Universidad Autónoma del Estado de México

Facultad de Ciencias

Licenciatura en Física

Apuntes de Clase:
Lenguajes de Pogramación

Alumno:
Francisco Javier de la Cruz Lugo

Profesor:
Dr. Erik Mendoza de la Luz

Segundo Parcial

Periodo Escolar 2020B

Índice

1. Clase 28 de Octubre

1.1. Modularidad con el uso de funciones. Declaración de funciones y parámetros.

Al momento de crear funciones en el lenguaje de C++, se debe prestar atencion a la propiafunciónn y en la forma en que interactúa con otras funciones, tal como la función **`main()`**. Esto inluye transmitir datos en forma corrrecta a una función cuando es invocada y devolver valores de una función. Para ello, primero se describe la primera parte de la interfaz, transmitir datos a una función y hacer que la función recibida almacene y procese en forma correcta los datos transmitidos.

La funcón se invoca o utiliza, dando el nombre de la función y transmitiendole datos, como argumentos, en el paréntesis que sigue al nombre de la función.

```
Nombre de la función (Datos - transmitidos)
```

Algunas observaciones ocaracterísticas de este tipo de funciones son las siguientes:

- La función invocada debe ser capaz de aceptar los datos que le son transmitidos por la función que hace la llamada.
- Sólo después que la función invocada recibe con éxito los datos pueden ser manipulados éstos para producir un resultado útil.

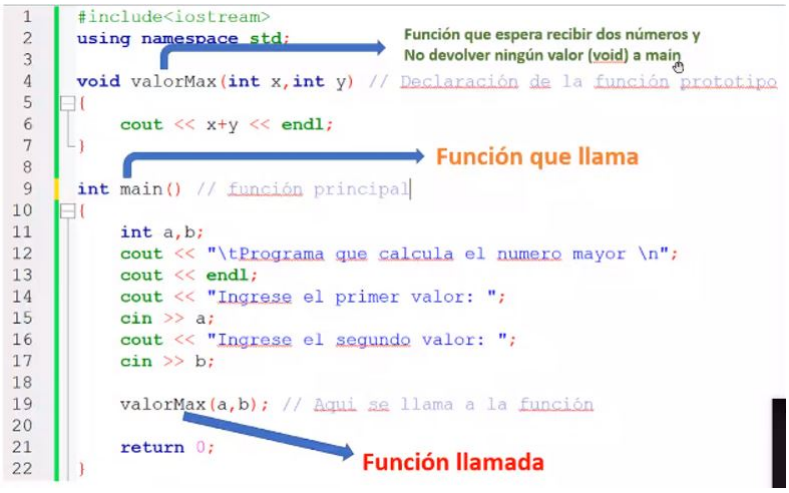


Figura 1: Ejemplo de función modular.

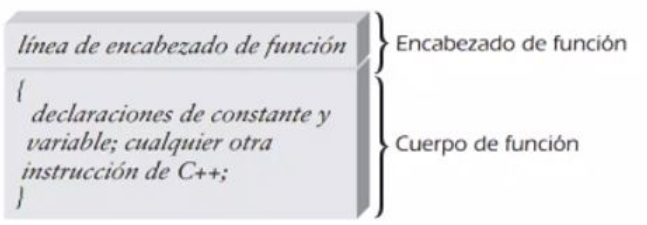
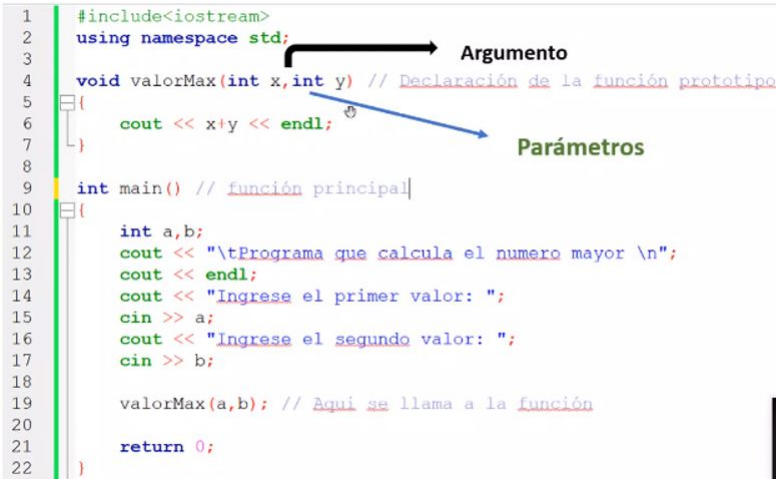


Figura 2: Estructura de una función modular.




```
1  #include<iostream>
2  using namespace std;
3
4  void valorMax(int x,int y) // Declaración de la función prototipo
5  {
6      cout << x+y << endl;
7  }
8
9  int main() // función principal
10 {
11     int a,b;
12     cout << "\tPrograma que calcula el numero mayor \n";
13     cout << endl;
14     cout << "Ingrese el primer valor: ";
15     cin >> a;
16     cout << "Ingrese el segundo valor: ";
17     cin >> b;
18
19     valorMax(a,b); // Aqui se llama a la función
20
21     return 0;
22 }
```

Figura 3: Parametros y argumentos de una función modular.

1.2. Programas Realizados.

- **programa#1.cpp**: Programa que utiliza una declaración de función modular para sumar dos numeros.
- **programa#2.cpp**: Programa que utiliza una declaración de función modular tipo prototipo para sumar dos numeros.
- **programa#3.cpp**: Programa que identifica un numero mayor mediante una función modular.
- **programa#4.cpp**: Programa que identifica un numero mayor mediante una función modular de tipo prototipo.
- **programa#5.cpp**: Programa que calcula suma, resta, producto, cociente de dos numeros usando una funcion modular.
- **programa#6.cpp**: Programa que calcula suma, resta, producto, cociente de dos numeros usando una funcion modular de tipo prototipo.

2. Clase 6 de Noviembre

2.1. Sobrecarga de Funciones.

En el código de un programa realizado en C++ es posible crearse funciones específicas que cumplan con una tarea única al ejecutar el programa, ya que durante este proceso se deben transmitir los datos de forma correcta para que la función pueda interpretarlos y operar con ellos adecuadamente. Sin embargo, muchas veces se debreán usarse diferentes tipos de datos dentro del programa por lo que la función creada deberá estar escrita de manera *generalizada* para que pueda interpretar el tipo de dato que se le introduce, para lograr esto es posible crear diferentes funciones con el mismo nombre y estar declaradas para los diferentes tipos de datos posibles, de esta forma al llamar la función, se empleará la función con el tipo de dato destinado, siendo en todos los casos la misma función pero destinada a diferentes tipos de dato a la vez.

Las figuras 4 y 5 muestran un ejemplo de sobrecarga de funciones en un programa. En el código de este se declaran tres funciones con el mismo nombre, sin embargo, cada una de ellas está destinada a un tipo de dato posible:

```
6   #include<iostream>
7   using namespace std;
8   // Declaración de la función prototipo
9   void suma(int ,int); // Entradas de tipo entero
10  void suma(float, float); // Entradas de tipo flotante
11  void suma(double, double); // Entradas de tipo doble precision
```

Figura 4: En el programa es tipeada la misma función prototipo para los diferentes tipos de datos posibles con los que pueda ser invocada en la función principal del programa.

```
44  void suma(int x,int y) // Inicio de la función
45  {
46      cout << "\nFuncion con entradas enteras: " << x + y << endl;
47  }
48
49  void suma(float x,float y) // Inicio de la función
50  {
51      cout << "\nFuncion con entradas flotantes: " << x + y << endl;
52  }
53
54  void suma(double x,double y) // Inicio de la función
55  {
56      cout << "\nFuncion con entradas doble precision: " << x + y << endl;
57  }
```

Figura 5: Más adelante, al final del código del programa son declaradas las funciones y la tarea en específico a realizar al ejecutar el programa.

2.2. Plantillas para función.

Una plantilla es una función única completa que sirve como modelo para una familia de funciones. Esta puede llegar a ser muy útil al escribir código debido a que ahorraría mucho trabajo y tiempo al emplearlas.

Estas plantillas para función, como la definición anterior lo indica, pueden invocarse las veces que sean necesarias sin necesidad de especificar el tipo de dato al que estarán destinadas a operar ya que se le asigna un tipo de dato en *general* que le permitirá adaptarse al tipo de datos que se le esté asignando de entrada. En gran parte, evitan la *sobrecarga de funciones* dentro del código de un programa.

Las figuras 6 y 7 muestran un ejemplo de dos plantillas para una función que devuelve el valor absoluto de un número, en ambas se utliza un prefijo para plantilla:

```
15  template < class T > // este es el prefijo de plantilla
16
17  void valor_abs(T num)
18  {
19      if (num < 0)
20      {
21          num = - num;
22      }
23      cout << "\nEl valor absoluto es: " << num << endl;
24      return ;
25  }
```

Figura 6: El prefijo de plantilla indica al compilador que la función que sigue es una plantilla que usa un tipo de datos.

```
15  template < class T > // este es el prefijo de plantilla
16
17  T valor_abs(T num)
18  {
19      T num_abs;
20      if (num < 0)
21      {
22          num_abs = - num;
23      }
24      else
25      {
26          num_abs = num;
27      }
28
29      return num_abs;
30  }
```

Figura 7: Este ejemplo de plantilla no utiliza `void`, como una forma alternativa de escribir plantillas para funciones.

2.3. Programas Realizados.

- **programa#7.cpp**: Ejemplo de programa con sobrecarga de funciones.
- **programa#8.cpp**: Ejemplo de programa que usa una plantilla de función que devuelve el valor absoluto de un número.
- **programa#9.cpp**: Ejemplo de programa que usa una plantilla de función alternativa que devuelve el valor absoluto de un número.
- **programa#10.cpp**: Programa que usa una plantilla de funcion llamada `maximo()` que devuelve el valor maximo de tres argumentos que se transmitan a la funcion cuando es llamada.
- **programa#11.cpp**: Programa que usa una plantilla de funcion llamada `alcuadrado()` que calcula y devuelve el cuadrado del argumento unico transmitido a la función cuando es llamada.

3. Clase 11 de Noviembre

3.1. Funciones de la clase String

La clase *string* es empleada para el uso de cadenas de caracteres. En la cual se requiere usar el siguiente encabezado:

```
#include <iostream>
#include <string>
```

Mismo que nos permitirá usar cadenas de caracteres en nuestro programa al momento de ejecutarlo. Incluso resulta ser de gran utilidad al momento de manipular ciertas características de las cadenas de caracteres o *strings* desde el código de nuestro programa. Las sintaxis requeridas para declarar variables con este tipo de datos son las siguentes:

Sintáxis 1:
```
string nombre_de_la_variable = "valor";
```
Ejemplo:
```
string Texto_1 = "texto muestra";
string Texto_2 = Texto_1;
string Texto_3 = Texto_1 + Texto_2;
```
.....(Concatenación)

Sintáxis 2:
```
string nombre_de_la_variable("valor");
```

Sintáxis 3 (subcadena a partir de una posición n):
```
string nombre_de_la_variable (Cadena_1,n);
```
.....donde n es un número entero

Sintáxis 4 (subacdena a partir de la posicion y un numero de caracteres):
```
string nombre_de_la_variable(cadena_1, posicion inicial, numero de caracteres);
```

Sintáxis 5 (Copias de una variable de tipo char n veces):
```
string nombre_de_la_variable(n, char);
```

Sintáxis 6:
```
string nombre_de_la_variable;
```

Sintáxis alternativa 1 para introducir la cadena como usuario:
```
getline(cin,nombre_de_la_variable);
```
Esta instruccion permite capturar todos los caracteres de la cadena que se introducen.

Sintáxis alternativa 2 para introducir la cadena como usuario:
```
getline(cin,nombre_de_la_variable,'caracter de termino del mensaje');
```
Esta instruccion permite capturar todos los caracteres de la cadena que se introducen usando el caracter entre comillas simples para indicar que la cadena introducida ha terminado.

3.2. Programas Realizados.

- **programa#12.cpp**: Este programa emplea las sintaxis presentadas para denotar variables de tipo `string`.
- **programa#13.cpp**: Este programa es una mejora del **programa_39.cpp** en el cual se permite introducir cadenas de caracteres mayores a 20 caracteres.

4. Clase 13 de Noviembre

4.1. Solucion del problema del delimitador de la cadena

En algunos códigos de programas se presenta el problema de que al ejecutarlo este le pide a un usuario ingresar cierta información desde el teclado, sin embargo, en ocasiones cuando hay líneas de código seguidas que piden ingresrar información desde el teclado algunas de estas llegan a ignorarse al presionar la tecla *enter* por lo que el programa termina por no ejecutarse como se espera. La solución a este problema viene dado por la línea de código que se presenta a continuación:

```
cin.ignore();
```

La cual solo debe ser colocada entre las líneas seguidas que piden información. De esta manera al teclear *enter* el programa se ejecutará correctamente.

4.2. Instrucciones para manipular *strings* o cadenas de caracateres

Dentro de los datos de tipo `string` o también conocidos como cadenas de caracteres podemos realizar varias manipulaciones de los datos de este tipo con las siguientes instrucciones:

Para medir la longitud (número de caracteres) de la cadena se usa la instrucción:

- `length()`
  Sintaxis: `int(nombre-del-string.length());`
- `size()`
  Sintaxis: `int(nombre-del-string.size());`

Para conocer el caracter que se encuetra en la posicion n se usa la instrucción:

- `at(n)`
  Sintaxis: `nombre-del-string.at(n);`

Para eliminar caracteres a partir de un indice en particular usamos:

- `erase()`
  Sintaxis:
  `nombre-del-string.erase(indice,no. de caracteres a eliminar);`
  `nombre-del-string.erase(indice);` (Elimina los caracteres a partir de este indice)

Para buscar un caracter o un conjunto de estos en una cadena usamos:

- `find()`
  Sintaxis: `nombre-del-string.find(çonjunto-de-cracteres");` (Funciona para la primera vez que la encuentra en el `string`)

Para insertar una cierta cadena en otra usamos la instruccion:

- `insert()`
  Sintaxis: `nombre-del-string.instert(indice,"texto-a-insertar");`

Para eliminar y reemplazar caracteres de una cadena por una subacadena usamos:

- `replace()`
  Sintaxis: `nombre-del-string.replace(posicion,numero de caracteres a eliminar,"texto sustituto");`

4.3. Programas Realizados.

- **programa#14.cpp**: Este programa emplea la solucion propuesta al problema del delimitador de la cadena.
- **programa#15.cpp**: Este programa emplea las instrucciones propuestas para manipular strings o cadenas de caracateres.
- **programa#16.cpp**: Programa que cuenta el numero de vocales en la cadena "Hola alumnos como estan".

5. Clase 18 de Noviembre

5.1. Alguna técnicas de encriptación de mensajes.

El uso frecuente y adecuado de la bibiloteca de C++ para los datos de tipo `string` puede ser de gran ayuda al momento de escribir cófigos que permitan encriptar mensajes, o en su caso, desencriptarlos conociendo la técnica de encriptación empleada. Para los programas que se realizan en esta sección del curso se emplean técnicas de encriptación básicas como el cifrado estilo *César* o el *Vigenere* que se verán adelante.

5.2. Programas realizados

- `programa#17.cpp`: Este programa desencripta el mensaje ZKTLMNR. Está escrito principalmente para este mensaje ya que la estructura del miismo se basa en la sustitución de lagunas letras del alfabeto.
- `programa#18.cpp`: Este programa permite encriptar un mensaje cambiando el orden de las letras del alfabeto en el código de este.
- `programa#18.cpp`:Este programa permite desencriptar un mensaje cambiando el orden de las letras del alfabeto en el código de este.

6. Clase 20 de Noviembre

6.1. Código ASCII

El código ASCII es un código de caracteres basado en el alfabeto latino, tal como se usa en inglés moderno. En lenguajes de programación este código puede ser empleado al monento de programar ya que mediante el uso de este se pueden facilitar la manipulación de datos, un buen ejemplo de esto es la encrpiatción de mensajes de una forma más compleja. Esto debido a que asocioa a cada caracter empleado de forma convencional con un numero entero con el cuál, mediante algunas operaciones aritméticas puede ser reemplazado por otro.

Caracteres ASCII de control

00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sínc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

ASCII extendido (Página de código 437)

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	|	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

Figura 8: Caracteres asiciados a un entero en el Código ASCII.

6.2. Programas realizados

- `programa#20.cpp`: Este programa permite codificar un mensaje tomando el valor de cada caracter del mensaje o cadena (`string`) y realizando un desplazamiento en el alfabeto.
- `programa#21.cpp`: Este programa permite decodificar un mensaje tomando el valor de cada caracter del mensaje o cadena (`string`) y realizando el desplazamiento correcto en el alfabeto.

- **programa#22.cpp**: Este programa permite codificar un mensaje tomando como base una palabra (cifrado Vigenere) tomando el valor de cada caracter en ASCII del mensaje o cadena con la letra de la palabra clave (`string`) y realizando un desplazamiento en el alfabeto.
- **programa#23.cpp**: Este programa permite decodificar un mensaje tomando como base una palabra (cifrado Vigenere) tomando el valor de cada caracter en ASCII del mensaje o cadena con la letra de la palabra clave (`string`) y realizando un desplazamiento en el alfabeto.

7. Clase 23 de Noviembre

7.1. Programas realizados

- **programa#24.cpp**: Este programa unifica los programas 22 y 23 en uno solo perimitiendo al usuario elegir entre codificar o decodificar un mensaje las veces que el usuario desee.

8. Clase 25 de Noviembre

8.1. La biblioteca de clase `cctype`.

Esta biblioteca de clase puede emplearse para tomar información acerca de los datos que se le pide analizar. En la clase `string` puede analizar y categorizar cada caracter de una cadena ingresada asiciandole cierta información. Mediante esto se pueden manipular los datos de una cadena ingresada. Esta debe invocarse en el encabezado del código del programa para ser utilizada:

```
#include <cctype>
```

8.2. Instrucciones para datos con `cctype`.

Algunas instrucciones para obtener información a partir de las cadenas `(string)` y con los cuales se pueden manipular datos:

- La siguiente instruccion identifica si el caracter es un dígito (un numero entero):
  `isdigit(variable_a_identificar);`
- La siguiente isntrucion identifica si el caracter es una letra:
  `isalpha(variable_a_identificar);`
- La siguiente instrucción identifica si el caracter es una letra o un digito:
  `isalnum(variable_a_identificar);`
- La siguiente instruccion indica si un caracter es una letra mayúscula:
  `isupper(variable_a_identificar);`
- La siguiente instruccion indica si un caracter es una letra minúscula:
  `islower(variable_a_identificar);`
- La siguiente instruccion indica si el caracter es un espacio vacío
  `isspace(variable_a_identificar);`

8.3. Programas realizados

- **programa#25.cpp**: Este es un programa prueba donde se utilizan las instrucciones mostradas con `cctype` para dar un demostrción de su funcionamiento.

9. Clase 27 de Noviembre

9.1. Programas realizados

- **programa#26.cpp**: Este programa cuenta el número de palabras en una cadena ingresada por el usuario.
- **programa#27.cpp**: Este programa cuenta el número de caracteres en una cadena ingresada por el usuario.

10. Clase 02 de Diciembre

10.1. Acentos ortográficos en C++

Durante la escritura de un programaa en el lenguaje de porgramación C++ no es muy común que estos sean escritos en un idioma que reconozca algunos símbolos ortográficos ya sea en su código o bien durante su ejecución. En la mayoría de los programas realizados anteriormente no se hace uso de los símbolos ortográficos en el español como son los acentos, para ello C++ incluye una biblioteca que permite hascer uso de ellos en los programas escritos en este lenguaje:

```
#include <locale.h>
```

Esta bibioteca puede usarse dentro del códiugo de un programa en C++ usando en la función principal de este el siguiente comando:

```
setlocale (LC_CTYPE, "Spanish");
```

De esta forma se puede hacer uso de acentos dentro del programa escrito.

10.2. Validación de datos

El uso de la bibioteca para datos de tipo `string` en C++ trae consigo una gran variedad de intrucciones empleables al momento de manipular datos a través de ellas. Por ejemplo, al momento de almacenar en cierta variable una cadena de números como dato ingresado por parte de un ususario, estos números pueden ser manipulables para obtener otr tipo de información, es decir, se harían ciertas operaciones con esos números, sin embargo esto no sería posible debido a que los datos fueron almacenados como cadena de caracteres y no como un número. Para ello se tienen las siguientes instrucciones para manipular números dentro de cadenas de caracteres:

- La siguinte instruccion convierte la expresión entre parentesis que es de tipo STRING en un número entero:
  `atoi(variable-STRING.c_str());`
  Cabe destacar que esta función detecta cada número en la cadena recorriendola cada lugar en su longitud. Si encuentra un caracter diferente de un número (incluso si este está antes de un número), entonces la instrucción se detiene automáticamente.
- La siguinte instruccion convierte la expresión entre parentesis de tipo STRING en un número de doble presición:
  `atof(variable-STRING.c_str());`
  Cabe destacar que esta función detecta cada número en la cadena recorriendola cada lugar en su longitud. Si encuentra un caracter diferente de un número (incluso si este está antes de un número), entonces la instrucción se detiene automáticamente.
- La instrucción itoa(expresión_entero) convierte un número entero a una cadena:
  `atof(variable-STRING.c_str());`
  Cabe destacar que esta función detecta cada número en la cadena recorriendola cada lugar en su longitud. Si encuentra un caracter diferente de un número (incluso si este está antes de un número), entonces la instrucción se detiene automáticamente.

10.3. Programas realizados

- **programa#28.cpp**: En este programa se muestran las funciónes e isntrucciones para convertir cadenas y se muestra como funcionan.
- **programa#29.cpp**: Este programa emplea las instrucciones anteriores para crear un programa que detecta cuando se ha ingresado un número no permitido, es decir, si tiene letras y otros símbolos no válidos.

11. Clase 04 de Diciembre

11.1. La función `try - catch`

.

"Las excepciones son un mecanismo de C++ para capturar errores que se produccen en tiempo de ejecución del programa, un programa puede estar bien hecho pero por causas externas (en este caso el usuario) pueden producirse errores". Para capturar errores o excepciones se puede usar la función `try - catch`.

La sintáxis de esta función es la siguente:

```
try
{
    código que lanza la excpeción
}
catch (expresión 1)
{
    se produce una excepción 1, la atrapa y la manajea
}
catch (expresión 2)
{
    se produce una excepción 2, la atrapa y la manajea
}
⋮
```

Si dentro de la función se detecta un error, lanzamos una excepción poniendo la palabra reservada `throw`.

11.2. Programas realizados

- **programa#30.cpp**: Este programa comprueba que cierta cadena de caratceres ingresada por el usuario pueda ser convertida en un número a partir de ciertas características.
- **programa#31.cpp**: Programa que detecta el caracter que invalida que la cadena sea convertida a entero.
- **programa#32.cpp**: Programa que elimina los espacios antes y después en una cadena de números para convertirla a entero.
- **programa#33.cpp**: Programa que comprueba dígito uno a uno para después concatenarlo y convertir la cadena a entero.
- **programa#34.cpp**: Programa que comprueba si el número que se ingreasa es de doble presición. Considerando un signo al inicio del número

12. Clase 7 de Diciembre

12.1. Creación de un Biblioteca de Funciones

Para crear una biblioteca de funciones en C++ para que posteriormente se puedan utilizar dichas funciones en otros programas sin la nececsidad de escribir esas funciones dentro del código del programa ayuda a ahorrar tiempo al momento de escribir el código del programa. Los pasos para crear una bibioteca de funciones es la siguiente:

1. Encapsular todas las funciones en uno o más espacios de nombres `(namespace)`.
2. Almacenar el código completo en uno o más archivos.

La sisntáxis para crear una biblioteca de funciones es:

```
namespace (Nombre_a_asignar)
{
    Colocar las funciones o clases a asignar;
}
```

12.2. Programas realizados

- **programa#35.cpp**: CREACIÓN DE UNA BIBIOTECA PERSONAL DE FUNCIONES
- **programa#36.cpp**: USO DE UNA BIBIOTECA PERSONAL DE FUNCIONES
- **programa#37.cpp**: Creación de funciones para una bibioteca personal de funciones
- **programa#38.cpp**: Creación de la biblioteca de funciones. Ejercicios 5c y 6c (Examen)
- **programa#39.cpp**: Creación de la biblioteca de funciones. Ejercicios (Examen)