* una variable cuando damos un valor a un identificador.

En C++ esto lo hacemos escribiendo una sentencia de la forma

tipo_de_dato **identificador_valido** = **valor_inicial**;

```
1 // EJEMPLOS: Definición de variable
2 int miEntero = 12;
3 float miFlotante = 3.1416;
4 string miCadena = "pepito";
```

- *Declaramos* una variable cuando decimos que un identificador **puede** tomar valores de un tipo.

En C++ esto lo hacemos escribiendo una sentencia de la forma

tipo_de_dato **identificador_valido** ;

```
1 // EJEMPLOS: Declaración de variable
2 int miOtroEntero;
3 float miOtroFlotante;
4 string miOtraCadena;
```

Tenemos que saber que una variable tiene (entre otras cosas)

1. un **tipo**: que indica que tipo de valores puede tomar,
2. un **nombre**: que es un identificador para referirse a ella, y
3. un **valor**: que es la información que guarda.

Recuerde:

Variable = tipo de dato + nombre + valor

Actividad 2.

Clasifica las siguientes sentencias como “definición de variable” o “declaración de variable”

```
1 float interes;  
2 float pesosPorDolar = 23;  
3 int habitacionesReservadas = 2;  
4 string correo = "Buen día: ...";  
5 bool necesitaReparacion;  
6 int botellasVacias = 12;  
7 float temperatura = 36.7;  
8 string moto = "una frase pegajosa";  
9 string nombreProfesor;  
10 int diasRestantes = 3;  
11 float gramosDeAzucar = 5.2;  
12 string materia = "Programación Orientada a Objetos";
```

Trabajo A2. Entrega Antes de: Jueves 7 de Marzo de 2019 a las 23:59.

Clasifica las siguientes sentencias como “definición de variable” o “declaración de variable”

```
1 int ejemplo0 = 0; // Definición de variable  
2 string mensaje = "soy un valor de cadena";  
3 float tiempo;  
4 int numeroDeManos;  
5 int costo = 23;  
6 string objetivo = "el alumno aprende C++";  
7 float temperatura = 36.7;  
8 string recuerda = "tu puedes";  
9 int memoriaGB = 1024;  
10 float trabajoRestante = 0.0;
```

- Enviar por correo a ezequiel_arceo@my.uvm.edu.mx
- El asunto del correo será P00_TrabajoA2_NOMBRE_DEL_AUTOR
- El nombre de cada archivo adjunto será P00_TrabajoA2_NOMBRE_DEL_AUTOR.*

Ejemplo:

ASUNTO: P00_TrabajoA2_JUAN_PEREZ_FERNANDEZ

ADJUNTO: P00_TrabajoA2_JUAN_PEREZ_FERNANDEZ.cpp

2.2. Valores y Tipos de datos

Nos centraremos en *valores numéricos*, *cadenas de caracteres* y *valores booleanos*

- a los valores numéricos que solo tienen dígitos, como 1, 3, -5, 14 ..., se les llama **de tipo entero**.

En C++ “de tipo entero” se dice **int**. Podemos usar valores **int** de dos formas

```
1 // EJEMPLOS: Uso de int
2 // opción 1: uso literal de valor int
3 12;
4 16;
5
6 // opción 2: definición de variable int
7 int miEdad = 34;
8 int numeroDeHermanos = 7;
```

- a los valores numéricos que tienen dígitos y punto, como 1.7, 3.1416, -55.7, 14.45 ..., se les llama **de tipo flotante** (decimal, real).

En C++ “de tipo flotante” se dice **float**. Podemos usar valores **float** de dos formas

```
1 // EJEMPLOS: Uso de float
2 // opción 1: uso literal de valor float
3 12.5;
4 16.24;
5
6 // opción 2: definición de variable float
7 float miEstatura = 1.54;
8 float miPeso = 102.86;
```

- a los valores que aparecen como caracteres entre comillas dobles (”), como “Hola”, “adios”, “esto es una cadena”, “supercalifragilisticoespialidoso” ..., se les llama **de tipo cadena** (palabras, texto).

En C++ “de tipo cadena” se dice **string**. Podemos usar valores **string** de dos formas

```

1 // EJEMPLOS: Uso de string
2 // opción 1: uso literal de valor string
3 "Hola Mundo";
4 "Programación en C++";
5
6 // opción 2: definición de variable string
7 string miNombre = "Ezequiel Arceo May";
8 string miLugarDeNacimiento = "Buctzotz"; // le reto a leer :)

```

- a los valores **true** y **false** se les llama **de tipo booleano** (**true**=verdadero, **false**=falso).

En C++ “de tipo booleano” se dice **bool**. Podemos usar valores **bool** de dos formas

```

1 // opción 1: uso literal de valor bool
2 true;
3 false;
4
5 // opción 2: definición de variable bool
6 bool meEncantaElSushi = true;
7 bool meGustaCocinar = false; // pero me encanta comer

```

Actividad 3.

Identifica el tipo, el nombre y el valor inicial en las siguientes definiciones de variable.

```

1 string trabalengua = "pablito clavó un clavito en la calva de un calvito";
2 float diferenciaDeHorario = -4;
3 int cubitosDeAzucar = 2;
4 bool conAzucar = false;
5 string salon = "E 204";
6 float temperaturaKelvin = -273.5;
7 string producto = "Suscripción Netflix 12 meses";
8 float precio = 974.50;
9 bool aprobado = true;
10 float masa = 54.25;
11 int ruta = 42;

```

Ejemplo

```
1 string ejemplo = "yo puedo aprender C++";  
2 // tipo: string  
3 // nombre: ejemplo  
4 // valor: "yo puedo aprender C++"
```

Trabajo A3. Entrega Antes de: Jueves 7 de Marzo de 2019 a las 23:59.

Identifica el tipo, el nombre y el valor inicial en las siguientes definiciones de variable.

```
1 string mensaje = "soy un valor de cadena";  
2 float tiempo = 8.40;  
3 int numeroDeManos = 2;  
4 int costo = 23;  
5 float temperatura = 36.7;  
6 string descripcion = "la descripción de algo";  
7 int numeroExterior = 63;  
8 float altura = 253.35;  
9 float gravedad = 9.81;  
10 int cantidadDePuertas = 4;
```

- Enviar por correo a ezequiel_arceo@my.uvm.edu.mx
- El asunto del correo será P00_TrabajoA3_NOMBRE_DEL_AUTOR
- El nombre de cada archivo adjunto será P00_TrabajoA3_NOMBRE_DEL_AUTOR.*

Ejemplo:

ASUNTO: P00_TrabajoA3_JUAN_PEREZ_FERNANDEZ

ADJUNTO: P00_TrabajoA3_JUAN_PEREZ_FERNANDEZ.cpp

3. Funciones I: las partes de una función

Una función es una entidad que ejecuta una tarea bien definida. Se expresa (escribe) como un conjunto de instrucciones agrupadas que se denomina **cuerpo de la función**. A la función se le da un identificador llamado **nombre de la función**. Para hacer su trabajo, una función puede requerir cero o más recursos (información) del exterior, a estos recursos se les llama **argumentos de la función**. Cuando termina su trabajo, una función puede o no devolver un valor, a este valor se le denomina **valor de retorno (de la función)**. Toda función tiene un tipo, **una función es del mismo tipo que su valor de retorno**.

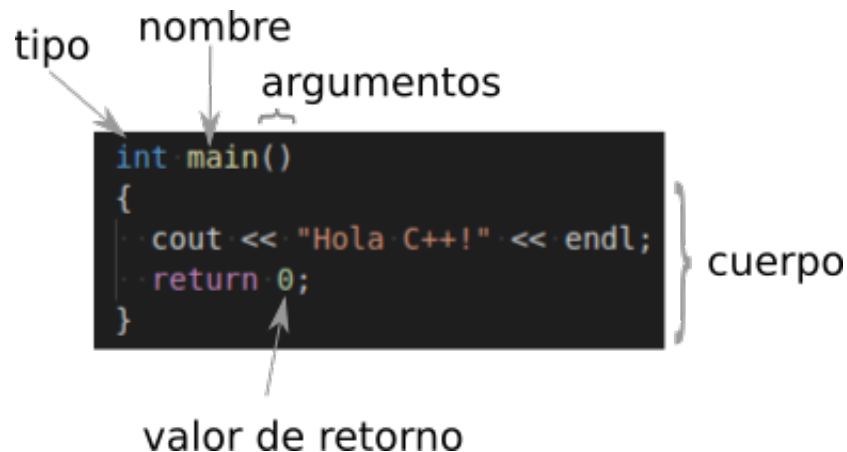


Figura 2: Las partes de una función.

Tenemos que saber que una función tiene

1. un **nombre**: que es un identificador para referirse a ella,
2. cero o más **argumentos** (entre paréntesis): que son las partes de información necesaria para hacer su trabajo,
3. un **cuerpo** (entre llaves): que son las instrucciones para hacer su trabajo,
4. un **valor de retorno**: que es el valor que devuelve al terminar su trabajo, y
5. un **tipo**: que es el tipo de dato de su valor de retorno.

En C++ una función que no tiene valor de retorno es de tipo **void**.

La primera función que vamos a describir es requerida en todo programa C++. Nos referimos a la función **main**. Esta es la primera función que se llama cuando nosotros ejecutamos nuestro programa.

main no necesita argumentos, y puede ser de dos tipos:

▪ **main** tipo **void**

```
1 // EJEMPLO: función main de tipo void
2 void main()
3 { // inicia el cuerpo de la función
4   // Aquí van las instrucciones del programa
5 }
```

- **nombre:** main,
- **argumentos** (entre paréntesis): ninguno,
- **cuerpo** (entre llaves): líneas 3 a 5,
- **valor de retorno:** ninguno, y
- **tipo:** void.

▪ **main** tipo **int**

```
1 // EJEMPLO: función main de tipo int
2 int main()
3 { // inicia el cuerpo de la función
4   // Aquí van las instrucciones del programa
5   return 0;
6 }
```

- **nombre:** main,
- **argumentos** (entre paréntesis): ninguno,
- **cuerpo** (entre llaves): líneas 3 a 6,
- **valor de retorno:** 0, y
- **tipo:** int.

En este ejemplo notará que una función devuelve su valor de retorno en la forma

```
1 // EJEMPLO: devolver un valor de retorno
2 return valorDeRetorno;
```

Actividad 4.

Identifique el nombre, argumentos, cuerpo, valor de retorno y tipo de las siguientes funciones

```
1 // EJEMPLO: funciones a describir
2
3 int sumaInt(int a, int b)
4 {
5     return a + b
6 }
7
8 int restaInt(int a, int b)
9 {
10    return a - b
11 }
12
13 void imprimeInt(int a)
14 {
15     cout << a << endl;
16 }
```

Ejemplo:

```
1 // EJEMPLO: descripción de función
2
3 float mitad(float decimal)
4 {
5     return decimal/2;
6 }
7 // nombre: mitad,
8 // argumentos: float decimal,
9 // cuerpo: líneas 4 a 6,
10 // valor de retorno: decimal/2,
11 // tipo: float
```

Trabajo B1. Entrega Antes de: Jueves 7 de Marzo de 2019 a las 23:59.

Identifique el nombre, argumentos, cuerpo, valor de retorno y tipo de las siguientes funciones

```
1 // EJEMPLO: funciones a describir
2
3 void echoInt(int a)
4 {
5     cout << a << endl;
6 }
7
8 int minimoInt(int a, int b)
9 {
10     if(a < b)
11     {
12         return a;
13     }
14     else
15     {
16         return b;
17     }
18 }
19
20 int pipaInt(int a)
21 {
22     cout << "Vi pasar al entero " << a << endl;
23     return a;
24 }
```

- Enviar por correo a ezequiel_arceo@my.uvm.edu.mx
- El asunto del correo será POO_TrabajoB1_NOMBRE_DEL_AUTOR
- El nombre de cada archivo adjunto será POO_TrabajoB1_NOMBRE_DEL_AUTOR.*

Ejemplo:

ASUNTO: POO_TrabajoB1_JUAN_PEREZ_FERNANDEZ

ADJUNTO: POO_TrabajoB1_JUAN_PEREZ_FERNANDEZ.cpp

4. Librerías

Cuando escribimos un programa, es casi 100 % seguro que necesitaremos usar funciones creadas previamente por alguien más. Estas funciones se agrupan y se hacen disponibles para el uso en lo que llamamos **librerías**. Para poder usar una librería es necesario *importarla*. Si queremos importar una librería llamada XYZ , necesitamos escribir una línea como la que sigue *en la parte superior de nuestro programa*

```
1 // EJEMPLO: importación de librería
2 // importa la librería llamada XYZ
3 #include <XYZ>
```

4.1. iostream: entrada y salida de datos

La primera librería que importaremos se llama **iostream**. Esta nos permite imprimir en pantalla (salida de datos) y leer información desde el teclado (entrada de datos). De la librería iostream usaremos los objetos `cout`, `cin` y `endl`.

```
1 // EJEMPLO: importación de librerías
2 // importa la librería iostream
3 #include <iostream>
```

Para imprimir en pantalla usamos **cout** en cualquiera de las siguientes formas válidas:

```
1 // EJEMPLO: formas válidas de imprimir en pantalla
2 #include <iostream>
3
4 using namespace std;
5
6 void main()
7 {
8     cout << "Hola" << endl;
9     cout << "Hola " << "Mundo # " << 1 << endl;
10 }
```

El objeto `endl` imprime un “salto de línea” en la pantalla, lo que obtenemos al presionar ENTER mientras escribimos de forma común.

Para leer del teclado usamos **cin** en la siguiente forma válida:

```
1 // EJEMPLOS: forma válida de leer desde el teclado
2 #include <iostream>
3
4 using namespace std;
5
6 void main()
7 {
8     // Declaramos una variable int llamada numero
9     int numero;
10    // Damos instrucciones al usuario
11    cout << "Inserte un entero: " << endl;
12    // Leemos un entero y lo guardamos en la variable numero
13    cin >> numero;
14    // Le damos retroalimentación al usuario
15    cout << "Su número es: " << numero << endl;
16 }
```

4.2. string: manejo de cadenas de caracteres

La librería **string** nos permite manejar cadenas de caracteres en forma natural con el tipo de dato **string**. Es necesario importar la librería **string** antes de poder usar variables de tipo **string**.

```
1 // EJEMPLO: uso de variables string
2 #include <iostream>
3 #include <string>
4
5 using namespace std;
6
7 void main()
8 {
9     string miCadena = "soy el valor de una variable string";
10    cout << miCadena << endl;
11 }
```

4.3. vector: manejo de colecciones de datos

No pasará mucho antes de que en un programa necesitemos manejar muchos valores. Podemos agrupar los valores de un mismo tipo dentro de un contenedor, y manejar al contenedor como una variable. Podemos requerir por ejemplo, una colección de calificaciones (float's), una colección de nombres (string's), una colección de asistencias (int's), etc.

La librería **vector** nos permite definir colecciones de **valores de un mismo tipo**, a los cuales le llama vectores, y son de tipo **vector**.

Para declarar una variable de tipo **vector** llamado `miVecInt`, para valores de tipo **int** se usa la siguiente sintaxis:

```
1 // EJEMPLO: declaración de un vector de enteros
2 // (sin valores iniciales)
3 #include <iostream>
4 #include <vector>
5
6 using namespace std;
7
8 void main()
9 {
10     // Crea un vector de enteros, VACIO
11     vector<int> miVecInt;
12     cout << "miVecInt tiene " << miVecInt.size() << " elementos" << endl;
13     // Añade el valor 4 a miVecInt
14     miVecInt.push_back(4);
15     cout << "miVecInt tiene " << miVecInt.size() << " elementos" << endl;
16     // Añade el valor 13 a miVecInt
17     miVecInt.push_back(13);
18     cout << "miVecInt tiene " << miVecInt.size() << " elementos" << endl;
19 }
```

```
1 // EJEMPLO: definición de un vector de enteros
2 // (con valores iniciales)
3 #include <iostream>
4 #include <vector>
5
6 using namespace std;
7
8 void main()
9 {
10     // Crea un vector de enteros, CON VALORES INICIALES
11     vector<int> miVecInt = {1,2,3};
12     cout << "miVecInt tiene " << miVecInt.size() << " elementos" << endl;
13     // Añade el valor 4 a miVecInt
14     miVecInt.push_back(4);
15     cout << "miVecInt tiene " << miVecInt.size() << " elementos" << endl;
16     // Añade el valor 13 a miVecInt
17     miVecInt.push_back(13);
18     cout << "miVecInt tiene " << miVecInt.size() << " elementos" << endl;
19 }
```

Podemos crear vectores para contener todos los tipos de valores que hemos visto hasta el momento:

```
1 // EJEMPLO: declaración de vector de diversos tipos
2 // (sin valores iniciales)
3 #include <iostream>
4 #include <string>
5 #include <vector>
6
7 using namespace std;
8
9 void main()
10 {
11     // Crea un vector de enteros, VACIO
12     vector<int> miVecInt;
13     // Crea un vector de flotantes, VACIO
14     vector<float> miVecFloat;
15     // Crea un vector de booleanos, VACIO
16     vector<bool> miVecBool;
17     // Crea un vector de cadenas, VACIO
18     vector<string> miVecString;
19 }
```

```
1 // EJEMPLO: definición de vector de diversos tipos
2 // (con valores iniciales)
3 #include <iostream>
4 #include <string>
5 #include <vector>
6
7 using namespace std;
8
9 void main()
10 {
11     // Crea un vector de enteros, CON VALORES INICIALES
12     vector<int> miVecInt = {31, 10, 85};
13     // Crea un vector de flotantes, CON VALORES INICIALES
14     vector<float> miVecFloat = {3.1416, 2.7182, 0.5772};
15     // Crea un vector de booleanos, CON VALORES INICIALES
16     vector<bool> miVecBool = {true, false, true, false};
17     // Crea un vector de cadenas, CON VALORES INICIALES
18     vector<string> miVecString = {"hola", "C++", " :)"};
19 }
```

Un vector de tamaño n se dice que tiene n elementos. El primer elemento está en la posición 0 y el último está en la posición $n - 1$.

Al tratar con variables de tipo vector (además de su creación) nos sirve saber

- **como determinar su tamaño.** Si `vec` es una variable de tipo vector, su tamaño (número de valores que contiene) lo obtenemos usando su “método `size`”, `vec.size()`.

Beneficio: usando el “método `size`” de un vector ya no necesito *recordar* su tamaño.

- **como añadir un valor (a la derecha).** Si `vec` es una variable de tipo vector, podemos añadirle un valor `val` usando su “método `push_back`”, `vec.push_back(val)`.

Beneficio: usando el “método `push_back`” de un vector ya puedo hacer que el vector *crezca*.

- **como acceder a un elemento.** Si `vec` es una variable de tipo vector, podemos acceder al elemento en la posición *i* escribiendo `vec[i]`. Las posiciones de un vector se enumeran a partir de cero, y en un vector la posición más alta es el tamaño del vector menos uno.

```
1 // EJEMPLO: acciones con vectores
2 // a) determinar el tamaño de un vector
3 // b) añadirle elementos a un vector
4 #include <iostream>
5 #include <vector>
6
7 using namespace std;
8
9 void main()
10 {
11     // Crea un vector de enteros, VACIO
12     vector<int> miVecInt;
13     cout << "miVecInt tiene " << miVecInt.size() << " elementos" << endl;
14     // Añade el valor 4 a miVecInt
15     miVecInt.push_back(4);
16     cout << "miVecInt tiene " << miVecInt.size() << " elementos" << endl;
17     // Añade el valor 13 a miVecInt
18     miVecInt.push_back(13);
19     // Añade el valor 51 a miVecInt
20     miVecInt.push_back(51);
21     cout << "miVecInt tiene " << miVecInt.size() << " elementos" << endl;
22 }
```

Actividad 5.

Escriba una función que cumpla con las siguientes características

```

1 // EJEMPLO: acciones con vectores
2 // c) acceder a los elementos de un vector
3 #include <iostream>
4 #include <vector>
5
6 using namespace std;
7
8 void main()
9 {
10     // Crea un vector de enteros, CON VALORES INICIALES
11     vector<int> miVecInt = {31, 10, 1985};
12     cout << "El elemento en la posicion 0 es " << miVecInt[0] << endl;
13     cout << "El elemento en la posicion 1 es " << miVecInt[1] << endl;
14     cout << "El elemento en la posicion 2 es " << miVecInt[2] << endl;
15     // miVecInt tiene 3 elementos, en tres posiciones: 0, 1 y 2.
16 }

```

- en el cuerpo define una variable llamada `edadesFamilia` tipo **vector** de **int**'s,
- `edadesFamilia` tiene como valores iniciales las edades de dos de sus familiares,
- en el cuerpo añade a `edadesFamilia` las edades de otros tres familiares,
- imprime todos los elementos en `edadesFamilia` usando la notación de corchetes.

Trabajo B2. Entrega Antes de: Jueves 7 de Marzo de 2019 a las 23:59.

Escriba una función que cumpla con las siguientes características

- en el cuerpo define una variable llamada `peliculasFavoritas` tipo **vector** de **string**'s,
 - `peliculasFavoritas` tiene como valores iniciales los nombres de sus dos películas favoritas,
 - en el cuerpo añade a `peliculasFavoritas` los nombres de su tercera y cuarta película favorita,
 - imprime todos los elementos en `peliculasFavoritas` usando la notación de corchetes.
- Enviar por correo a ezequiel_arceo@my.uvm.edu.mx
 - El asunto del correo será `P00_TrabajoB2_NOMBRE_DEL_AUTOR`

-
- El nombre de cada archivo adjunto será P00_TrabajoB2_NOMBRE_DEL_AUTOR.*

Ejemplo:

ASUNTO: P00_TrabajoB2_JUAN_PEREZ_FERNANDEZ

ADJUNTO: P00_TrabajoB2_JUAN_PEREZ_FERNANDEZ.cpp

5. Algoritmos 0: Pseudo-código y diagramas de flujo

Antes de escribir un programa para resolver un problema debemos tener una comprensión exhaustiva del problema, así como una solución cuidadosamente planeada.

5.1. Algoritmo

La solución a cualquier problema computacional involucra la ejecución de una serie de acciones en un orden específico. Un **procedimiento** para resolver un problema en términos de

- las **acciones** a ser ejecutadas, y
- el **orden** en el que estas acciones serán ejecutadas,

se llama **algoritmo**. Especificar el orden de ejecución es importante, y a esto se le denomina **control del programa**.

5.2. Pseudo-código

Normalmente los algoritmos se escriben en un lenguaje artificial e informal llamado **pseudo-código**. El pseudo-código nos ayuda a pensar en un programa antes de empezar a programar: nos esconde los detalles y nos enfoca en los requerimientos del problema. En pocas palabras nos brinda una visión panorámica del problema. Cuando está *bien escrito*, el pseudo-código es fácil de traducir en un lenguaje de programación.

El pseudo-código consiste solamente en *acciones* y *decisiones* (por ejemplo, las declaraciones de variables no se incluyen)

Adoptaremos el siguiente esquema de pseudo-código:

Estructura de un Algoritmo en pseudo-código

- **Algoritmo:** nombre del algoritmo
- **Descripción:** descripción del algoritmo
- **Constantes:** valores que NO van a cambiar en el algoritmo.
- **Variables:** identificadores con valores que SI van a cambiar en el algoritmo.
- **INICIO**
 1. Operación # 1
 2. Operación # 2
 3. ...
- **FIN**

Donde cada Operación puede ser una acción o una decisión.
Veamos algunos ejemplos para familiarizarnos:

Ejemplo # 1: suma dos enteros

- **Algoritmo:** suma dos enteros
- **Descripción:** suma dos números enteros cualquiera
- **Constantes:**
- **Variables:** num1, num2, suma
- **INICIO**
 1. lee num1 y num2
 2. asigna a suma la suma de num1 y num2
 3. imprime el valor de suma
- **FIN**

Ejemplo # 2: promedia tres enteros

- **Algoritmo:** promedia tres enteros
- **Descripción:** determina el promedio de tres números enteros cualquiera
- **Constantes:**
- **Variables:** num1, num2, num3, suma, promedio
- **INICIO**
 1. lee num1, num2 y num3
 2. asigna a suma la suma de num1, num2 y num3
 3. asigna a promedio el valor de suma dividido entre tres.
 4. imprime el valor de promedio
- **FIN**

Ejemplo # 3: evalúa polinomio

- **Algoritmo:** evalúa polinomio
- **Descripción:** evalúa un polinomio en la variable x con coeficientes constantes $y = 2x^2 + 5x + 7$
- **Constantes:** 2,5,7
- **Variables:** x, y
- **INICIO**
 1. lee x
 2. asigna a y el valor de $2*x*x + 5*x + 7$
 3. imprime el valor de y
- **FIN**

Actividad 6.

Calcular el promedio de 3 números positivos.

En caso de que se proporcione algún número negativo:

- enviar un mensaje
- volver a pedir TODOS los números.

Actividad 7.

Calcular el promedio de 3 números positivos.

En caso de que se proporcione algún número negativo:

-
- enviar un mensaje
 - volver a pedir SOLO los números negativos.

Trabajo A4. Entrega Antes de: Jueves 21 de Marzo de 2019 a las 23:59 horas.

Elaborar un algoritmo para calcular la siguiente operación:

$$y = \frac{3x}{7x - 14}$$

Debe cumplir las siguientes condiciones:

- El valor de z es proporcionado por el usuario.
- La división debe ser válida (no dividir entre 0).
- En caso de que la división no sea válida (es un error):
 - el algoritmo debe solicitar un nuevo valor para x .
 - permitir un máximo 3 intentos (3 errores).
 - Una vez agotados los 3 intentos (3 errores), enviar el mensaje: “Número de intentos agotados”, salir del algoritmo.
- Enviar por correo a `ezequiel_arceo@my.uvm.edu.mx`
- El asunto del correo será `P00_TrabajoA4_NOMBRE_DEL_AUTOR`
- El nombre de cada archivo adjunto será `P00_TrabajoA4_NOMBRE_DEL_AUTOR.*`














Ejemplo:

ASUNTO: `P00_TrabajoA4_JUAN_PEREZ_FERNANDEZ`

ADJUNTO: `P00_TrabajoA4_JUAN_PEREZ_FERNANDEZ.cpp`

5.3. Diagramas de flujo

Un **diagrama de flujo** es una *representación gráfica* de los pasos que seguimos para realizar un proceso; partiendo de una entrada, y después de realizar una serie de acciones, llegamos a una salida. Cada paso se apoya en el anterior y sirve de sustento al siguiente.

Símbolo	Significado	Descripción
	Inicio o final	Indica el principio o el final del proceso.
	Entrada Datos	Entrada o salida de datos.
	Entrada manual	Introducción de datos.
	Subproceso predefinido	Conexión con procesos o rutinas ya definidas.
	Acción o proceso	Proceso, acción, tarea,...
	Decisión	Análisis de datos o proceso para tomar un camino u otro.
	Conector	Enlaza dos partes de un diagrama en la misma página.
	Conector fuera de página	Enlaza dos partes de un diagrama en diferentes páginas.
	Documento	Salida de documento.
	Multidocumento	Salida varios documentos.
	Línea de flujo	Dirección del proceso.
	Interacción	Une dos formas que tienen que realizarse a la vez.
	Repetición	Indica que se repita un número de veces las instrucciones.

¿Por qué usar diagramas de flujo?

- Presenta información en forma clara, ordenada y concisa
- Permite visualizar las frecuencias y relaciones entre las etapas indicadas.
- Permite detectar problemas, des-conexiones, pasos de escaso valor añadido etc.
- Compara y contrasta el flujo actual del proceso contra el flujo ideal, para identificar oportunidades de mejora.
- Identifica los lugares y posiciones donde los datos adicionales pueden ser recopilados e investigados.

-
- Ayuda a entender el proceso completo.
 - Permite comprender de forma rápida y amena los procesos.

Procedimiento de construcción

1. Establezca el alcance del proceso a describir. De esta manera quedará fijado el comienzo y el final del diagrama. Frecuentemente el comienzo es la salida del proceso previo y el final la entrada al proceso siguiente.
2. Identifique y liste las principales actividades/subprocesos que están incluidos en el proceso a describir y su orden cronológico. Si el nivel de detalle definido incluye actividades menores, listarlas también.
3. Identifique y liste los puntos de decisión.
4. Construya el diagrama respetando la secuencia cronológica y asignando los correspondientes símbolos.
5. Asigne un título al diagrama y verifique que esté completo y describa con exactitud el proceso elegido.

Reglas:

- Los Diagramas de flujo deben escribirse de arriba hacia abajo, y/o de izquierda a derecha
- Los símbolos se unen con líneas, las cuales tienen en la punta una flecha que indica la dirección que fluye la información procesos, se deben de utilizar solamente líneas de flujo horizontal o verticales (nunca diagonales).
- Se debe evitar el cruce de líneas, para lo cual se quisiera separar el flujo del diagrama a un sitio distinto, se pudiera realizar utilizando los conectores. Se debe tener en cuenta que solo se van a utilizar conectores cuando sea estrictamente necesario.
- No deben quedar líneas de flujo sin conectar
- Todos los símbolos pueden tener más de una línea de entrada, a excepción del símbolo final.
- Solo los símbolos de decisión pueden y deben tener mas de una línea de flujo de salida.

Ejemplo # 1: suma dos enteros

- **Algoritmo:** suma dos enteros
- **Descripción:** suma dos números enteros cualquiera
- **Constantes:**
- **Variables:** num1, num2, suma
- **INICIO**
 1. lee num1 y num2
 2. asigna a suma la suma de num1 y num2
 3. imprime el valor de suma
- **FIN**

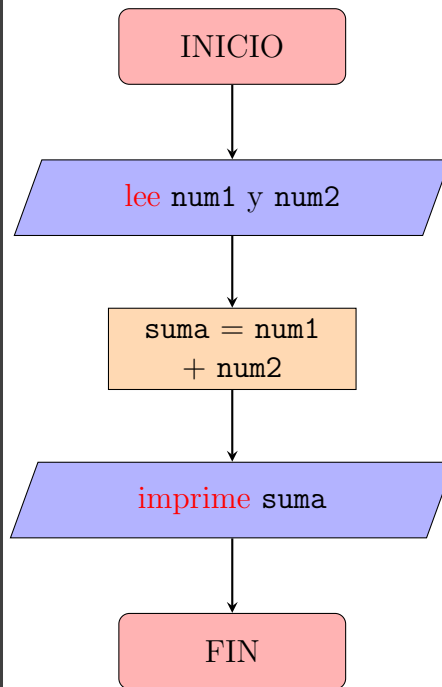


Figura 3: Pseudo-código y diagrama de flujo para el algoritmo “suma dos enteros”

Ejemplo # 2: promedia tres enteros

- **Algoritmo:** promedia tres enteros
- **Descripción:** determina el promedio de tres números enteros cualquiera
- **Constantes:**
- **Variables:** num1, num2, num3, suma, promedio
- **INICIO**
 1. lee num1, num2 y num3
 2. asigna a suma la suma de num1, num2 y num3
 3. asigna a promedio el valor de suma dividido entre tres.
 4. imprime el valor de promedio
- **FIN**

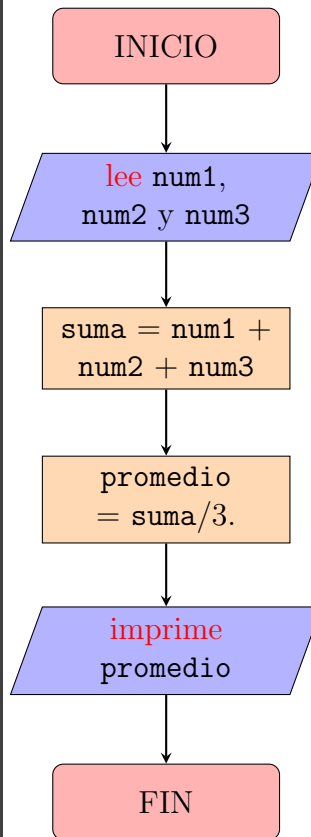


Figura 4: Pseudo-código y diagrama de flujo para el algoritmo “promedia tres enteros”

Ejemplo # 3: evalúa polinomio

- **Algoritmo:** evalúa polinomio
- **Descripción:** evalúa un polinomio en la variable x con coeficientes constantes $y = 2x^2 + 5x + 7$
- **Constantes:** 2,5,7
- **Variables:** x , y
- **INICIO**
 1. lee x
 2. asigna a y el valor de $2*x*x + 5*x + 7$
 3. imprime el valor de y
- **FIN**

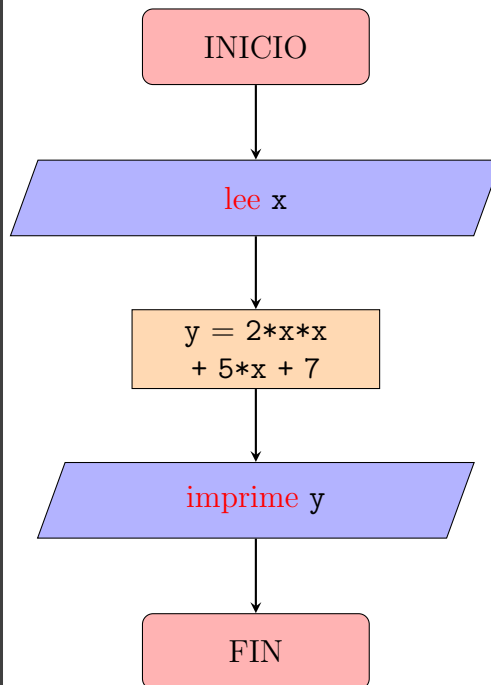


Figura 5: Pseudo-código y diagrama de flujo para el algoritmo “evalúa polinomio”

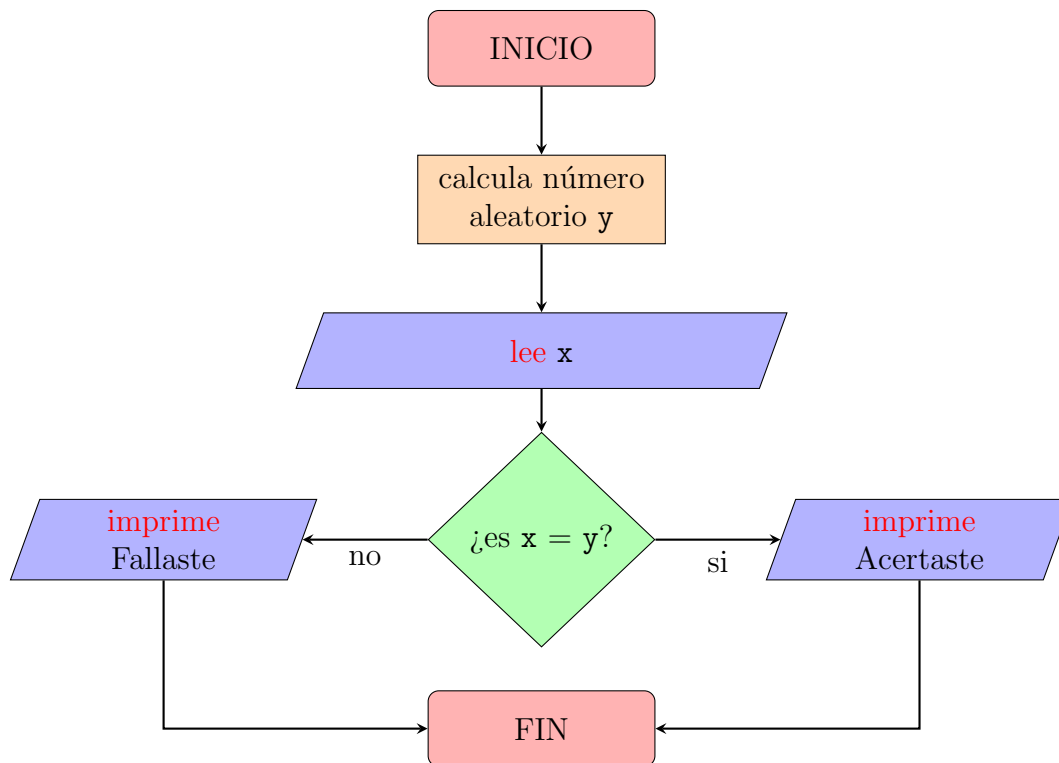


Figura 6: Diagrama de flujo para el algoritmo “adivina mi número”

Actividad 8.

Escribe el pseudo-código del algoritmo “adivina mi número”.

Actividad 9.

Haz un diagrama de flujo para determinar cuál de los dos números proporcionados por el usuario, es mayor.

Actividad 10.

Haz un diagrama de flujo para determinar cuál de los tres números proporcionados por el usuario, es mayor.

Trabajo A5. Entrega Antes de: Jueves 21 de Marzo de 2019 a las 23:59.

Hacer un diagrama de flujo para simular la compra de una bebida en una máquina dispensadora.

- Enviar por correo a ezequiel.arceo@my.uvm.edu.mx
- El asunto del correo será P00_TrabajoA5_NOMBRE_DEL_AUTOR
- El nombre de cada archivo adjunto será P00_TrabajoA5_NOMBRE_DEL_AUTOR.*

Ejemplo:

ASUNTO: P00_TrabajoA5_JUAN_PEREZ_FERNANDEZ

ADJUNTO: P00_TrabajoA5_JUAN_PEREZ_FERNANDEZ.cpp

6. Algoritmos I: Toma de decisiones

Dentro del cuerpo de una función colocamos un algoritmo. Un algoritmo es un “conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de un tipo de problemas.” Los bloques de construcción de un algoritmo son las variables, *las sentencias condicionales*, y *los ciclos*.

En esta lección vamos a ver los condicionales.

Si un programa no fuera más que una lista de órdenes a ejecutar de forma secuencial, una por una, no tendría mucha utilidad. Los condicionales nos permiten comprobar condiciones y hacer que nuestro programa se comporte de una forma u otra, que ejecute un fragmento de código u otro, dependiendo de esta condición.

6.1. if

La forma más simple de una sentencia condicional es un `if` (del inglés si) seguido de la condición a evaluar entre paréntesis, y entre llaves el código a ejecutar en caso de que se cumpla dicha condición.

```
1 // EJEMPLOS: sentencia if
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 void main()
7 {
8     string saborFavorito = "chocolate";
9     if ( saborFavorito == "chocolate" )
10     { // lo que sucede cuando la condición SI se cumple
11         cout << "Buen gusto!" << endl;
12     }
13 }
```

Como vez es bastante sencillo. No olvides colocar dentro de las llaves *todas* las instrucciones que quieras ejecutar cuando se cumple la condición.

6.2. if ... else

Vamos a ver ahora un condicional algo más complicado. ¿Qué haríamos si quisiéramos que se ejecutaran unas ciertas órdenes en el caso de que la condición no se cumpliera? En este caso usamos un `if/else`

```
1 // EJEMPLOS: sentencia if/else
2 #include <iostream>
3 #include <string>
4 using namespace std;
5
6 void main()
7 {
8     string saborFavorito = "chocolate";
9     if ( saborFavorito == "chocolate" )
10     { // lo que sucede cuando la condición SI se cumple
11         cout << "Buen gusto!" << endl;
12     }
13     else
14     { // lo que sucede cuando la condición NO se cumple
15         cout << "Para gustos se hicieron los sabores" << endl;
16     }
17 }
```

Observe que, en una sentencia `if/else`

- lo que sucede si la condición SI se cumple está entre las llaves después del `if`,
- lo que sucede si la condición NO se cumple está entre las llaves después del `else`.

Las condiciones más simples son las comparaciones de valor, para las cuales usamos *operadores relacionales*. Los operadores relacionales producen valores booleanos que son importantes como condiciones en las sentencias condicionales y en los ciclos.

Operador	Descripción/Pregunta	Ejemplos
$a == b$	¿son iguales a y b?	<code>5 == 3; // false</code>
$a != b$	¿son distintos a y b?	<code>5 != 3; // true</code>
$a < b$	¿es a menor que?	<code>5 < 3; // false</code>
$a > b$	¿es a mayor que?	<code>5 > 3; // true</code>
$a \leq b$	¿es a menor o igual que?	<code>5 ≤ 3; // false</code>
$a \geq b$	¿es a mayor o igual que?	<code>5 ≥ 3; // true</code>

```

1 // EJEMPLOS: Operadores relacionales
2 #include <iostream>
3 using namespace std;
4
5 void main()
6 {
7     int a = 7;
8     int b = 2;
9     cout << "[INFO] Operadores relacionales: " << endl;
10    // ¿son iguales a y b?
11    cout << "a == b = " << (a == b) << endl;
12    // ¿son distintos a y b?
13    cout << "a != b = " << (a != b) << endl;
14    // ¿es a menor que b?
15    cout << "a < b = " << (a < b) << endl;
16    // ¿es a mayor que b?
17    cout << "a > b = " << (a > b) << endl;
18    // ¿es a menor o igual que b?
19    cout << "a <= b = " << (a <= b) << endl;
20    // ¿es a mayor o igual que b?
21    cout << "a >= b = " << (a >= b) << endl;
22 }
23 // NOTA: los paréntesis alrededor de las expresiones relacionales
24 // no son estrictamente necesarios, pero ayudan a entender mejor.

```

6.3. Operadores lógicos

En ocasiones una condición simple no basta para resolver nuestro problema. Lo bueno es que las condiciones pueden combinarse.

El operador `&&` (que se lee como “y”) sirve para combinar dos condiciones de forma que ambas se deben cumplir al mismo tiempo para que el resultado sea verdadero.

El operador `||` (que se lee como “o”) sirve para combinar dos condiciones de forma que al menos una se debe cumplir para que el resultado sea verdadero.

Los operadores `&&` y `||` se describen completamente por las siguientes tablas de verdad:

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

a	b	a b
true	true	true
true	false	true
false	true	true
false	false	false

```
1 // EJEMPLOS: Operador &&
2 #include <iostream>
3 using namespace std;
4
5 void main()
6 {
7     int a = 7;
8     if ( (a > 0) && (a % 2 == 0) )
9     {
10         cout << "a es mayor que cero Y par" << endl;
11     }
12 }
```

```
1 // EJEMPLOS: Operador &&
2 #include <iostream>
3 using namespace std;
4
5 // && es muy útil para determinar
6 // si un número se encuentra en un rango.
7 void main()
8 {
9     int a = 7;
10    if ( (a > 2) && (a < 10) )
11    {
12        cout << "a es mayor que 2 Y menor que 10" << endl;
13    }
14 }
```

```

1  // EJEMPLOS: Operador ||
2  #include <iostream>
3  #include <string>
4  using namespace std;
5  void main()
6  {
7      string formaDePago = "efectivo"
8      if ( (formaDePago == "credito") || (formaDePago == "debito") )
9      {
10         cout << "forma de pago aceptada" << endl;
11     }
12     else
13     {
14         cout << "solamente aceptamos credito y debito" << endl;
15     }
16 }

```

```

1  // EJEMPLOS: Operador ||
2  #include <iostream>
3  #include <string>
4  using namespace std;
5  void main()
6  {
7      int numero = 9;
8      if ( (numero <= 0) || (numero >= 10) )
9      {
10         cout << "fuera de (0,10)" << endl;
11     }
12     else
13     {
14         cout << "en (0,10)" << endl;
15     }
16 }

```

```
1 // EJEMPLOS: Operador ||
2 #include <iostream>
3 #include <string>
4 using namespace std;
5 void main()
6 {
7     int numero = 9;
8     if ( (numero < 0) || (numero > 10) )
9     {
10         cout << "fuera de [0,10]" << endl;
11     }
12     else
13     {
14         cout << "en [0,10]" << endl;
15     }
16 }
```

Actividad 11.

Escriba un programa que

- pida un número entero al usuario
- imprima “su número es positivo” si el número es positivo,
- imprima “su número es negativo” si el número es negativo,
- imprima “su número es cero” si el número es cero,

Nota, el programa solamente debe imprimir una de estas tres opciones.

Actividad 12.

Escriba un programa que

- pida un número entero al usuario
- imprima “en (1,5)” si el número se encuentra en el rango (1,5),
- imprima “fuera de (1,5)” si el número NO se encuentra en el rango (1,5)
- use el operador `&&` en la condición compuesta.

Nota, el programa solamente debe imprimir una de estas tres opciones.

Actividad 13.

Escriba un programa que

- pida un número entero al usuario
- imprima “en (1,5)” si el número se encuentra en el rango (1,5),
- imprima “fuera de (1,5)” si el número NO se encuentra en el rango (1,5)
- use el operador `||` en la condición compuesta.

Nota, el programa solamente debe imprimir una de estas tres opciones.

Trabajo A6. Entrega Antes de: Lunes 11 de Marzo de 2019 a las 23:59.

Escriba un programa que

- pida al usuario su temperatura en grados Celcius (tipo **float**),
- imprima “Su temperatura es normal” si la temperatura es mayor que 36.1 grados, y menor que 37.2 grados (es decir, si la temperatura está en el rango (36.1,37.2))
- imprima “Visite a su médico” si la temperatura no es mayor que 36.1 grados, y menor que 37.2 grados, (es decir, si la temperatura NO está en el rango (36.1,37.2))
- **Enviar por correo a ezequiel_arceo@my.uvm.edu.mx**

-
- El asunto del correo será P00_TrabajoA6_NOMBRE_DEL_AUTOR
 - El nombre de cada archivo adjunto será P00_TrabajoA6_NOMBRE_DEL_AUTOR.*

Ejemplo:

ASUNTO: P00_TrabajoA6_JUAN_PEREZ_FERNANDEZ

ADJUNTO: P00_TrabajoA6_JUAN_PEREZ_FERNANDEZ.cpp

7. Algoritmos II: Repeticiones

Mientras que los condicionales nos permiten ejecutar distintos fragmentos de código dependiendo de ciertas condiciones, los bucles nos permiten ejecutar un mismo fragmento de código un cierto número de veces, mientras se cumpla una determinada condición.

7.1. for

El ciclo `for`, nos sirve para repetir un fragmento de código una cantidad conocida de veces. Requiere tres sentencias entre paréntesis

- la primera es el valor inicial de una variable “contador” para llevar cuenta de las repeticiones,
- la segunda es la condición necesaria para continuar,
- la tercera es una actualización el contador, generalmente haciéndolo avanzar.

El código a repetir va entre llaves después de los paréntesis.

El siguiente ejemplo nos imprime en pantalla los números enteros del 0 al 9.

```
1 // EJEMPLOS: ciclo for
2 #include <iostream>
3 using namespace std;
4
5 void main()
6 {
7     for( int contador = 0; contador < 10; contador++ )
8     { // lo que sucede mientras 'contador < 10' es cierto
9         cout << "El contador vale " << contador << endl;
10    }
11 }
```

```
1 // EJEMPLOS: ciclo for
2 #include <iostream>
3 using namespace std;
4
5 void main()
6 {
7     for( int i = 0; i < 10; i++ )
8     { // lo que sucede mientras 'i < 10' es cierto
9         cout << "El contador vale " << i << endl;
10    }
11 }
```

Con los ejemplos anteriores vemos que el nombre del contador no es relevante, y que podemos usar el valor del contador dentro del cuerpo del ciclo `for`.

El operador `++` después del nombre de una variable entera (por ejemplo `a++`) se llama operador de post-incremento. Primero devuelve el valor y luego lo incrementa en uno.

El operador `++` antes del nombre de una variable entera (por ejemplo `++a`) se llama operador de pre-incremento. Primero incrementa el valor en uno y luego lo devuelve.

```
1 // EJEMPLOS: operadores de incremento
2 #include <iostream>
3 using namespace std;
4
5 void main()
6 {
7     int a = 0;
8     int b = 0;
9     cout << "a++ = " << a++ << endl; // imprime 0
10    cout << "a = " << a << endl;      // imprime 1
11    cout << "++b = " << ++b << endl; // imprime 1
12    cout << "b = " << b << endl;      // imprime 1
13 }
```

El operador `--` después del nombre de una variable entera (por ejemplo `a--`) se llama operador de post-decremento. Primero devuelve el valor y luego lo decrementa en uno.

El operador `--` antes del nombre de una variable entera (por ejemplo `--a`) se llama operador de pre-decremento. Primero decrementa el valor en uno y luego lo devuelve.

```
1 // EJEMPLOS: operadores de decremento
2 #include <iostream>
3 using namespace std;
4
5 void main()
6 {
7     int a = 1;
8     int b = 1;
9     cout << "a-- = " << a-- << endl; // imprime 1
10    cout << "a = " << a << endl;      // imprime 0
11    cout << "--b = " << --b << endl; // imprime 0
12    cout << "b = " << b << endl;      // imprime 0
13 }
```

Actividad 14.

Escriba un programa que imprima en pantalla los números enteros del 0 al 20. Use un ciclo `for`.

Actividad 15.

Escriba un programa que imprima en pantalla los números enteros del 20 al 0. Use un ciclo `for`.

7.2. while

El ciclo `while` (mientras) ejecuta un fragmento de código mientras se cumpla una condición. Se escribe `while` seguido de la condición entre paréntesis, y el código a ejecutar entre llaves.

```
1 // EJEMPLOS: ciclo while
2 #include <iostream>
3 using namespace std;
4
5 void main()
6 {
7     int edad = 0; // acaba de nacer
8     while( edad < 18 )
9     {
10         cout << "Con " << edad << " años eres menor de edad." << endl;
11         edad++;
12     }
13     cout << "Felicidades, ya eres mayor de edad" << endl;
14 }
```

Con un ciclo `while` primero se evalúa la condición, y si esta se cumple se ejecuta la condición. Esto quiere decir que el fragmento puede no ejecutarse.

7.3. do/while

Si queremos ejecutar el fragmento antes de evaluar la condición usamos `do/while`.

El fragmento se ejecuta al menos una vez.

Actividad 16.

Escriba un programa que imprima en pantalla los números enteros pares del 0 al 20. Use un ciclo `while`.

Actividad 17.

Escriba un programa que imprima en pantalla los números enteros impares del 20 al 0. Use un ciclo `while`.

Trabajo A7. Entrega Antes de: Lunes 11 de Marzo de 2019 a las 23:59.

Escriba un programa que

- pida un número entero al usuario mientras,

```

1 // EJEMPLOS: ciclo while
2 #include <iostream>
3 using namespace std;
4
5 void main()
6 {
7     int edad = 1;
8     do
9     {
10         cout << "Con " << edad << " años eres menor de edad" << endl;
11         edad++;
12     }while( edad < 18 )
13     cout << "Felicidades, ya eres mayor de edad" << endl;
14 }

```

- imprima la suma de los números ingresados,
 - deje de pedir números cuando el usuario ingresa un número negativo,
 - imprima la cantidad de números positivos ingresados y la suma de ellos.
-
- Enviar por correo a ezequiel_arceo@my.uvm.edu.mx
 - El asunto del correo será P00_TrabajoA7_NOMBRE_DEL_AUTOR
 - El nombre de cada archivo adjunto será P00_TrabajoA7_NOMBRE_DEL_AUTOR.*

Ejemplo:

ASUNTO: P00_TrabajoA7_JUAN_PEREZ_FERNANDEZ

ADJUNTO: P00_TrabajoA7_JUAN_PEREZ_FERNANDEZ.cpp

8. Funciones II: Definición y Llamado

El ciclo de vida de una función consta de dos etapas: definición y llamado.

1. **Definición.** Es el momento en el que creamos a la función

La **definición de una función** con n argumentos tiene la forma:

```
tipoDeLaFuncion  nombreDeLaFuncion( $t_1$   $a_1, t_2$   $a_2, \dots, t_n$   $a_n$ )
{
    // sentencias
    return valorDeRetorno
}
```

donde

- a_1 es el nombre con el que la función se refiere a su primer argumento, y su tipo es t_1 ,
- a_2 es el nombre con el que la función se refiere a su segundo argumento, y su tipo es t_2 ,
- ...
- a_n es el nombre con el que la función se refiere a su n -ésimo argumento, y su tipo es t_n .

Al definir una función tenemos la libertad de elegir los nombres de sus argumentos.

2. **Llamado.** Es el momento en el que usamos a la función, es decir le pedimos que realice la tarea para la que fue creada.

El **llamada de función** requiere que escribamos el nombre de la función, seguido por **todos** los argumentos que requiere, entre paréntesis, con los mismos tipos que en la definición.

En las siguientes subsecciones ejemplificaremos la creación y uso de diversas funciones, clasificadas tomando en cuenta si tienen o no argumentos y valor de retorno

8.1. Funciones sin argumentos y sin valor de retorno

Quizás el tipo más “simple” de función es aquella sin argumentos y sin valor de retorno. Este tipo de función hace su trabajo sin usar datos, o usando información disponible “en su contexto” (vea la sección 9)

Ahora estamos listos para escribir nuestro primer programa en C++, será nuestro *hola mundo*, que simplemente imprimirá “Hola Mundo” en la pantalla

```

1 // EJEMPLO: imprime "Hola Mundo" en pantalla
2 // importa la librería iostream
3 #include <iostream>
4
5 // Usa el espacio de nombres estándar
6 using namespace std;
7
8 // Define la función principal de nuestro programa
9 void main() {
10     cout << "Hola Mundo" << endl;
11 }

```

Con cout podemos imprimir valores literales

```

1 // EJEMPLO: impresión de valores literales
2 #include <iostream>
3
4 using namespace std;
5
6 void main() {
7     cout << "Imprimo un int literal: " << 14 << endl;
8     cout << "Imprimo un float literal: " << 3.1416 << endl;
9     cout << "Imprimo una string literal: " << "Hola Mundo" << endl;
10    cout << "Imprimo un bool literal: " << true << endl;
11 }

```

Con cout podemos imprimir variables

```

1 // EJEMPLO: impresión de variables
2 #include <iostream>
3 #include <string> // importa la libreria string !!!
4
5 using namespace std;
6
7 void main() {
8     int miEntero = 14;
9     float miDecimal = 3.1416;
10    string miCadena = "Hola Mundo";
11    bool miBooleano = true;
12    cout << "Imprimo una variable int: " << miEntero << endl;
13    cout << "Imprimo una variable float: " << miDecimal << endl;
14    cout << "Imprimo una variable string: " << miCadena << endl;
15    cout << "Imprimo una variable bool: " << miBooleano << endl;
16 }

```

Con cin podemos pedir distintos tipos de variables

```
1 // EJEMPLO: petición de distintos tipos de variables
2 #include <iostream>
3 #include <string>
4
5 using namespace std;
6
7 void main() {
8     int miEntero;
9     float miDecimal;
10    string miCadena;
11    bool miBooleano;
12    cout << "Inserte un entero: ";
13    cin >> miEntero;
14    cout << "Su entero es: " << miEntero << endl;
15    cout << "Inserte un decimal: ";
16    cin >> miDecimal;
17    cout << "Su decimal es: " << miDecimal << endl;
18    cout << "Inserte una cadena: ";
19    cin >> miCadena;
20    cout << "Su cadena es: " << miCadena << endl;
21    cout << "Inserte un booleano: ";
22    cin >> miBooleano;
23    cout << "Su booleano es: " << miBooleano << endl;
24 }
```

Actividad 18.

Escriba una función sin argumentos y sin valor de retorno que imprima en pantalla lo siguiente en orden:

- Si nombre completo,
- Su edad,
- Su estatura en metros.

Use variables para guardar estos datos e imprima las variables.

Ejemplo:

Me llamo Ezequiel Arceo May

Tengo 34 años

Mido 1.57 metros.

8.2. Funciones sin argumentos y con valor de retorno

Un buen uso para este tipo de funciones es la adquisición de datos por parte del usuario

```
1 // EJEMPLO: petición de distintos tipos de variables
2 #include <iostream>
3 #include <string>
4
5 using namespace std;
6
7 float pideFloat()
8 {
9     float miDecimal;
10    cout << "Inserte un decimal: ";
11    cin >> miDecimal;
12    return miDecimal;
13 }
14
15 void main()
16 {
17     float miDecimal = pideFloat();
18     cout << "Usted insertó: " << miDecimal;
19 }
```

Actividad 19.

Escriba una función sin argumentos que pida una cadena al usuario, guarde el valor de la cadena en una variable, y devuelva esta variable como valor de retorno.

Actividad 20.

Escriba una función sin argumentos que pida un booleano al usuario, guarde el valor booleano en una variable, y devuelva esta variable como valor de retorno.

Trabajo B3. Entrega Antes de: Jueves 7 de Marzo de 2019 a las 23:59.

Escriba una función sin argumentos que pida un entero al usuario, guarde el valor entero en una variable, y devuelva esta variable como valor de retorno.

- Enviar por correo a ezequiel_arceo@my.uvm.edu.mx
- El asunto del correo será P00_TrabajoB3_NOMBRE_DEL_AUTOR
- El nombre de cada archivo adjunto será P00_TrabajoB3_NOMBRE_DEL_AUTOR.*

Ejemplo:

ASUNTO: P00_TrabajoB3_JUAN PEREZ FERNANDEZ

ADJUNTO: P00_TrabajoB3_JUAN PEREZ FERNANDEZ.cpp

8.3. Funciones con argumentos y sin valor de retorno

Estas funciones usan información del exterior, pero nadie depende de la información que generan. Un buen uso para este tipo de funciones es la impresión de datos como medida informativa

```
1 // EJEMPLO: petición de distintos tipos de variables
2 #include <iostream>
3 #include <string>
4 #include <vector>
5
6 using namespace std;
7
8 void ecoInt(int entero)
9 {
10     cout << "Su entero es: " << entero << endl;
11 }
12
13 void ecoString(string cadena)
14 {
15     cout << "Su cadena es: " << cadena << endl;
16 }
17
18 // Imprime todos los elementos de un vector de int's en pantalla
19 void ecoVecInt(vector<int> vec)
20 {
21     cout << "Los " << vec.size() << " elementos de su vector son:\n";
22     for(int i = 0; i < vec.size(); i++)
23     {
24         cout << vec[i] << endl;
25     }
26 }
27
28 void main()
29 {
30     // Llamada a la función ecoInt
31     int miEntero = 14;
32     ecoInt(miEntero);
33     // Llamada a la función ecoString
34     string miCadena = "Hola Mundo";
35     ecoString(miCadena);
36     // Llamada a la función ecoVecInt
37     vector<int> miVecInt= {1, 2, 3};
38     ecoVecInt(miVecInt);
39 }
```

Actividad 21.

Escriba una función cuyo argumento sea un número **float** y lo imprima en pantalla.

Actividad 22.

Escriba una función cuyo argumento sea un **vector** de valores **float**, y que imprima todos los elementos del vector en pantalla.

Trabajo B4. Entrega Antes de: Jueves 7 de Marzo de 2019 a las 23:59.

Escriba una función cuyo argumento sea un **vector** de valores **float**, y que imprima todos los elementos *negativos* del vector en pantalla.

- Enviar por correo a `ezequiel.arceo@my.uvm.edu.mx`
- El asunto del correo será `P00_TrabajoB4_NOMBRE_DEL_AUTOR`
- El nombre de cada archivo adjunto será `P00_TrabajoB4_NOMBRE_DEL_AUTOR.*`

Ejemplo:

ASUNTO: `P00_TrabajoB4_JUAN_PEREZ_FERNANDEZ`

ADJUNTO: `P00_TrabajoB4_JUAN_PEREZ_FERNANDEZ.cpp`

Trabajo B5. Entrega Antes de: Jueves 7 de Marzo de 2019 a las 23:59.

Escriba una función cuyo argumento sea un **vector** de valores **int**, y que imprima todos los elementos *pares* del vector en pantalla.

- Enviar por correo a `ezequiel.arceo@my.uvm.edu.mx`
- El asunto del correo será `P00_TrabajoB5_NOMBRE_DEL_AUTOR`
- El nombre de cada archivo adjunto será `P00_TrabajoB5_NOMBRE_DEL_AUTOR.*`

Ejemplo:

ASUNTO: `P00_TrabajoB5_JUAN_PEREZ_FERNANDEZ`

ADJUNTO: `P00_TrabajoB5_JUAN_PEREZ_FERNANDEZ.cpp`

8.4. Funciones con argumentos y con valor de retorno

Estas funciones requieren información del exterior, realizan sus procesos en base a ellos, y devuelven un valor que será utilizado en el ambiente en el que fueron llamadas.

```
1 // EJEMPLO: petición de distintos tipos de variables
2 #include <iostream>
3 #include <string>
4
5 using namespace std;
6
7 float minimoFloat(float a, float b)
8 {
9     if(a < b)
10     {
11         // Si a < b es cierto
12         return a;
13     }
14     else
15     {
16         // Si a < b es falso
17         return b;
18     }
19 }
20
21 void main()
22 {
23     float a = 1.3;
24     float b = 3.1;
25     cout << "Comparando " << a << " y " << b
26     << " el mínimo es " << minimoFloat(a,b) << endl;
27 }
```

Trabajo B6. Entrega Antes de: Jueves 7 de Marzo de 2019 a las 23:59.

Escriba una función con dos argumentos de tipo **vector** de valores **int**, cuyo valor de retorno sea el tamaño del vector más pequeño que le es dado como argumento. HINT: repase la definición de la función `ecoVecInt` para saber como definir una función con argumento de tipo vector.

- Enviar por correo a ezequiel.arceo@my.uvm.edu.mx
- El asunto del correo será `P00_TrabajoB6_NOMBRE_DEL_AUTOR`
- El nombre de cada archivo adjunto será `P00_TrabajoB6_NOMBRE_DEL_AUTOR.*`

Ejemplo:

ASUNTO: P00_TrabajoB6_JUAN_PEREZ_FERNANDEZ

ADJUNTO: P00_TrabajoB6_JUAN_PEREZ_FERNANDEZ.cpp

A continuación se muestran dos ejemplos: una función con argumento vector (int) y una función de tipo vector (int).

```
1  #include <iostream>
2  #include <string>
3  #include <vector>
4  using namespace std;
5
6  void imprimeVecInt(vector<int> vec)
7  {
8      for (int i = 0; i < vec.size(); i++)
9      {
10         cout << vec[i] << endl;
11     }
12 }
13
14 // Devuelve la factorización prima (factores)
15 // de su argumento (numero)
16 vector<int> primos(int numero)
17 {
18     vector<int> factores;
19     vector<int> primos = {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,
20     int indice = 0;
21     int divisor = 1;
22     int cociente = numero;
23     while ( cociente != 1 )
24     {
25         divisor = primos[indice];
26         cout << cociente << " es divisible entre " << divisor << " ?";
27         if ( cociente % divisor == 0 )
28         {
29             factores.push_back(divisor);
30             cout << " => SI" << endl;
31             cociente = cociente / divisor;
32         }
33         else
34         {
35             indice = indice + 1;
36             cout << " => NO" << endl;
37         }
38     }
39     return factores;
40 }
```

```

1  int main()
2  {
3      vector<int> factores1;
4      vector<int> factores2;
5      int numero1 = 12;
6      int numero2 = 15;
7      //cout << "Inserte un entero: ";
8      //cin >> numero;
9      //cout << "Su entero es: " << numero << endl;
10     factores1 = primos(numero1);
11     factores2 = primos(numero2);
12     cout << "Los factores primos del número " << numero1 << " son: " << endl;
13     imprimeVecInt(factores1);
14     cout << "Los factores primos del número " << numero2 << " son: " << endl;
15     imprimeVecInt(factores2);
16     //system("pause");
17     return 0;
18 }

```

Actividad 23.

Use la función `primos` para escribir una función que

- pida un entero al usuario,
- calcule la factorización prima del número ingresado (use `primos`)
- escriba en pantalla la factorización prima con el formato de los siguientes ejemplos

$$\begin{array}{rcl}
 8 & = & 2 * 2 * 2 \\
 12 & = & 2 * 2 * 3 \\
 15 & = & 3 * 5 \\
 & & \vdots
 \end{array}$$

Actividad 24.

Escriba una función llamada `encuentraEnVector` con las siguientes características:

- La función es de tipo **bool**.
- Tiene dos argumentos, un entero, y un vector de enteros.
- Devuelve **true** si el valor entero dado se encuentra en el vector de entero dado. Devuelve **false** si el valor entero dado NO se encuentra en el vector de entero dado.

NOTA: ni el número que se busca, ni el vector donde se busca se definen ni se piden con `cout/cin` dentro de la función `encuentraEnVector`. Son pasados a la función como argumentos.

Ejemplo de uso:

```
1 // líneas de código previo
2
3 void usaEncuentraEnVector()
4 {
5     // Inicialización de un vector con valores
6     vector<int> miVector = {1,2,3,5,6};
7     int miNumero;
8     bool encontrado = false;
9
10    miNumero = 2;
11    // caso en el que 'encuentraEnVector' devuelve 'true'
12    encontrado = encuentraEnVector(miNumero, miVector);
13    cout << "El " << miNumero << " se encuentra?: " << encontrado << endl;
14
15    miNumero = 4;
16    // caso en el que 'encuentraEnVector' devuelve 'false'
17    encontrado = encuentraEnVector(miNumero, miVector);
18    cout << "¿El " << miNumero << " se encuentra?: " << encontrado << endl;
19 }
```

Actividad 25.

Escriba una función llamada `cuentaEnVector` con las siguientes características:

- La función es de tipo **int**.
- Tiene dos argumentos, un entero, y un vector de enteros.
- Devuelve la cantidad (entera) de veces que se encontró al entero dentro del vector de enteros.

NOTA: ni el número que se cuenta, ni el vector donde se cuenta se definen ni se piden con `cout/cin` dentro de la función `cuentaEnVector`. Son pasados a la función como argumentos.

Ejemplo de uso:

```
1 // líneas de código previo
2
3 void usaCuentaEnVector()
4 {
5     // Inicialización de un vector con valores
6     vector<int> miVector = {1,2,3,5,3,6};
7     int miNumero;
8     int conteo = 0;
9
10    miNumero = 2;
11    // caso en el que 'cuentaEnVector' devuelve 1
12    conteo = cuentaEnVector(miNumero, miVector);
13    cout << "El " << miNumero << " se encontró " << conteo << "veces" << endl;
14
15    miNumero = 3;
16    // caso en el que 'cuentaEnVector' devuelve 2
17    conteo = cuentaEnVector(miNumero, miVector);
18    cout << "El " << miNumero << " se encontró " << conteo << "veces" << endl;
19
20    miNumero = 4;
21    // caso en el que 'cuentaEnVector' devuelve 0
22    conteo = cuentaEnVector(miNumero, miVector);
23    cout << "El " << miNumero << " se encontró " << conteo << "veces" << endl;
24 }
```

Trabajo B7. Entrega Antes de: Lunes 11 de Marzo de 2019 a las 23:59.

Escriba una función llamada `unicosEnVector` con las siguientes características:

- La función es de tipo vector `int`.
- Tiene un argumento, un vector de enteros.
- Devuelve un vector de enteros (nuevo) con los elementos únicos del vector que recibe como argumento.

NOTA: el vector sobre el que se buscan elementos únicos no se define ni se pide con `cout/cin` dentro de la función `unicosEnVector`. Solo el vector de elementos únicos se define dentro de la función.

Ejemplo de uso:

```
1 void usaUnicosEnVector() {  
2     // Inicialización de un vector con valores  
3     vector<int> miVector = {1,2,3,5,3,6};  
4     // Inicialización de un vector para coleccionar a los elementos únicos  
5     vector<int> unicos;  
6  
7     unicos = unicosEnVector(miVector);  
8     cout << "Los elementos únicos son: " << endl;  
9     // Imprime 1 2 3 5 6  
10    imprimirVecInt(unicos);  
11 }
```

- Enviar por correo a ezequiel_arceo@my.uvm.edu.mx
- El asunto del correo será `P00_TrabajoB7_NOMBRE_DEL_AUTOR`
- El nombre de cada archivo adjunto será `P00_TrabajoB7_NOMBRE_DEL_AUTOR.*`

Ejemplo:

ASUNTO: `P00_TrabajoB7_JUAN_PEREZ_FERNANDEZ`

ADJUNTO: `P00_TrabajoB7_JUAN_PEREZ_FERNANDEZ.cpp`

Trabajo B8. Entrega Antes de: Lunes 11 de Marzo de 2019 a las 23:59.

Use la función `primos` para escribir una función que

- pida un entero al usuario,
- calcule la factorización prima del número ingresado (use `primos`)
- escriba en pantalla la factorización prima con el formato de los siguientes ejemplos

$$\begin{array}{lcl} 8 & = & 2 \wedge 3 \\ 12 & = & 2 \wedge 2 * 3 \\ & & \vdots \end{array}$$

HINT: Use las funciones `unicosEnVector` y `cuentaEnVector`.

- Enviar por correo a ezequiel_arceo@my.uvm.edu.mx
- El asunto del correo será `P00_TrabajoB8_NOMBRE_DEL_AUTOR`
- El nombre de cada archivo adjunto será `P00_TrabajoB8_NOMBRE_DEL_AUTOR.*`

Ejemplo:

ASUNTO: `P00_TrabajoB8_JUAN_PEREZ_FERNANDEZ`

ADJUNTO: `P00_TrabajoB8_JUAN_PEREZ_FERNANDEZ.cpp`

Actividad 26.

Preparación para el primer examen parcial: defina las funciones

- `sumaFloat` de tipo **float**, con dos argumentos **float** que devuelva la suma de sus argumentos.
- `multiplicaFloat` de tipo **float**, con dos argumentos **float** que devuelva la multiplicación de sus argumentos.
- `potenciaFloat` de tipo **float**, con dos argumentos **float** que devuelva su primer argumento (base) elevado a la potencia de su segundo argumento (exponente). Se recomienda usar la función `pow` de la librería `cmath`.

```
1 // EJEMPLOS: Uso de la función pow
2 #include <stdio>
3 #include <cmath>
4 using namespace std;
5 void main ()
6 {
7     cout << "5.0 ^ 2.0 = " << pow(5.0, 2.0) << endl; // imprime 25.0
8     cout << "2.0 ^ 3.0 = " << pow(2.0, 3.0) << endl; // imprime 8.0
9 }
```

Use las funciones `sumaFloat`, `multiplicaFloat`, `potenciaFloat` para definir las siguientes funciones **float** de un argumento **float**

1. $f_1(x) = 2.0x + 3.0$

2. $f_2(x) = 5.0x^2 - x + 1.0$

3. $f_3(x) = (4.0x - 2.0)^{3.0}$

4. $f_4(x) = 2.0x + 3.0 + 5.0x^2 - x + 1.0 + (4.0x - 2.0)^{3.0}$

Tenga cuidado con el orden de operaciones al momento de definir las $f_i(x)$.

Calcule el valor de las funciones $f_i(x)$ en los puntos $x = 1.0$, y $x = 3.0$ dentro de la función `main`.

9. Datos II: donde es “visible/accesible” una variable/función

Un *ámbito* es una región de texto en el programa. Un identificador (de variable o función) se declara en un ámbito y es válido (“está en ámbito”) desde el punto de su declaración y hasta la primera llave que empareja con la llave de apertura antes de que la variable fuese definida.

```
1 void f() // inicia ámbito de 'f'
2 {
3     g(); // <- ERROR, 'g' no está en ámbito
4 }
5
6 void g() // inicia ámbito de 'g'
7 {
8     cout << "Hola desde g" << endl;
9 }
10
11 void h()
12 {
13     int x = y; // inicia ámbito de 'x'. ERROR, 'y' no está en ámbito
14     int y = x; // inicia ámbito de 'y'. OK, 'x' está en el ámbito
15     g(); // OK, 'g' está en ámbito
16 } // termina ámbito de 'x' y 'y'
17 // termina ámbito de 'f', 'g' y 'h'
```

El ejemplo anterior muestra cuándo las variables son visibles y cuando dejan de estar disponibles (es decir, cuando *salen del ámbito*). Una variable se puede utilizar sólo cuando se está dentro de su ámbito. Los ámbitos pueden estar anidados, indicados por parejas de llaves dentro de otras parejas de llaves. El anidado significa que se puede acceder a una variable en un ámbito que incluye el ámbito en el que se está.

```
1 void main()
2 {
3     int x = 2; // inicia ámbito de 'x'
4     if (x > 0)
5     {
6         int y = 2*x; // inicia ámbito de 'y'
7     } // termina ámbito de 'y'
8 } // termina ámbito de 'x'
```

Existen diversos tipos de ámbitos que usamos para controlar donde un identificador puede ser usado:

- *el ámbito global*: el ámbito más externo (fuera de todos los ámbitos). Por ejemplo, la función `main` siempre se define en el ámbito global.
- *el ámbito de espacio de nombres*: es un ámbito con nombre dentro del ámbito global o dentro de otro espacio de nombres. Un ejemplo de ámbito de espacio de nombres es `std`, el cual usamos al escribir `using namespace std`.
- *el ámbito local*: que se da entre llaves `{ ... }` en un **bloque** o en el cuerpo de una función.
- *el ámbito de sentencia*: por ejemplo al definir una variable de contador `int i = 0` dentro de un ciclo `for(int i=0; i < 10 ; i++){}`.
- *el ámbito de clase*: que es el área de código dentro de una clase. Este es el ámbito que nos interesará entender de ahora en adelante.

El propósito principal de un ámbito es mantener los nombres locales, de tal forma que no interfieran (**colisionen**) con nombres declarados en otro lugar. Por ejemplo:

```

1 void f(int x) // 'f' es global; 'x' es local a 'f'
2 {
3     int z = x+7; // 'z' es local a 'f'
4 }
5
6 int g(int x) // 'g' es global; 'x' es local a 'g'
7 {
8     int f = x+2; // 'f' es local a 'g'
9     return 2*f;
10 }
```

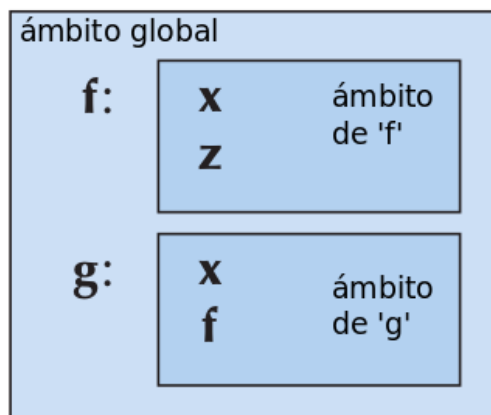


Figura 7: Aquí, la `x` de `f()` es diferente de la `x` de `g()`. Ambas `x` no *colisionan* porque no están en el mismo ambiente. Dos declaraciones incompatibles en el mismo ámbito se conocen como una **colisión**. Semejantemente, la `f` definida y usada dentro de `g()` no es la función `f()`.

Un ejemplo realista del ambiente local es el siguiente

```

1  int max(int a, int b) // "max" es global; "a" y "b" son locales a "max"
2  {
3      if(a > b)
4      {
5          return a;
6      }
7      else
8      {
9          return b;
10     }
11 }
12
13 int abs(int a) // "abs" es global; "a" es local a "abs",
14 { // y diferente de la "a" de "max"
15     if(a > 0)
16     {
17         return a;
18     }
19     else
20     {
21         return -a;
22     }
23 }

```

Con la notable excepción del ámbito global, los ámbitos sirven para mantener los nombre locales. Para la mayoría de los propósitos, la localidad de los nombres es buena, de tal forma que debemos mantener los nombres como locales en la medida de lo posible. Cuando declaro mis variables, funciones, etc, dentro de funciones, clases o espacios de nombres ellas no colisionarán con las tuyas. Recuerda, los programas *reales* tienen miles de identificadores. Entonces, los nombres locales hacen que el programa sea manejable.

Recomendaciones

- Evita definir variables con el mismo nombre dentro de ambientes anidados, promueve errores difíciles de encontrar,
- Evita definir funciones dentro de funciones, C++ no lo permite,
- El anidamiento es casi inevitable, trata de mantenerlo al mínimo,
- Usa nombres informativos proporcionalmente al tamaño del ámbito de una variable.
- Reduce el uso de variables globales: los buenos programas tienen muy pocas (si las tienen, tal vez una o dos) variables globales. Conforme el programa crece se vuelve cada vez más difícil saber donde se define o altera el valor

de una variable. Creame, el verdadero trabajo no es escribir un programa, sino mantenerlo (arreglarlo) y expandirlo.

9.1. Uso de ámbitos: declaración de funciones

Si usted está acostumbrado a usar variables globales y funciones `void` sin valor de retorno, es muy probable que defina todas las funciones auxiliares antes de la función `main`, y las llame dentro de `main`, como se muestra en el siguiente ejemplo:

```
1 // Ambito: función global ANTES de main - OK
2 #include <iostream>
3 using namespace std;
4
5 // Definición de la función doble
6 int doble(int a) // inicia el ámbito de doble
7 {
8     return 2*a;
9 }
10
11 int main()
12 {
13     int a = 3;
14     cout << "el doble de " << a << " es " << doble(a) << endl;
15     return 0;
16 }
17 // termina el ámbito de doble
```

Quizás habrá notado que si movemos la definición de la función `doble` hacia abajo de `main` el código no compila, **intente compilar y lea el error en pantalla**

```
1 // Ambito: función global DESPUÉS de main - ERROR
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     int a = 3;
8     cout << "el doble de " << a << " es " << doble(a) << endl;
9     return 0;
10 }
11
12 // Definición de la función doble
13 int doble(int a) // inicia el ámbito de doble
14 {
15     return 2*a;
16 }
17 // termina el ámbito de doble
```

¿Por qué el código anterior no compila?

Ahora añadamos la *declaración de función* para `doble` antes de `main`, sin mover ninguna de las dos definiciones:

```
1 // Ambito: función global DESPUÉS de main - OK
2 #include <iostream>
3 using namespace std;
4
5 // Declaración de la función doble
6 int doble(int a); // inicia el ámbito de doble
7
8 int main()
9 {
10     int a = 3;
11     cout << "el doble de " << a << " es " << doble(a) << endl;
12     return 0;
13 }
14
15 // Definición de la función doble
16 int doble(int a)
17 {
18     return 2*a;
19 }
20 // termina el ámbito de doble
```

¿Qué le sucede al ámbito de `doble` cuando usamos la declaración?

Trabajo B9. Entrega Antes de: Jueves 21 de Marzo de 2019 a las 23:59.

Llene la tabla con todos los nombres de variables y funciones dentro del siguiente programa

Identificador	variable/función	inicio de ámbito	fin del ámbito	ámbito que lo contiene
factores1	variable	37	47	main

- Enviar por correo a ezequiel_arceo@my.uvm.edu.mx
- El asunto del correo será `P00_TrabajoB9_NOMBRE_DEL_AUTOR`
- El nombre de cada archivo adjunto será `P00_TrabajoB9_NOMBRE_DEL_AUTOR.*`

Ejemplo:

ASUNTO: `P00_TrabajoB9_JUAN_PEREZ_FERNANDEZ`

ADJUNTO: `P00_TrabajoB9_JUAN_PEREZ_FERNANDEZ.cpp`

```

1  #include <iostream>
2  #include <vector>
3  using namespace std;
4  void imprimeVecInt(vector<int> vec)
5  {
6      for (int i = 0; i < vec.size(); i++) cout << vec[i] << endl;
7  }
8
9  vector<int> primos(int numero)
10 {
11     vector<int> factores;
12     vector<int> primos = {2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,
13     53,59,61,67,71,73,79,83,89,97,101};
14     int indice = 0, divisor = 1;
15     int cociente = numero;
16     while ( cociente != 1 )
17     {
18         divisor = primos[indice];
19         cout << cociente << " es divisible entre " << divisor << " ?";
20         if ( cociente % divisor == 0 )
21         {
22             factores.push_back(divisor);
23             cout << " => SI" << endl;
24             cociente = cociente / divisor;
25         }
26         else
27         {
28             indice = indice + 1;
29             cout << " => NO" << endl;
30         }
31     }
32     return factores;
33 }
34
35 void main()
36 {
37     vector<int> factores1, factores2;
38     int numero1, numero2;
39     cout << "Inserte un entero: "; cin >> numero1;
40     cout << "Inserte otro entero: "; cin >> numero2;
41     factores1 = primos(numero1);
42     factores2 = primos(numero2);
43     cout << "Los factores primos del número " << numero1 << " son: " << endl;
44     imprimeVecInt(factores1);
45     cout << "Los factores primos del número " << numero2 << " son: " << endl;
46     imprimeVecInt(factores2);
47 }
48 // fin del archivo

```

Ampliamos el acceso a educación de calidad global para formar personas productivas que agregan valor a la sociedad. 64