

Universidad Autónoma del Estado de México



Facultad de Ciencias
Licenciatura en Física

Apuntes de Clase:
LENGUAJES DE PROGRAMACIÓN

Alumno:
Francisco Javier de la Cruz Lugo

Profesor:
Dr. Erik Mendoza de la Luz

PRIMER PARCIAL
Periodo Escolar 2020B

Índice

1. Clase 4 de Septiembre	3
1.1. Introducción a los Programas de Computadora	3
1.2. ¿Qué es Python?	4
1.3. ¿Qué es C++?	4
2. Clase 7 de Septiembre	4
2.1. Algoritmos	4
2.2. Pseudocódigo	5
2.3. Programas Modulares	5
2.4. La función <code>main()</code>	5
2.5. El objeto <code>cout</code>	5
3. Clase 9 de Septiembre	5
3.1. Programa Hola Mundo en Python	5
3.2. Programas Realizados:	7
4. Clase 11 de Septiembre	7
4.1. Tipos de datos en C++	7
4.2. El carácter de escape	8
4.3. Programas Realizados:	8
5. Clase 18 de Septiembre	9
5.1. Más tipos de datos en los lenguajes de Programación	9
5.2. Tipos de datos de punto flotante (float)	9
5.3. Programas Realizados:	9
6. Clase 21 de Septiembre	10
6.1. Operadores de asignación	10
6.2. Programas realizados	11
7. Clase 23 de Septiembre	11
7.1. Dar fromato a números para salida del programa. Manipu- ladores	11
7.2. Procesador <code>iomanip</code>	13
7.3. Programas realizados	13
8. Clase 25 de Septiembre	14
8.1. Programas realizados	14
9. Clase 28 de Septiembre	14
9.1. Las instrucciones <code>if - else</code>	14
9.2. Operadores lógicos	16
9.3. Programas realizados	16
10.Clase 30 de Septiembre	17
10.1 Instrucciones <code>if</code> anidadas	17
10.2La instrucción <code>switch</code>	18
10.3Programas realizados	19

11.Clase 2 de Octubre	19
11.1.Ciclos	19
11.2.Ciclos while	20
11.3.Programas realizados	20
12.Clase 7 de Octubre	20
12.1.Ciclos for	20
12.2.Programas realizados	21
13.Clase 14 de Octubre	21
13.1.Ciclos do - while	21
13.2.Programas realizados	21
14.Clase 16 de Octubre	21
14.1.Programas realizados	21
15.Clase 19 de Octubre	21
15.1.Programas realizados	21
16.Clase 28 de Octubre	22
16.1.Modularidad con el uso de funciones.	
Declaración de funciones y parámetros.	22
16.2.Programas Realizados.	24

1. Clase 4 de Septiembre

1.1. Introducción a los Programas de Computadora

Podemos entender un *programa de computadora* como un conjunto independiente de instrucciones interpretadas con el fin de realizar una tarea en específico. A estos programas también suele conocerse por el nombre de *software*. Al proceso de escribir y dictar un software o programa se le llama *programación*, mientras que al conjunto que dicta se le conoce como *lenguaje de programación*.

Lenguaje Máquina: Los programas escritos en este tipo de lenguaje son los únicos que una computadora es capaz de interpretar y posteriormente ejecutar. Estos *programas ejecutables* se componen de secuencias de instrucciones escritas en código binario, mismas que constan de dos partes; unas de instrucción u operación (*opcode*) como sumar, restar, multiplicar, etc. Y una parte de dirección de los datos a utilizar.

Lenguaje Ensamblador: Este lenguaje funciona sustituyendo los símbolos de palabras (ADD, SUB, MUL) por *opcodes* binarios y los números decimales, y las etiquetas por las direcciones en memoria.

Para que una computadora pueda ejecutar un programa en *lenguaje ensamblador* primero este debe traducirse a *lenguaje máquina* debido a que una computadora solo entiende este lenguaje. Al programa traductor se le llama *ensamblador*.

Lenguaje de bajo nivel: Usan instrucciones que se vinculan directamente con un tipo de computadora.

Lenguaje de alto nivel: Utilizan gran variedad de instrucciones similares a idiomas cotidianos (inglés), además de que son ejecutables en diferentes tipos de computadoras.

Los *programas o códigos fuente* son programas escritos en un lenguaje, ya sean de bajo o alto nivel. Sin embargo, estos también deben ser traducidos por un *lenguaje ensamblador* al lenguaje máquina de la computadora para que este pueda ejecutarse.

Cuando una declaración de un programa fuente de alto nivel es traducida individualmente y ejecutada de inmediato, el lenguaje de programación usado en este caso es un *lenguaje interpretado* y el lenguaje traductor se le llama *intérprete*. Por otro lado, si las instrucciones de un programa fuente de alto nivel son traducidas como una unidad completa antes de ejecutar cualquier declaración, entonces se trata de un *lenguaje compilado* y su programa traductor se llama *compilador*.

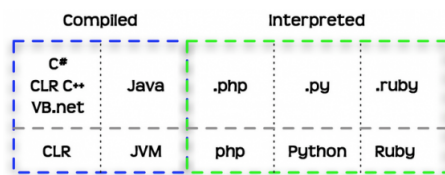


Figura 1: Ejemplos de lenguajes interpretados y compilados.

Una ventaja de los lenguajes compilados es que su ejecución es más rápida, aunque por otro lado, los lenguajes interpretados son más flexibles y portables.

La característica de *tipado dinámico* en un lenguaje de programación se refiere a que no es necesario declarar el tipo de dato que va a contener una variable, el tipo de este se determinará al momento de la ejecución de acuerdo al tipo de valor asignado. Además, el tipo de esta variable puede cambiar si se le asigna algún otro tipo de valor.

Un lenguaje *fuertemente tipado* no permite tratar una variable como si fuera un tipo de dato distinto al que se le asigna en un principio, por lo que es necesario convertir de forma explícita dicha variable en un nuevo tipo previamente. Por ejemplo, no se puede tratar una

variable de cadena de caracteres tipo `STRING` (como un número y esperar sumar el carácter ("9") con el número (8), esto podría generar varios errores en un código de programa.

1.2. ¿Qué es Python?

Python es un lenguaje de programación creado a principios de los 90's por Guido van Rossum, posee una sintaxis muy sencilla por lo que un código en este lenguaje es bastante legible. Se trata de un lenguaje interpretado con tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos.

El intérprete de Python es multiplataforma por lo que no son necesarias librerías específicas para poder ejecutar un programa en distintos sistemas operativos. La orientación a objetos en programación nos permite trasladar conceptos del mundo real a clases y objetos de nuestro programa. Su ejecución consiste en interacciones entre los objetos. En Python también se permite la programación imperativa, programación funcional y programación orientada a aspectos.

1.3. ¿Qué es C++?

Este lenguaje de programación fue creado en 1979 por Bjarne Stroustrup cuando comenzó a trabajar en un lenguaje llamado *C con clases*. A diferencia de Python este es un lenguaje compilado con una sintaxis heredada por el lenguaje C, está fuertemente tipado y puede trabajarse para desarrollar programación orientada a objetos, cuenta con una biblioteca estándar que suele venir con el compilador, posee compatibilidad con el lenguaje C por lo que un compilador de C++ puede compilar un código en C, además de también ser multiplataforma.

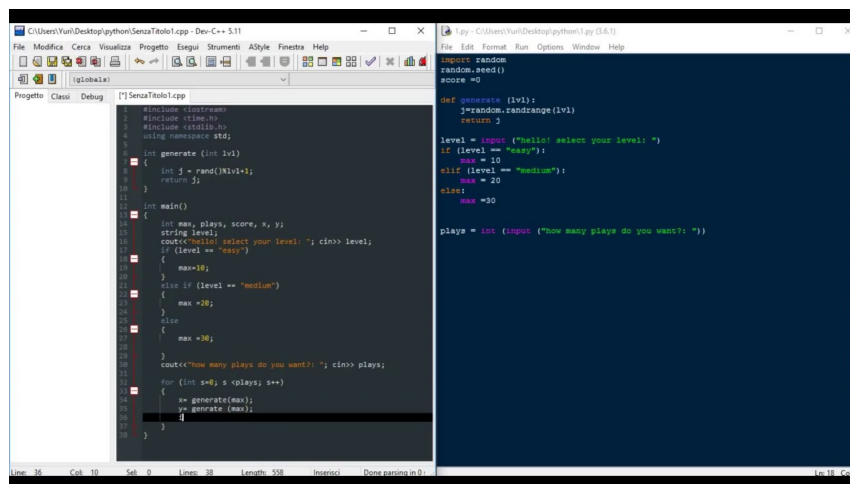


Figura 2: Sintaxis de un mismo programa escrito en C++ (izquierda) y en Python (derecha).

2. Clase 7 de Septiembre

2.1. Algoritmos

la palabra **algoritmo** hace referencia a la secuencia de pasos e instrucciones a seguir y que describen como es que deben procesarse los datos para producir ciertas salidas específicas. Un algoritmo responde a la pregunta "¿Qué método se usará para resolver este problema?".

2.2. Pseudocódigo

La palabra **pseudocódigo** alude a *la forma en la que se que debe realizar un programa en la forma más parecida al lenguaje de programación*. Representa por pasos la solución que tiene para un problema o algoritmo de la forma más detallada posible usando un lenguaje cercano al de programación. Este pseudocódigo no es ejecutable en un ordenador, ya que como indica su nombre, es un código falso cuya finalidad es ser entendido por el ser humano y no por una máquina.

2.3. Programas Modulares

Consisten programas cuyas estructuras se forman por segmentos interrelacionados, organizados en orden lógico y fácilmente comprensibles para formar una unidad integrada completa. Por otro lado, a los segmentos más pequeños usados para cosntruir un programa modular reciben el nombre de **módulos**. Cada uno de estos módulos esta diseñado y desarrollado para realizar una tarea específica y se trata de un subprograma pequeño en sí mismo. En C++ los módulo pueden ser *clases* o *funciones*.

2.4. La función main ()

La función main() se conoce como **función controladora** ya que controla o indica a los otros módulos la secuancia en la que deben ser ejecutados.

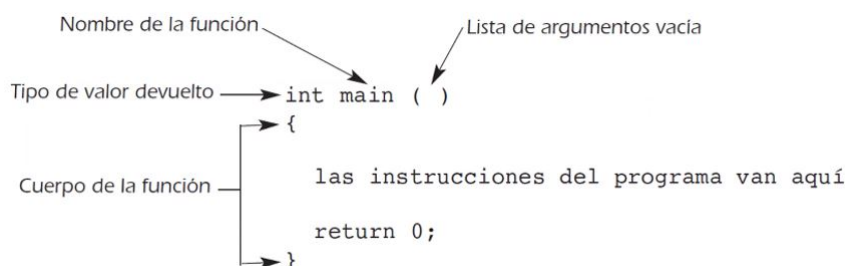


Figura 3: La primera línea de la función se conoce como encabezado de la función.

2.5. El objeto cout

Se trata de uno de los objetos más versátiles de C++. Este objeto, cuyo nombre se deriva de *Console OUTPUT* es un objeto de salida que envía los datos introducidos en él al dispositivo estándar de salida.

3. Clase 9 de Septiembre

3.1. Programa Hola Mundo en Python

Como ejemplo de un programa en Python tenemos el famoso Hola Mundo:

```
print("Hello World")
```

Dónde la indicación `print` dentro del lenguaje de Python pide a la consola *imprimir* el ó los elementos dentro del paréntesis, en este caso se pide imprimir la cadenta de caracteres de tipo indicada entre las comillas. Posteriormente al ejecutar en nuestra consola nos devuelve el texto:

```
Hello World
```

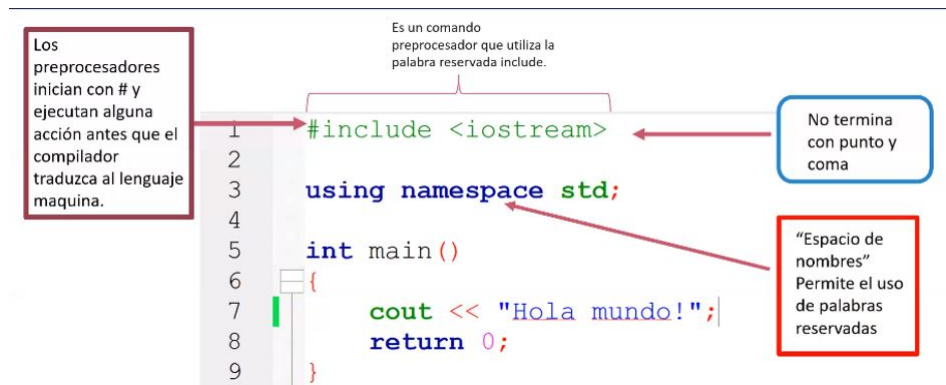


Figura 4: Estructura del programa *Hola Mundo* en C++.

```

1 # Programa 1
2 # Lenguajes de programación
3 # Alumno: Francisco Javier de la Cruz Lugo
4 # 9 de Septiembre de 2020
5 # Descripción: Uso de "print" e "input"
6 print("Hola Mundito Bomnito jsjs!")
7 input("Presiona Enter para finalizar el programa: ")
8

```

Figura 5: Sintáxis del programa *Hola Mundo* en Python en el editor Sublime Text.

La función *input* nos permite interactuar con el usuario que ejecute el código mediante el tecleo de caracteres por medio del teclado.

```

6 #include <iostream> // Esta instrucción es un preprocesador
7
8 using namespace std; // Este es un espacio para nombres
9
10
11 int main() // Indicamos la función principal
12 {
13     cout<<"Hola Mundito Bomnito xD!"; // La instrucción "cout" sirve para...
14     /* Este es un comentario
15     que abarca
16     tres líneas jsjsjs */
17     return 0;
18 }
19

```

Hola Mundito Bomnito xD!
Process returned 0 (0x0) execution time : 1.520 s
Press any key to continue.

Figura 6: Nuestro primer programa en C++ consistió en imprimir el mensaje *Hola Mundo*.


```

6
7 print("Hola Mundito Bomnito jsjs!") # La función "print" nos imprime el objeto especificado
8
9 input("Presiona Enter para finalizar el programa: ") # La función "input" nos permite escribir dentro
10
11 '''Comentario
12 de tres
13 líneas'''
14

```

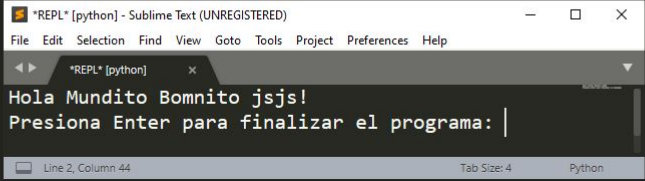


Figura 7: Nuestro primer programa en Python consistió en imprimir el mensaje Hola Mundo.

3.2. Programas Realizados:

- [primer_programa.cpp](#) : Consiste en imprimir el mensaje *Hola Mundo* al ejecutarlo usando la sintaxis del lenguaje C++.
- [Programa.py](#) : Consiste en imprimir el mensaje *Hola Mundo* al ejecutarlo usando la sintaxis del lenguaje Python.

4. Clase 11 de Septiembre

4.1. Tipos de datos en C++

Se define un **tipo de dato** como un conjunto de valores y de operaciones que pueden aplicarse a estos valores.

En C++ se tienen dos tipos de datos: *tipos de datos de clase* que son creados por el propio programador; y los *tipos de datos integrados* que son proporcionados como parte integral del compilador de C++. De estos datos integrados tenemos los *datos numéricos* los cuales se dividen en enteros `int` y los flotantes `float` o números de punto flotante que son números decimales.

Tipo de dato integrado	Operaciones
Números Enteros (<code>int</code>)	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code> , <code>=</code> , <code>==</code> , <code>!=</code> , <code><=</code> , <code>>=</code> , <code>sizeof()</code> y operaciones con bits
Flotantes (<code>float</code>)	<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , <code>%</code> , <code>=</code> , <code>==</code> , <code>!=</code> , <code><=</code> , <code>>=</code> , <code>sizeof()</code>

Cuadro 1: Operaciones con datos integrados.

Dentro de los tipos de datos numéricos enteros `int` tenemos los siguientes: `bool`, `char`, `short int`, `int`, `long int`, `unsigned char`, `unsigned short int`, `unsigned int`, `unsigned long int`.

4.2. El carácter de escape

En C++ existe un carácter con un significado algo especial, se trata de la diagonal inversa que es conocida como **carácter de escape**. Cuando se coloca este carácter directamente frente a un grupo de caracteres indica al compilador que *escape* de la forma en que estos se interpretarían de la forma normal. Una combinación de la diagonal inversa con uno de estos caracteres específicos se le llama **secuencia de escape**.

Secuencia de Escape	Carácter de Escape	Significado
<code>\n</code>	Línea nueva	Se mueve a una línea nueva.
<code>\t</code>	Tabulador horizontal	Se mueve a la siguiente posición del tabulador horizontal.
<code>\v</code>	Tabulador vertical	Se mueve a la siguiente posición del tabulador vertical.
<code>\b</code>	Retroceso	Retrocede un espacio.
<code>\r</code>	Retorno de carro	Mueve el cursor al inicio de la línea actual para sobre escribir.
<code>\f</code>	Alimentación de forma	Expulsa una hoja para iniciar otra.
<code>\a</code>	Alerta	Emite una alerta (por lo general un sonido de campana).
<code>\\</code>	Diagonal inversa	Inserta un carácter de diagonal inversa (ésta se utiliza para colocar un carácter de diagonal inversa real dentro de una cadena).
<code>\?</code>	Signo de interrogación	Inserta un carácter de signo de interrogación.
<code>\'</code>	Comilla sencilla	Inserta un carácter de comilla sencilla (ésta se utiliza para colocar una comilla sencilla interior dentro de un conjunto de comillas sencillas exteriores).
<code>\"</code>	Comilla doble	Inserta un carácter de comilla doble (ésta se utiliza para colocar una comilla doble interior dentro de un conjunto de comillas dobles exteriores).
<code>\nnn</code>	Número octal	El número nnn (n es un dígito) se considera un número octal.
<code>\xhhh</code>	Número hexadecimal	El número hhhh (h es un dígito) se considera un número hexadecimal.
<code>\0</code>	Carácter nulo	Inserta un carácter Null, el cual se define como un valor de 0.

Cuadro 2: Secuencias y caracteres de escape en C++.

4.3. Programas Realizados:

- [programa_2.cpp](#) : Consiste en imprimir una serie de datos personales usando el carácter de escape `\n` usando la sintaxis del lenguaje C++.
- [programa_3.cpp](#) : Consiste en imprimir un texto con un formato usando el carácter de escape `\n` usando la sintaxis del lenguaje C++.
- [programa2.py](#) : Consiste en imprimir una serie de datos personales usando el el carácter de escape `\n` en la sintaxis del lenguaje Python.
- [programa3.py](#) : Consiste en imprimir un texto con formato usando el la sintaxis del lenguaje Python.

5. Clase 18 de Septiembre

5.1. Más tipos de datos en los lenguajes de Programación

En la siguiente tabla encontraremos más tipos de datos manejables en los lenguajes de programación vistos hasta ahora (Python y C++):

Nombre del tipo de dato	Tamaño del almacenamiento (en <i>bytes</i>)	Rango de valores
char	1	256 caracteres
bool	1	Verdadero (el cual es considerado como cualquier valor positivo) y Falso (el cual es 0)
short int	2	-32,768 a +32,768
unsigned short int	2	0 a +65,535
int	4	-2,147,483,648 a +2,147,483,647
insigned int	4	0 a +4,294,967,295
long int	4	-2,147,483,648 a +2,147,483,647
unsigned long int	4	0 a +4,294,967,295

Cuadro 3: Tipos de datos en lenguajes de programación.

5.2. Tipos de datos de punto flotante (float)

Un número de *punto flotante*, el cuál es también un número real, bien puede ser un número cero o cualquier otro número que contenga punto decimal. En C++ se aceptan tres tipos de flotantes: float, double y long double. Lo que difiere a estos tres tipos de datos flotantes es su almacenamiento. En la siguiente tabla se muestran algunas características de estos tipo de datos:

Tipo	Almacenamiento	Rango absoluto de valores (- y +)
float	4 bytes	1.40129846432481707e-45 a 3.40282346638528860e+38
double y long double	8 bytes	4.94065645841246544e-324 a 1.79769313486231570e+308

Cuadro 4: Características de datos de tipo flotante.

Presición: En teoría numérica, este concepto alude por lo general a la exactitud numérica.

5.3. Programas Realizados:

- [programa_4.cpp](#) : Realiza algunas operaciones aritmeticas y las imprime como texto en C++.
- [programa_5.cpp](#) : Realiza un calculo del promedio para dos variables de tipo flotante double en C++.
- [programa_6.cpp](#) : Mejora el código del programa anterior omitiendo algunas líneas de código.
- [programa_7.cpp](#) : Mejora el programa anterior permitiendole al usuario ingresar los valores desde el teclado para ser operados en C++.

- [programa_8.cpp](#) : Utiliza una variable de tipo char.
- [programa_9.cpp](#) : Permite al usuario tipear su nombre para guardarlo en una variable de tipo char.
- [programa_10.cpp](#) : Permite la usuario llenar un formato en el cual ingresar sus datos personales y los guarda en variables y al final imprimendolos en pantalla.

Los siguientes programas realizados en Python cumplen con la función equivalente respectivamente a los realizados en C++ y que se mostraron en la lista anterior:

- [programa4.py](#)
- [programa5.py](#)
- [programa6.py](#)
- [programa7.py](#)
- [programa8.py](#)
- [programa9.py](#)

6. Clase 21 de Septiembre

6.1. Operadores de asignación

El operador de asignación cumple con la sintáxis que se muestra a continuación, dónde el símbolo = indica la *asignación* para la variable especificada:

```
variable = expresión;
```

Para este caso la palabra *expresión* alude a prácticamente cualquier combinación de constantes, variables y llamadas a funciones que puedan evaluarse para posteriormente producir un resultado específico. Como ejemplos tenemos los siguientes:

```
suma = 7 + 3;
diferencia = 7 - 3.25;
producto = 1.05 * 3.71;
Total = suma + cantidad;
```

En programación puede realizarse una *acumulación* de valores para una variable en específico, por ejemplo, de la expresión $96+70+85+60$ se tiene la siguiente secuencia de instrucciones:

En la secuencia de instrucciones anterior se muestra lo que es una instrucción de conteo misma que tiene la siguiente sintaxis y un par de ejemplos:

```
variable = variable + número fijo;
i = i + 1;
n = n + 1;
contador = contador + 1;
```

El siguiente cuadro muestra un par de alternativas para una instrucción de conteo:

Instrucción	Valor en suma
<code>suma = 0;</code>	0
<code>suma = suma + 96;</code>	96
<code>suma = suma + 70;</code>	166
<code>suma = suma + 85;</code>	252
<code>suma = suma + 60;</code>	311

Expresión	Alternativa
<code>i = i + 1;</code>	<code>++ i;</code> : Operador de prefijo para incremento. <code>i ++;</code> : Operador de posfijo para incremento.
<code>i = i - 1;</code>	<code>-- i;</code> : Operador de prefijo para descenso. <code>i --;</code> : Operador de posfijo para descenso.

Cuadro 5: Alternativas para instrucción de conteo.

6.2. Programas realizados

- [programa_11.cpp](#) : Realiza el cálculo de la pendiente de una recta dados dos puntos en C++.
- [programa_12.cpp](#) : Realiza una reasignación del valor de una variable con una instrucción de conteo.
- [programa_13.cpp](#) : Reasigna el valor para una variable sumando diferentes valores por cada vez.
- [Ejercicio_1.cpp](#) : Calcula el n-ésimo término en una sucesión aritmética, usando una instrucción de asignación. (sin usar la instrucción `cin`).
- [Ejercicio_2.cpp](#) : Calcula el n-ésimo término en una sucesión aritmética, usando una instrucción de asignación. (usando la instrucción `cin`).

En los siguientes ejercicios se corrigen algunos errores y observaciones en algunos códigos de programas:

- [Ejercicio_3_a.cpp](#)
- [Ejercicio_3_b.cpp](#)
- [Ejercicio_3_c.cpp](#)

7. Clase 23 de Septiembre

7.1. Dar formato a números para salida del programa. Manipuladores

Estos *manipuladores* de datos nos ayudan a agregar un detalle meramente estético a los programas al momento de ejecutarlos. Por ejemplo al desplegar un resultado monetario como 1.897 no cumpliría con las condiciones aceptadas para los informes. El despliegue

debería ser \$1.90 o bien \$1.89, dependiendo si se usa redondeo o truncamiento. El fomrato de los números desplegados por `cout` puede controlarse mediante manipuladores de ancho de campo incluidos en cada flujo de salida.

Manipulador	Acción
setw(<i>n</i>)	Establece el ancho de campo en <i>n</i> .
setprecision(<i>n</i>)	Establece la precisión del punto flotante en <i>n</i> lugares. Se designa el manipulador <code>fixed</code> , <i>n</i> especifica el número total de dígitos desplegado después del punto decimal; de otra manera, <i>n</i> especifica el número total de dígitos significativos desplegados (números enteros más dígitos fraccionarios).
setfill('x')	Establece el carácter de relleno a la izquierda o omisión en <i>x</i> (El carácter de relleno principal por omisión es un espacio, el cual es la salida por defecto). Rellenar el frente de un campo de salida siempre que el ancho del campo sea mayor que el valor que está desplegado.
setiosflags(<i>flags</i>)	Establece el formato de los indicadores.
scientific	Establece la salida para desplegar números reales en notación científica.
showbase	Despliega la base usada para los números. Se despliega un 0 a la izquierda para los números octales y un 0x a la izquierda para los números hexadecimales.
showpoint	Siempre despliega seis dígitos en total (combinación de partes enteras y fraccionarias). Rellena con ceros a la derecha si es necesario. Para valores enteros mayores, revierte a notación científica.
showpos	Despliega todos los números positivos con un signo de + a la izquierda.
boolalpha	Despliega valores booleanos como verdadero y falso, en lugar de como 1 y 0.
dec	Establece la salida para un despliegue decimal por omisión.
endl	Da salida a un carácter de línea nueva y despliega todos los caracteres en el búfer.
fixed	Siempre muestra un punto decimal y usa seis dígitos por omisión después del punto decimal. Rellena con ceros a la derecha si es necesario.
flush	Despliega todos los caracteres del búfer.
left	Justifica a la izquierda todos los números.
hex	Establece la salida para un despliegue hexadecimal.
oct	Establece la salida para un despliegue octal.
uppercase	Despliega dígitos hexadecimales y el exponente en notación científica en mayúsculas.
right	Justifica a la derecha todos los números (éste es el valor por omisión).
noboolalpha	Despliega valores booleanos como 1 y 0, en lugar de verdadero y falso.
noshowbase	No despliega números octales con un 0 a la izquierda y los números hexadecimales con un 0x a la izquierda.
noshowpoint	No usa un punto decimal para números reales sin partes fraccionarias; no despliega ceros a la derecha en la parte fraccionaria de un número; despliega un máximo de sólo seis dígitos decimales.
noshowpos	No despliega signos de + a la izquierda (éste es el valor por omisión).
nouppercase	Despliega dígitos hexadecimales y el exponente en notación científica en minúsculas.

Cuadro 6: Manipuladores de datos en C++.

7.2. Procesador iomanip

Cuando se usa un manipulador que requiere un argumento debe incluirse el archivo de encabezado `iomanip` como parte del programa. Esto se logra con el comando preprocesador `#include<iomanip>`. Dar formato completo a números de punto flotante requiere el uso de tres manipuladores de ancho de campo:

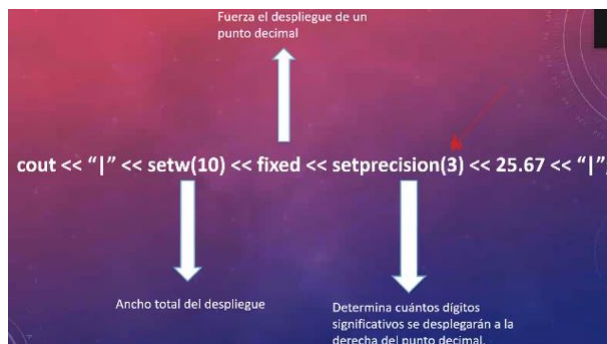


Figura 8: Línea que da formato a un número de salida.

Manipuladores	Número	Despliegue	Comentarios
<code>setw(2)</code>	3	3	El número cabe en el campo.
<code>setw(2)</code>	43	43	El número cabe en al campo.
<code>nsetw(2)</code>	143	143	El ancho de campo se ignora.
<code>setw(2)</code>	2.3	2.3	El ancho de campo se ignora.
<code>setw(5)</code> <code>fixed</code> <code>setprecision(2)</code>	2.366	2.37	Ancho de campo de de cinco con dos dígitos decimales.
<code>setw(5)</code> <code>fixed</code> <code>setprecision(2)</code>	42.3	42.30	El número cabe en el ancho de campo con precisión especificada.
<code>setw(5)</code> <code>setprecision(2)</code>	142.364	1.4e+002	El ancho de campo de se ignora y se usa notación científica con el manipulador <code>setprecision</code> especificando el número total de dígitos significativos (enteros más fraccionarios).

Cuadro 7: Ejemplos de manipulación de formato para datos de salida.

7.3. Programas realizados

- [programa_17.cpp](#) : Realiza un a opeación sencilla.
- [programa_18.cpp](#) : Mejora el programa anterior eliminando algunos `cout`.
- [programa_19-v1.0.cpp](#) : Realiza la misma operación anterior agregando un formato sencillo.
- [programa_19-v2.0.cpp](#) : Mejora el programa anterior agregando más formato a los datos de salida.
- [programa_20.cpp](#) : Mejora el formato de algunas operaciones sencillas.

Indicador	Significado
<code>ios::fixed</code>	Siempre se muestra un punto decimal con seis dígitos después del punto decimal. Rellena con 0 a la derecha si es necesario. Este indicador tiene precedencia sobre <code>ios::scientific</code> y <code>ios::showpoint</code> . Se establece con el indicador <code>ios::showpoint</code> .
<code>ios::scientific</code>	Usa despliegue exponencial en la salida.
<code>ios::showpoint</code>	Siempre despliega un punto decimal y seis dígitos significativos en total (combinación de partes enteras y fraccionarias). Rellena con 0 a la derecha después del punto decimal si es necesario. Para valores enteros más grandes, revierte a notación científica a menos que se establezca el indicador <code>ios::fixed</code> .
<code>ios::showpos</code>	Despliega un + a la izquierda cuando el número es positivo.
<code>ios::left</code>	Justifica a la izquierda la salida.
<code>ios::right</code>	Justifica a la derecha la salida.

Cuadro 8: Indicadores para formato de datos de salida.

- [programa_21-v1.cpp](#) : Calcula la resistencia combinada de algunos resistores.
- [programa_21.-v2cpp](#) : Mejora el código del programa anterior permitiendo al usuario ingresar los valores para los resistores.
- [programa_22.cpp](#) : Calcula el momento de flexión máxima de una varilla.

8. Clase 25 de Septiembre

8.1. Programas realizados

- [programa_24.cpp](#) : Calcula la raíz cuadrada de un número usando la expresión `sqrt()`.
- [programa_25.cpp](#) : Calcula el seno, coseno y la tangente de un ángulo en radianes usando las expresiones `sin()`; `cos()`; `tan()`; respectivamente.
- [programa_26.cpp](#) : Calcula la potencia y el exponencial de algunos números usando las expresiones `pow(x, y)`; `exp()`; respectivamente.
- [programa_27.cpp](#) : Calcula el logaritmo natural y el logaritmo base 10 de algunos números usando las expresiones `log()`; `log10()`; respectivamente.

9. Clase 28 de Septiembre

9.1. Las instrucciones `if - else`

La instrucción `if - else` en C++ se usa para poner en práctica una estructura de decisión en su forma más simple. La de elegir entre dos alternativas. La **sintaxis** de pseudocódigo que más de esta instrucción es:

```
if (Condición)
    instrucción ejecutada si la condición es verdadera
else (Condición)
    instrucción ejecutada si la condición es falsa
```

La instrucción `if`, la condición es evaluada para determinar su valor numérico, el cuál es interpretado entonces como verdadero o falso. Si la condición produce cualquier valor numérico positivo o negativo diferente de cero, la condición es considerada como una condición *verdadera* y se ejecuta la instrucción que sigue a `if`. Si la condición produce un valor

numérico de cero, la condición es considerada como *falsa* y se ejecuta la instrucción que sigue a *else*. La parte *else* de la instrucción es opcional y puede omitirse.

La condición usada en una instrucción *if* puede ser cualquier expresión válida de C++ (incluyendo una expresión de asignación). Las expresiones más usadas por lo común, sin embargo, se llaman *expresiones relacionales*. Una *expresión relacional simple* consiste en un operador relacional que compara dos operandos.

$$\text{raíz} \leq 5$$

Operador relacional	Significado	Ejemplo
<	menor que	edad < 30
>	mayor que	altura > 6.2
<=	menor que o igual a	gravable <= 20000
>=	mayor que o igual a	temp >= 98.6
==	igual a	calificación == 100
!=	no es igual a	número != 250

Figura 9: Operadores relacionales y su significado.

Expresión	Valor	Interpretación
'A' > 'C'	0	falso
'D' <= 'Z'	1	verdadero
'E' == 'F'	0	falso
'g' >= 'm'	0	falso
'b' != 'c'	1	verdadero
'a' == 'A'	0	falso
'B' < 'a'	1	verdadero
'b' > 'Z'	1	verdadero

Figura 10: Valores de char y su interpretación.

Expresión	Valor	Interpretación	Comentario
"Hola" > "Adios"	1	verdadero	La primera 'H' en Hola es mayor que la primera 'A' en Adiós
"SOLANO" > "JIMENES"	1	verdadero	La primera 'S' en SOLANO es mayor que la primera 'J' en JIMENEZ
"123" > "1227"	1	verdadero	El tercer carácter, el '3', en 123 es mayor que el tercer carácter, el '2', en 1227.
"Bejuco" > "Beata"	1	verdadero	El tercer carácter, la 'j', en Bejuco es mayor que el tercer carácter 'a' en Beata.
"Hombre" == "Mujer"	0	falso	La primera 'H' en Hombre no es igual a la primera 'M' en Mujer
"planta" < "planeta"	0	falso	La 't' en planta es mayor que la 'n' en planeta.

Figura 11: Clases String.

Expresión	Valor	Interpretación
'A' > 'C'	0	falso
'D' <= 'Z'	1	verdadero
'E' == 'F'	0	falso
'g' >= 'm'	0	falso
'b' != 'c'	1	verdadero
'a' == 'A'	0	falso
'B' < 'a'	1	verdadero
'b' > 'z'	1	verdadero

Figura 12: Valores de char y su interpretación.

9.2. Operadores lógicos

Operador	Asociatividad
! unitario - ++ --	derecha a izquierda
* / %	izquierda a derecha
+ -	izquierda a derecha
< <= > >=	izquierda a derecha
== !=	izquierda a derecha
&&	izquierda a derecha
	izquierda a derecha
= += -= *= /=	derecha a izquierda

Figura 13: Operadores lógicos y su indicación.

9.3. Programas realizados

- [programa_28.cpp](#) : Calcula la distancia entre dos puntos dadas sus coordenadas.
- [programa_29.cpp](#) : Calcula la altura de una escalera inclinada conociendo el ángulo.
- [programa_30.cpp](#) : Calcula la altura maxima que alcanza una pelota lanzada a velocidad en millas/hora.
- [programa_31.cpp](#) : Aproxima la función seno mediante la serie de Taylor.
- [programa_32.cpp](#) : Operadores relacionales.
- [programa_33.cpp](#) : Realiza la comparación de dos números usando `if - else`.
- [programa_34.cpp](#) : Programa que devuelve el valor para una onda.
- [programa_35.cpp](#) : Programa que devuelve la presión de una onda.
- [programa_36.cpp](#) : Programa que pide un código de acceso.
- [programa_37.cpp](#) : Programa que pide un código para uso de un generador.
- [programa_38.cpp](#) : Programa que pide un código para uso de un generador.

Expresión	Expresión equivalente	Valor	Interpretación
$i + 2 == k - 1$	$(i + 2) == (k - 1)$	0	falso
$3 * i - j < 22$	$(3 * i) - j < 22$	1	verdadero
$i + 2 * j > k$	$(i + (2 * j)) > k$	1	verdadero
$k + 3 <= -j + 3 * i$	$(k + 3) <= ((-j) + (3*i))$	0	falso
$'a' + 1 == 'b'$	$('a' + 1) == 'b'$	1	verdadero
$key - 1 > 'p'$	$(key - 1) > 'p'$	0	falso
$key + 1 == 'n'$	$(key + 1) == 'n'$	1	verdadero
$25 >= x + 1.0$	$25 >= (x + 1.0)$	1	ver

Figura 14: Expresiones de asignación.

10. Clase 30 de Septiembre

10.1. Instrucciones `if` anidadas

La inclusión de una o más instrucciones `if` dentro de una instrucción `if` se les conoce como instrucciones `if` anidadas, y tiene una sintaxis como la que sigue:

```

if (condición_1)
{
    if (condición_2)
    {
        instrucción_1;
        instrucción_2;
    }
    else
    {
        instrucción_1;
        instrucción_2;
    }
}
else
{
    if (condición_3)
    {
        instrucción_1;
        instrucción_2;
    }
    else
        instrucción_1;
}

```

En una sitáxis más simple se tiene la siguiente:

```

if (expresión_1)
    instrucción_1;
else if (expresión_2)
    instrucción_2;
else if (expresión_3)
    instrucción_3;
:
else if (expresión_n)
    instrucción_n;

```

10.2. La instrucción switch

Proporciona una alternativa a la instrucción `if - else` para casos que comparan el valor de una expresión de número entero con un valor específico, cumple con la sintáxis siguiente:

```

switch (expresión)
{
    case valor_1:
        instrucción_1;
        instrucción_2;
:
        break;
    case valor_2:
        instrucción_1;
        instrucción_2;
:
        break;

```

```

:
case valor_n:
    instrucción_1;
    instrucción_2;
:
    break;
default:
    instrucción_1;
    instrucción_2;
:
}

```

10.3. Programas realizados

- [programa_39.cpp](#) : Programa que utiliza la instrucción `switch` para realizar operaciones con un par de números.
- [programa_40.cpp](#) : Programa que resuelve una ecuación cuadrática dados sus coeficientes.
- [programa_41.cpp](#) : Programa que clasifica ángulos de acuerdo al valor del ángulo introducido.
- [programa_42.cpp](#) : Programa que convierte de grados Celsius a grados Fahrenheit y viceversa.
- [programa_43.cpp](#) : Programa que permite seleccionar una fecha en un calendario.
- [programa_44.cpp](#) : Programa que permite seleccionar una fecha en un calendario considerando un año bisiesto.

11. Clase 2 de Octubre

11.1. Ciclos

Un programa puede ser considerablemente *complejo* dependiendo de cuánto debe realizar una y otra vez el mismo tipo de operación. Al construir una sección de código meramente *repetitiva* requiere de los siguientes elementos:

- Instrucción de repetición (mismas que requieren de una condición a evaluar):
 - `while`
 - `for`
 - `do while`
- Condición a evaluarse
- Instrucción que establece la condición al inicio (debe colocarse antes de que la condición sea evaluada por primera vez para asegurar la ejecución correcta del ciclo.)
- Debe haber una instrucción dentro de la sección de código repetitiva que permita que la condición se vuelva falsa para asegurar que en algún punto se detengan las repeticiones.

11.2. Ciclos while

En C++ un ciclo `while` se construye usando una instrucción `while`. La sintáxis de esta instrucción es la siguiente:

```
while (expresión/condición)
    instrucción;
```

11.3. Programas realizados

- [programa_45.cpp](#) : Programa que muestra el funcionamiento del ciclo `while`.
- [programa_46.cpp](#) : Modifica el código del programa anterior eliminando líneas.
- [programa_47.cpp](#) : Este programa calcula el cuadrado, cubo y raíz cuadrada de los números del 1 al 15 y los imprime en pantalla.
- [programa_48.cpp](#) : Modifica el código del programa anterior permitiendo al usuario ingresar el rango de números a evaluar.
- [programa_49.cpp](#) : Programa que calcula la distancia de un auto por cada hora transcurrida.
- [programa_50v3.cpp](#) : Programa que aproxima el valor de $\sin(x)$ para un ángulo dado con un error mínimo de 0.000001 usando `while`.

12. Clase 7 de Octubre

12.1. Ciclos for

En C++ un ciclo `for` se construye usando una instrucción `for`. La sintáxis de esta instrucción es la siguiente:

```
for (lista de inicialización; expresión; lista de alteración)
    instrucción;
```

12.2. Programas realizados

- [programa_51.cpp](#) : Programa que calcula el factorial de un numero usando `while`.
- [programa_51v2.cpp](#) : Programa que calcula el factorial de un numero usando `for`.
- [programa_52.cpp](#) : Programa que muestra el funcionamiento del ciclo `for`.
- [programa_53.cpp](#) : Programa que aproxima el valor de $\sin(x)$ para un angulo dado con un error minimo de 0.000001 usando `for`.

13. Clase 14 de Octubre

13.1. Ciclos `do - while`

Un ciclo `do - while` de prueba posterior se crea usando una instrucción `do`. Esta instrucción permite hacer otras instrucciones antes que sea evaluada una instrucción final del ciclo. Cumple con la sintaxis siguiente:

```
do
    instrucción;
while (expresión);
```

13.2. Programas realizados

- [programa_54.cpp](#) : Programa que permite al usuario ingresar números indefinidamente hasta que se decida no continuar usando el ciclo `do - while`.
- [programa_54v2.cpp](#) : Mejora el código del programa anterior limpiando la pantalla cada vez que se ingresa un nuevo número.
- [programa_55.cpp](#) : Programa que calcula suma, resta, producto, cociente de dos números permitiendo al usuario elegir entre las operaciones que desea realizar o finalizar el programa.
- [programa_57.cpp](#) Programa que aproxima la raíz de un polinomio dado mediante el metodo de bisección.

14. Clase 16 de Octubre

14.1. Programas realizados

- [programa_56v1.cpp](#) : Programa que aproxima raíces de polinomios usando el metodo de Newton-Rapson dada una ecuación.

15. Clase 19 de Octubre

15.1. Programas realizados

- [programa_58.cpp](#): Programa que calcula la raiz de una ecuacion por METODO DE ITERACION SIMPLE DE PUNTO FIJO.

16. Clase 28 de Octubre

16.1. Modularidad con el uso de funciones.

Declaración de funciones y parámetros.

Al momento de crear funciones en el lenguaje de C++, se debe prestar atención a la propia función y en la forma en que interactúa con otras funciones, tal como la función `main()`. Esto incluye transmitir datos en forma correcta 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 función se invoca o utiliza, dando el nombre de la función y transmitiéndole 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 manipu- lados éstos para producir un resultado útil.

```
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); // Aquí se llama a la función
20     return 0;
21 }
22
```

Función que espera recibir dos números y No devolver ningún valor (void) a main

Función que llama

Función llamada

Figura 15: Ejemplo de función modular.

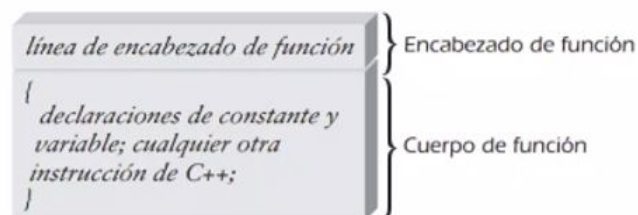


Figura 16: Estructura de una función modular.

```

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); // Aquí se llama a la función
20
21     return 0;
22 }

```

Figura 17: Parametros y argumentos de una función modular.

16.2. 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.