

CS1102 – PROGRAMACIÓN ORIENTADA A OBJETOS 1

CICLO 2019-1

Unidad 1:

Elementos de Programación y Estructuras de Control

<http://bit.ly/2HRBWgq>

Profesores:

Ernesto Cuadros- Vargas, PhD.

ecuadros@utec.edu.pe

María Hilda Bermejo, M. Sc.

mbermejo@utec.edu.pe

Telegram:

1. **Configurar tu cuenta**
2. <http://bit.ly/2TJnwBq>

Logro del curso:

Al finalizar el curso, el estudiante crea programas orientados a objetos, diseñando sus propias clases, objetos y utilizando los mecanismos que hacen posible la implementación de conceptos tales como: abstracción, encapsulamiento, relaciones entre clases y polimorfismo.

Se dicta:



**2 horas
teoría**

+



**2 horas
Laboratorio**

+

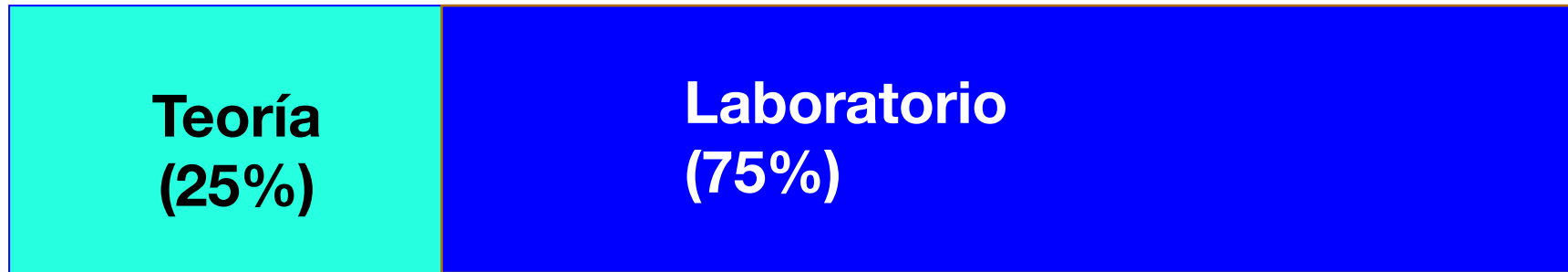


**2 horas
Laboratorio**

Lista de temas:

- Elementos de programación y Estructuras de control
- Funciones
- Arrays unidimensionales y vectores
- Matrices
- Punteros
- Asignación dinámica de memoria
- Archivos
- POO: clases y objetos
- Constructores, destructores
- Relaciones entre clases
- Polimorfismo

Sistema de Evaluación:



Sistema de evaluación:

$$\begin{aligned} \text{NF} = & 0.25 * E1 + 0.05 * C1 + 0.05 * C2 + \\ & 0.10 * PC1 + 0.10 * PC2 + 0.10 * PC3 + 0.10 * PC4 + \\ & 0.10 * P1 + 0.15 * P2 \end{aligned}$$

Donde:

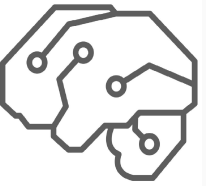
E1: Examen (1) corresponde a las evaluaciones de las clases desarrolladas en el Auditorio.

C : Evaluación Continua: C1 (semanas 1 - 7), C2 (Semanas 8 - 15)

PC : Prácticas Calificada (4)

P : Proyecto (2)

Digital Transformation & The New Technology Revolution



[Marcos Ramírez](#) is passionate about technology, innovation, and Data Science. He holds a Master in Management from IE Business School and studied Industrial Engineering at Tecnológico de Monterrey. He has also been an alumnus from the Master in Business Analytics and Big Data at IE School of Human Sciences and Technology. He has completed the Internet of Things Bootcamp at MIT last year. Marcos has participated at several conferences on Data Science and Cybersecurity around the world such as: Disney Analytics, RSA San Francisco, Black Hat, CyberTech, Big Data Summit Boston, GITEX, among others.

FECHA
27 marzo

HORA
6:30 p.m.

ESPACIO
Auditorio

LUGAR
UTEC

Ingreso libre previa inscripción: <https://goo.gl/v44tqh>

Computer Science



Microsoft



Linux



airbnb

Google

facebook

amazon



ORACLE®

You Tube



Viber
Free Phone Calls



U B E R



Dropbox

YAHOO!



waze



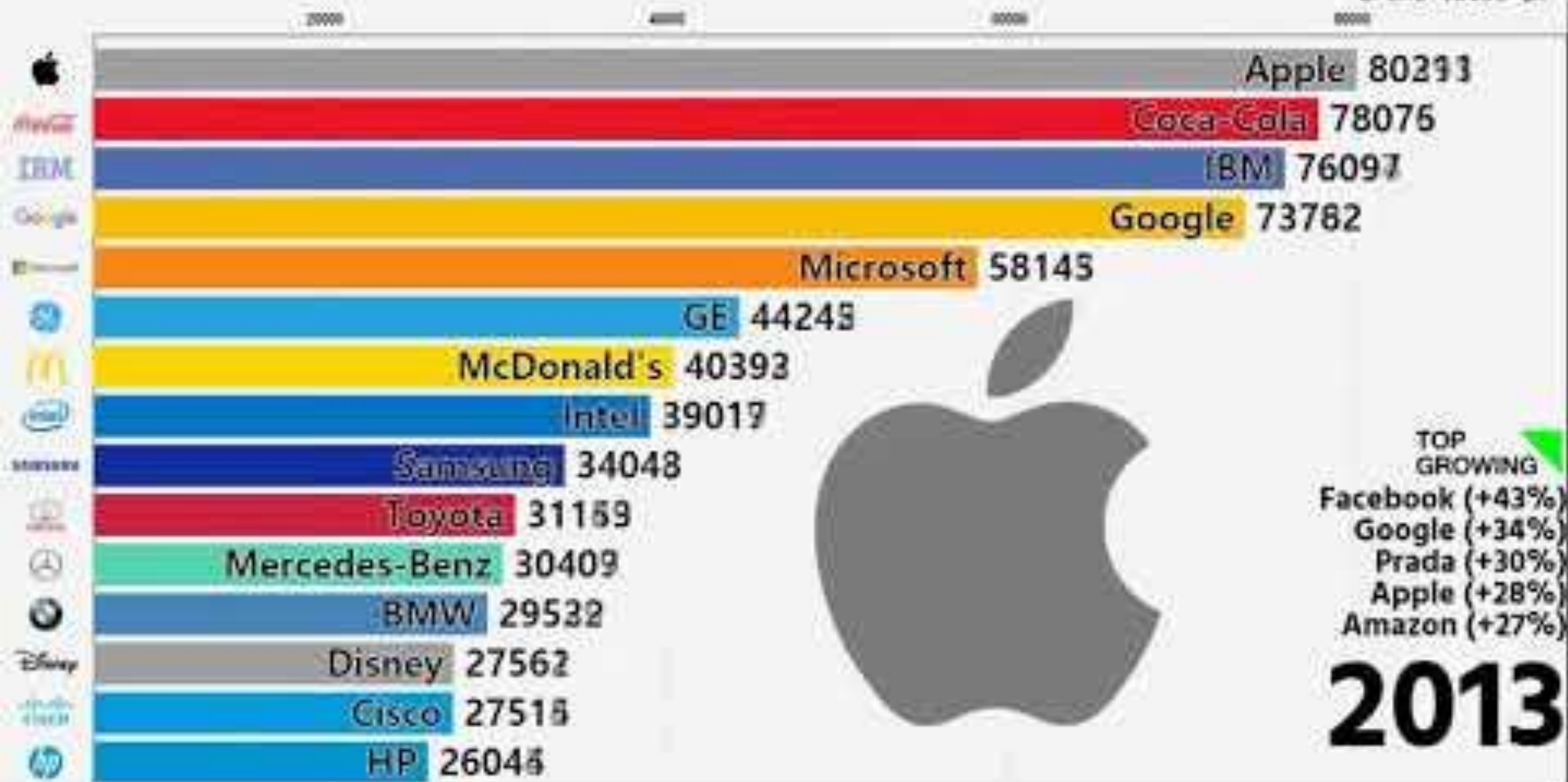
WhatsApp



Spotify

Top 15 Best Global Brands Ranking

Brand Value \$B



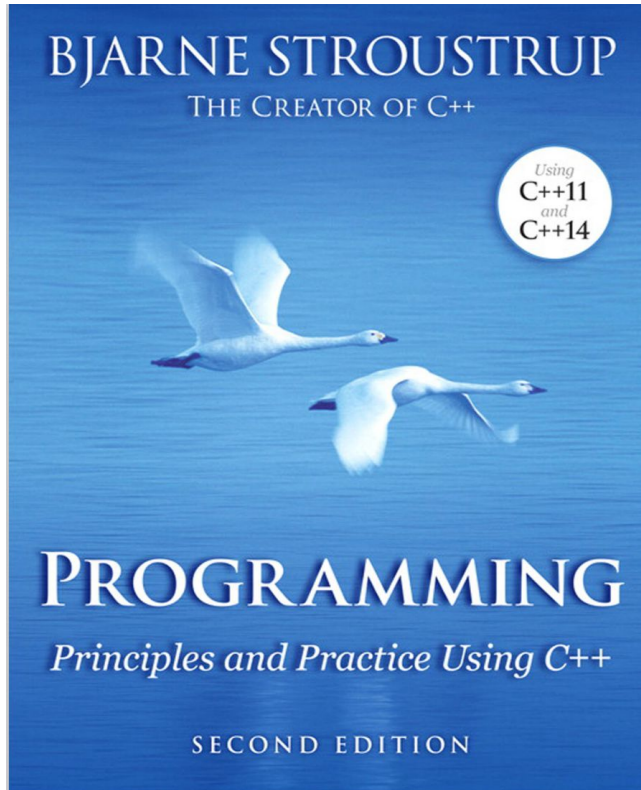
Logro de la sesión:

Al finalizar la sesión, los alumnos conocen:

- 1. Las partes de un programa en C++,**
- 2. Tipos de datos, variables y constantes.**
- 3. Instrucciones de I/O**
- 4. Estructuras de control selectivas y repetitivas**

El Lenguaje C++

Según (Bjarne Stroustrup, 2014), es un “Lenguaje de propósito general parcialmente centrado en el desarrollo de programación de sistemas, soporta abstracción de datos, y los siguientes paradigmas de programación: **procedural, modular, Orientada a Objetos, genérica y funcional**”.



Bjarne Stroustrup



I'm the designer and original implementer of the C++ programming language. I have used the language, and many other programming languages, for a wide variety of programming tasks over the last 40 years or so. I just love elegant and efficient code used in challenging applications, such as robot control, graphics, games, text analysis, and networking. I have taught design, programming, and C++ to people of essentially all abilities and interests. I'm a founding member of the ISO standards committee for C++ where I serve as the chair of the working group for language evolution.

This is my first introductory book. My other books, such as *The C++ Programming Language* and *The Design and Evolution of C++*, were written for experienced programmers.

Entidades:

El **ISO C++ estándar** define dos tipos de entidades:

Core language features: como tipos de datos Ej: char, int, y bucles Ej: for, while

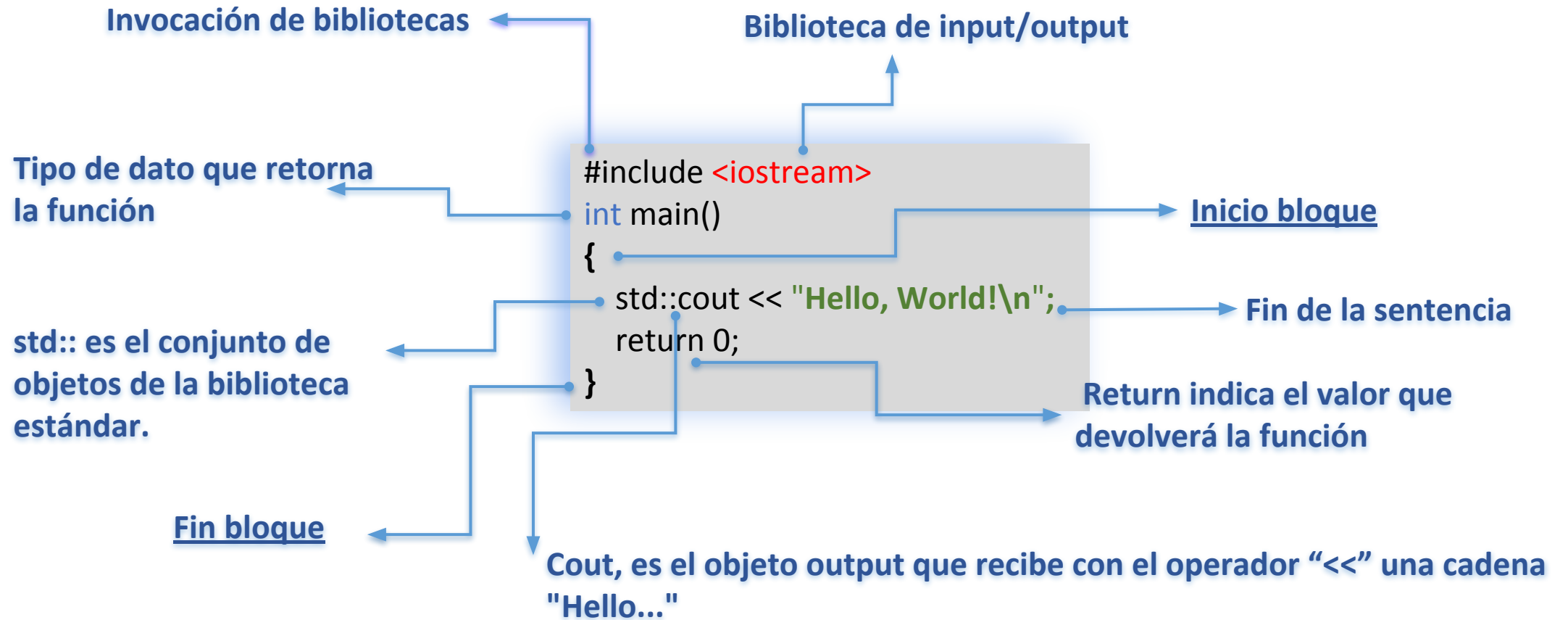
Standard-library components, como contenedores (Ej: vector, map) y I/O operations Ej: << y getline()

La **Standard-library components,** es proveída por cada implementación de C++, y puede ser implementada en C++ en sí misma, por lo que el C++ es suficientemente expresivo y eficiente para los sistemas de programación más demandantes.

El C++ es un lenguaje *tipado*, esto significa que cada tipo de entidad Ej: objeto, valor, nombre y expresión, debe ser conocida por el compilador en el punto que se use.

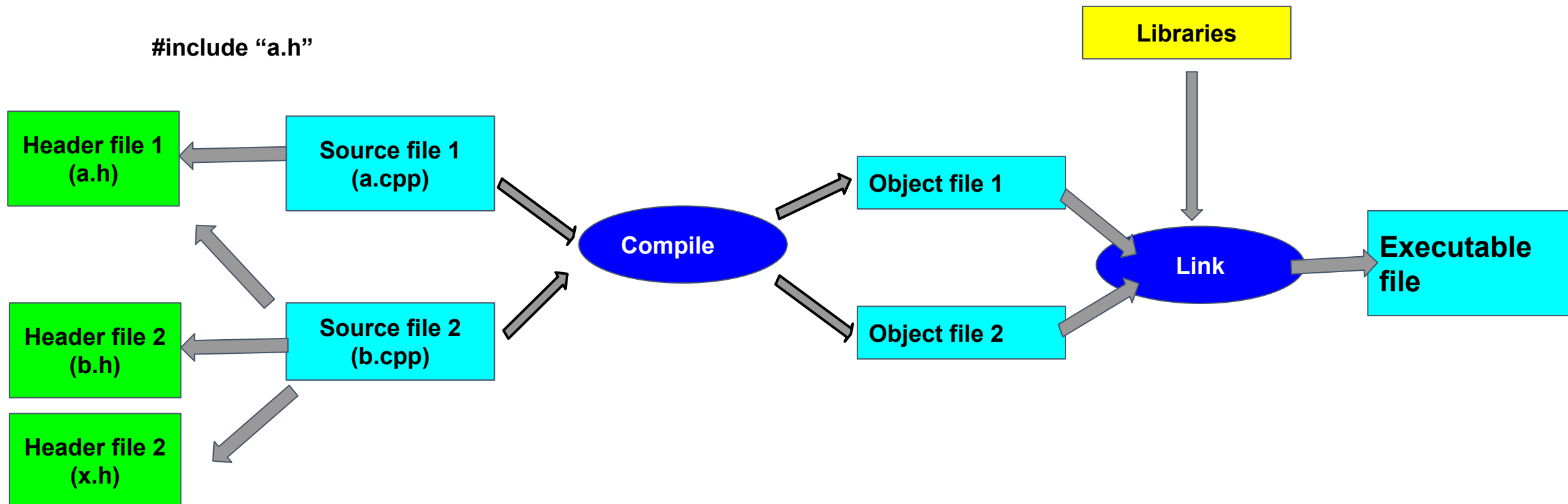
El tipo de un objeto determina el conjunto de operaciones aplicables al el. (Stroustrup,2018)

Estructura de un programa en C++:



Programas:

C++ es un Lenguaje compilado. Para que un programa se ejecute el código fuente tiene que ser procesado por un “Compilador”, produciendo archivos objetos, los cuales son enlazados por un “Linker” produciendo el archivo ejecutable.



Estructura de un programa:

```
#include <iostream> // #include, importa las declaraciones del I/O de la librería stream

int main()
{
    std::cout << "Hola Mundo!\n";
    return 0; // retorna el valor "al Sistema Operativo".
              // No todos los sistemas operativos utilizan este valor, pero el Linux/Unix si lo hace.
}
```

Todo programa, tiene al menos la función **main()** y si tuviese más, la ejecución del programa siempre empieza por la función main().

Usando namespaces:

```
#include <iostream> // #include, importa las declaraciones del I/O de la librería stream
```

```
using namespace std; // hace visible los nombres del std sin utilizar std::
```

```
int main()  
{  
    cout << "Hola Mundo!\n";  
    cout << "CS1102 - Programacion Orientada a Objetos!! \n";  
    cout << "en C S - U T E C \n";  
    return 0;  
}
```

Los namespaces, es un mecanismo que permite agrupar algunas declaraciones en conjunto y evitar que sus nombres entren en conflicto con otros nombres similares. (Stroustrup,2018)

Estructura de un programa, si hay mas de una función:

Forma 1

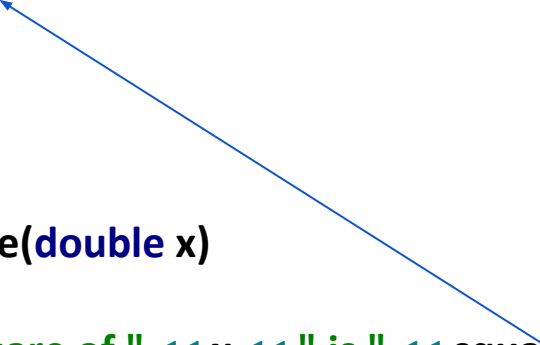
```
#include <iostream>

using namespace std;

double square(double x)
{
    return x*x;
}

void print_square(double x)
{
    cout << "the square of " << x << " is " << square(x) << "\n";
}

int main()
{
    print_square(1.234);
    //-- se imprime : the square of 1.234 is 1.52276
    return 0;
}
```




Forma 2

```
#include <iostream>
using namespace std;
//--- Se declaran los encabezados de las funciones
double square(double x);
void print_square(double x);

int main()
{
    print_square(1.234);
    //-- se imprime : the square of 1.234 is 1.52276
    return 0;
}

double square(double x)
{
    return x*x;
}

void print_square(double x)
{
    cout << "the square of " << x << " is " << square(x) <<
    "\n";
}
```



Tipos y variables:

Cada nombre y cada expresión tiene un tipo que determina las operaciones que se pueden realizar con ellos. Por ejemplo;

int inch; //-- especifica qué **inch** es de tipo **int**, es decir **inch** es una variable de tipo **int**

Una declaración es una sentencia que introduce una entidad en el programa, esto significa que:

El **type** define los posibles valores y el conjunto de operaciones para el objeto

Un **Object** es una parte de memoria que almacena un valor para algún **type**

Un **value** is un conjunto de bits interpretados de acuerdo al tipo de dato.

Una **variable** es un nombre de un objeto

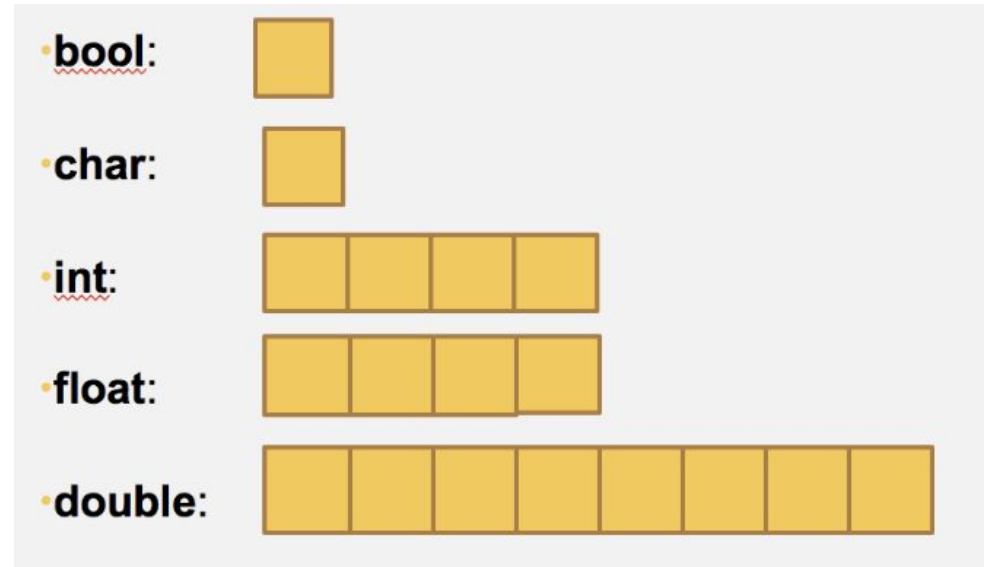
Tipos de datos:

El C++ ofrece varios tipos de datos, los fundamentales son:

bool // booleano, los valores posibles son *true* y *false*
char // character, Ej: 'a', 'z', '9'
int // integer, Ej: -273, 42, 1066
double // double-precision floating-point Ej: -273.15, 3.14, 6.625e-34
unsigned // non-negative integer Ej: 0, 1, 999

Cada tipo se corresponde directamente con las facilidades de hardware y tiene un tamaño que determina el rango de valores que puede almacenar.

Tipos de datos:



Una variable de tipo *char* almacena un carácter y en la máquina típicamente utiliza 8 bits o un byte. El tamaño de los otros tipos son múltiplos del tamaño de un char. El tamaño de un tipo de dato lo determina la implementación y puede variar en diferentes máquinas. Si se quiere conocer el tamaño se puede utilizar el operador *sizeof*.

Por ejemplo:

sizeof(char) es 1

sizeof(int) es 4

Tipos de datos:

Los números pueden ser enteros o de punto flotante.

Los enteros, por default estan expresados en base 10 Ej: 42

El prefijo **0b** indica un número en base 2 Ej: 0b10101010

El prefijo **0x** indica un número en base 16 Ej: 0xBAD1234

El prefijo **0** indica un número en base 8, Ej: 0334

Los de punto flotante pueden ser:

float de simple precisión

double de doble precisión

long double de precisión extendida

Para mayor referencia de los tipos de datos puedes consultar este link:

<https://en.cppreference.com/w/cpp/language/types>

Operadores aritméticos:

$x+y$ // suma
 $+x$ // suma unaria
 $x-y$ // diferencia
 $-x$ // diferencia unaria
 $x*y$ // multiplicacion
 x/y // division
 $x\%y$ // módulo solo para números enteros

Operadores de comparación

$x==y$ // igualdad
 $x!=y$ // diferente
 $x<y$ // menor
 $x>y$ // mayor
 $x\leq y$ // menor igual
 $x\geq y$ // mayor igual

Operadores lógicos:

```
if( a>10 && a<50 ) // and  
if( x=='A' || x=='B') // or  
  
bool esta = false;  
if ( !x) // not (negación)
```

El resultado de evaluar expresiones con estos operadores es **true** o **false**

Operadores lógicos para manejo de bits

```
int a =5, x=4, y=5;  
a = x & y; // and para manejo de bits  
a = x | y; // or para manejo de bits  
a = x ^ y; // or exclusivo para manejo de bits  
a = ~x; // complemento
```

Conversiones:

Cuando se utiliza operaciones aritméticas, el C++ realiza conversiones profundas entre tipos de datos básicos por lo que se pueden utilizar libremente, por ejemplo:

```
#include <iostream>
using namespace std;

int main()
{
    double d=2.2;
    int i = 7;

    d = d+i;  //--- imprime 9.2
    cout << d << "\n";
    i = d*i;  //--- imprime 64 y no 64.4
    cout << i << "\n";
    return 0;
}
```

Las conversiones que se usan se llaman “*the usual arithmetic conversions*”, y las expresiones son realizadas utilizando operaciones de alta precisión. Por ejemplo la suma de un *double* con un *int* es calculada utilizando *aritmética de punto flotante de doble precisión*.

Mas Operadores:

En adición a los operadores aritméticos y lógicos, C++ ofrece operadores específicos para modificar el tipo de una variable.

`x+=y // x = x+y`

`++x // incrementa x = x+1`

`x-=y // x = x-y`

`--x // x = x-1`

`x*=y // x = x*y`

`x/=y // x = x/y`

`x%=y // x = x%y solo con enteros`

Inicialización:

Antes de usar un objeto, este debe ser inicializado.

```
double d1 =2.3;  
double d2 {2.3};  
double d3 = {2.3};
```

```
vector<int> v {1,2,3,4,5,6};
```

Cuando se define una variable, se puede deducir el tipo por su inicialización:

```
auto b = true; // bool  
auto ch = 'x'; // char  
auto i = 123; // int  
auto d = 1.2; // double  
auto z = sqrt(y); // z tiene el tipo de dato que retorna  
sqrt(y)  
auto bb {true}; // bool
```

El uso de auto es especialmente importante en programación genérica.

Constantes:

const significa *“I promise not to change the value”*

```
const int dmv=17;
```

constexpr significa *“to be a evaluated at compile time”*.

```
const int dmv = 17;  
constexpr double max1= 1.4*square(dmv)
```

Ejemplo 1:

El siguiente programa permite hallar el Área y el Volumen de una esfera.

Se han utilizado estas fórmulas:

$$\text{Area} = 4 * 3.1415 * r^2$$

$$\text{Volumen} = \frac{4}{3} * \text{PI} * r^3$$

Solucion 1 - main.cpp

```
#include <iostream>
#include <cmath> //--- se incluye esta librería para poder utilizar la función pow
using namespace std;

const double PI=3.1415;

int main()
{ double radio=0, area=0, volumen=0;

  cout << "Radio : ";
  cin >> radio;
  area = 4 * PI * radio * radio;
  volumen = 4.0/3 * PI * pow(radio,3);
  cout << "\n";
  cout << "El area es  : " << area << "\n";
  cout << "El volumen es : " << volumen << "\n";
  return 0;
}
```



Solucion 2 - main.cpp

```
#include <iostream>
#include <cmath> //--- se incluye esta librería para poder utilizar la función pow
using namespace std;
const double PI=3.1415;

double areaDelaEsfera(double r)
{
    return(4 * PI * r * r);
}

double volumenDelaEsfera(double r)
{
    return (4.0/3 * PI * pow(r,3));
}

int main()
{ double radio=0, area=0, volumen=0;
  double z;
  cout << "Radio : ";
  cin >> radio;
  cout << "\n";
  cout << "El area es  : " << areaDelaEsfera(radio) << "\n";
  cout << "El volumen es : " << volumenDelaEsfera(radio) << "\n";
  return 0;
}
```


Ejemplo 2:

En el siguiente programa se desea convertir grados Fahrenheit a grados Celsius.

La fórmula para convertir entre grados de estos dos tipos de escala de temperaturas es:

$$\frac{C}{5} = \frac{F - 32}{9}$$

En la solución que se muestra a continuación, hay un error de lógica, podrías encontrar el error?.

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    double fahrenheit=0, celsius=0;
```

```
    cout << "Fahrenheit : ";
```

```
    cin >> fahrenheit;
```

```
    celsius = 5/9 *(fahrenheit - 32);
```

```
    cout << "Grados Celsius " << celsius;
```

```
    return 0;
```

```
}
```

Corrida número 1:

Fahrenheit : 78

Grados Celsius 0

Corrida número 2:

Fahrenheit : 120

Grados Celsius 0



Precedencia y asociatividad:

Una expresión que tenga dos o más operadores es una **expresión compuesta**. Evaluar una expresión compuesta involucra que se agrupen los operandos y los operadores. La precedencia y asociatividad, determina como los operandos se agrupan para poder evaluar el expresión.

Los operandos de operadores con alto grado de precedencia se agrupan más fuertemente que que los operandos de operadores con menor grado de precedencia.

Ej: la multiplicación y división tienen el mismo grado de precedencia entre ellos, pero tiene mayor precedencia que la suma.

Cuando se tiene una expresión compuesta, formada por operadores aritméticos del mismo grado de precedencia estos se evalúan de izquierda a derecha.

Precedencia y asociatividad:

Ej:

$3 + 4 * 5$ es 23 y no 35 por la precedencia de operadores

$20 - 15 - 3$ es 2 por la asociatividad. Como el - y + tiene el mismo grado de precedencia, la expresión se evalúa de izq a derecha.

$6 + 3 * 4 / 2 + 2$ es 14 esta expresión equivale a:
 $((6 + ((3*4) / 2)) + 2)$

Estructuras de control

Las estructuras de Control:

Se clasifican en:

Selectivas



**if else
switch**

Repetitivas



**while
do while
for**

if

```
if ( condición)  
    instrucción 1;
```

```
if ( condición)  
{  
    instrucción 1;  
    instrucción 2;  
    instrucción 3;  
    ....  
    instrucción n;  
}
```



```
if ( condición)  
    instrucción1;  
else  
    instrucción2;
```

```
if ( condición)  
{ instrucción1;  
  instrucción2;  
  instrucción3;  
  ....  
  instrucciónn;  
}  
else  
{ instrucciónx;  
  instruccióny;  
  instrucciónz;  
  ....  
  instrucciónw;  
}
```

```
if ( condición)  
    instrucción1;  
else  
{ instrucciónx;  
  instruccióny;  
  instrucciónz;  
  ....  
  instrucciónw;  
}
```


Ejemplo 3:

Escribir un programa que permita leer como dato un número entero e imprima si el número es par, es impar o es cero.



Solucion 1 - main.cpp

```
#include <iostream>
using namespace std;

int main()
{ int numero=0;

  cout << "Numero: ";
  cin >> numero;
  if( !numero)
    cout << "El numero es cero ";
  else
    if (numero%2==0)
      cout << "El numero es par ";
    else
      cout << "El numero es impar";
  return 0;
}
```

Solucion 2 - main.cpp

```
#include <iostream>
using namespace std;
int main()
{ int numero=0;
  cout << "Numero: ";
  cin >> numero;
  if( !numero)
  { cout << "*****\n";
    cout << "El numero es cero\n";
    cout << "*****\n";
  }
  else
    if (numero%2==0)
    { cout << "*****\n";
      cout << "El numero es par\n";
      cout << "*****\n";
    }
    else
    { cout << "*****\n";
      cout << "El numero es impar\n";
      cout << "*****\n";
    }
  return 0;
}
```

switch

Esta instrucción permite seleccionar un bloque de código a ejecutar según el valor de la expresión.

```
switch(expresión)
{
    case opción1:
        bloque_de_código 1;
    case opción2:
        bloque_de_código 2;
    ...
    default:
        bloque_de_código;
}
```

Las opciones tienen que ser datos ordinales, no pueden ser: float, double o string.
Las opciones no pueden ser rangos.

A partir de la opción seleccionada todos los bloques inferiores se ejecutarán al menos que se incluya la instrucción **break** antes del bloque para detener la ejecución de los bloques posteriores.

```
switch (expresión)
{
    case opción_a:
        bloque de instrucciones a;
        break; // termina el switch y continua
    case opción_b:
        bloque de instrucciones b;
        break; // termina el switch y continua
    ...
}
```

Ejemplo 4:

Escribir un programa que permita leer un número entero que podría tener los valores 1,2,3 ó 4, e imprima la estación de año que le corresponda según la tabla :

- 1 Otoño**
- 2 Invierno**
- 3 Primavera**
- 4 Verano**

main.cpp

```
#include <iostream>
using namespace std;
int main()
{ int numero=0;

  cout <<"Numero :";
  cin>>numero;
  switch(numero)
  {
    case 1: cout<<"Otoño\n";
            break;
    case 2: cout<<"Invierno\n";
            break;
    case 3: cout<<"Primavera\n";
            break;
    case 4: cout<<"Verano\n";
            break;
    default:
            cout << "No corresponde a una estación";
  }
  return 0;
}
```

Modifique el programa anterior para que ahora:

Si es 1, 10, 100 ó 1000 imprima Otoño

Si es 2, 20, 200 ó 2000 imprima Invierno

Si es 3, 30, 300 ó 3000 imprima Primavera

Si es 4, 40, 400 ó 4000. imprima Verano


```
#include <iostream>
using namespace std;
int main()
{ int numero=0;

  cout <<"Numero :";
  cin>>numero;
  switch(numero)
  {
    case 1:
    case 10:
    case 100:
    case 1000: cout<<"Otoño\n"; break;
    case 2:
    case 20:
    case 200:
    case 2000: cout<<"Invierno\n"; break;
    case 3:
    case 30:
    case 300:
    case 3000: cout<<"Primavera\n"; break;
    case 4:
    case 40:
    case 400:
    case 4000: cout<<"Verano\n"; break;
    default:
      cout << "No corresponde a una estación";
  }
  return 0;
}
```

ESTRUCTURAS DE CONTROL REPETITIVAS:

Repetitivas



while
do while
for

while

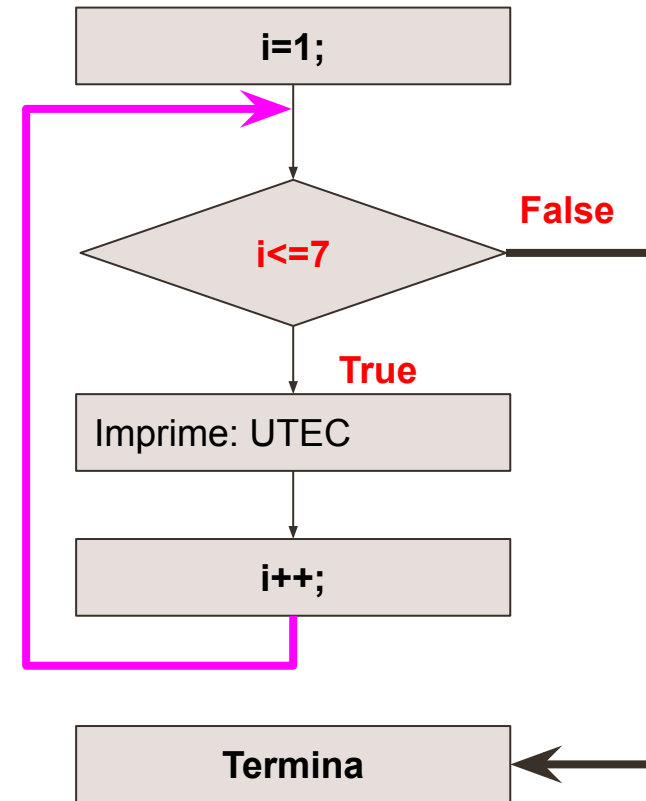
Este bucle, permite ejecutar el conjunto de instrucciones, mientras la expresión lógica o condición sea verdad. El bucle termina cuando la expresión lógica es falsa

```
while(expresiónlógica)  
    instruccion;
```

```
while(expresiónlógica)  
{  instrucción_1;  
    instrucción_2;  
    instrucción_3;  
}
```

Se imprime 7 veces la palabra UTEC, usando el bucle while

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int i;
7
8      i=1;
9      while(i<=7)
10     {
11         cout << "UTEC \n";
12         i++;
13     }
14     return(0);
15 }
```



do while

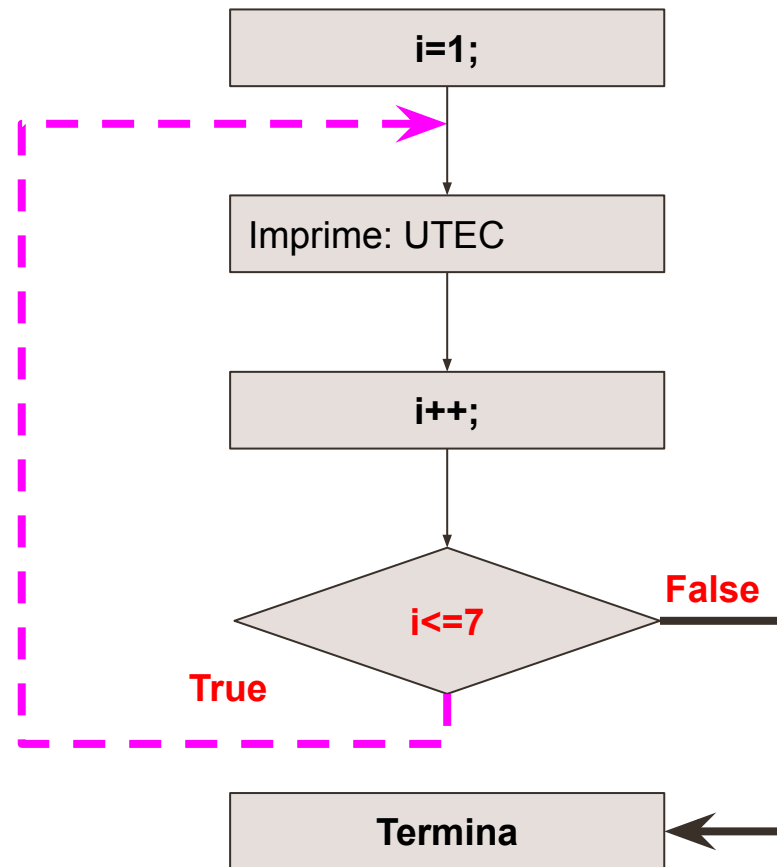
Este bucle se ejecuta mientras la expresión lógica sea verdad, termina cuando la expresión lógica es falsa.

Note, que la condición lógica se verifica al final, por lo tanto este bucle por lo menos se ejecuta una vez.

```
do
{ instruccion_1
  instruccion_2;
  instruccion_3;
  ...
  instruccion_n;
}while(expresion_logica);
```

Se imprime 7 veces la palabra UTEC, utilizando el bucle: **do while**

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int i;
7
8      i=1;
9      do
10     {
11         cout << "UTEC \n";
12         i++;
13     }while(i<=7);
14
15     return(0);
16 }
```



Tiene diferentes formas de uso:

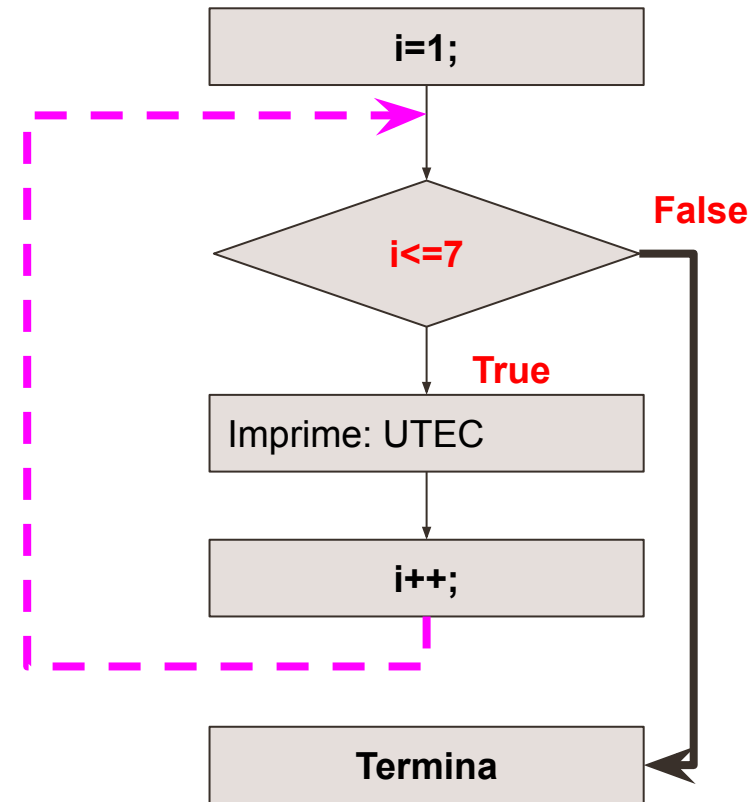
- **Uso tradicional del for**

```
for(inicialización ; condición; variaciones)  
    instrucción 1;
```

```
for(inicialización ; condición; variaciones)  
{  
    instrucción 1;  
    instrucción 2;  
    ...  
    instrucción n;  
}
```

Se imprime 7 veces la palabra UTEC, usando for

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6
7      for(int i=1; i<=7; i++)
8          cout << "UTEC \n";
9
10     return(0);
11 }
```



- Realizando más de una inicialización

```
for(inicialización ; condición; variaciones)  
    instrucción 1;
```

```
//--- Halla la suma de los múltiplos de cinco, desde el 10 al 95  
#include <iostream>  
using namespace std;  
  
int main()  
{ int suma;  
  
  for(int contador=10, suma=0; contador<=95; contador+=5 )  
    suma+=contador;  
  cout << "La sumatoria es " << suma;  
  return 0;  
}
```

- Sin usar la parte de la inicialización

```
for( ; condición; variaciones)  
    instrucción 1;
```

*//---Imprime de manera descendente el abecedario
utilizando letras mayúsculas*

```
#include <iostream>  
using namespace std;
```

```
int main()  
{ char letra='Z';  
  for( ; letra>='A'; letra--)  
    cout << letra << " ";  
  return 0;  
}
```

- Sin usar la parte de la condición (se supondrá que la condición es verdadera).

```
for(inicialización ; ; variaciones)
    {bloque de instrucciones }
```

```
//-- Imprime numeros del 1 al 1000
#include <iostream>
using namespace std;
int main()
{
    /* no hay condicion */
    for(int contador=1;          ; contador++ )
    {
        cout << contador << " ";
        if(contador==1000)
            break;
    }
    //-- esta construcción no es buena, pero puede ilustrar el trabajo del for.
    return 0;
}
```

Usando rangos:
Se imprime 7 veces la palabra UTEC

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6      int rango[] = {1,2,3,4,5,6,7};
7
8      for( auto i:rango)
9          cout << "UTEC \n";
10
11     return(0);
12 }
```

- Usando rangos:

```
for( declaración : expresión)  
    {bloque de instrucciones}
```

```
//--Imprime los días que se indican en vector  
#include <iostream>  
#include <vector>  
using namespace std;  
int main()  
{  
    vector <string> dias = {"lunes", "martes", "miercoles", "jueves",  
                           "viernes", "sabado", "domingo" };  
  
    for( auto i:dias)  
        cout << i << " ";  
    return 0;  
}
```

- Usando rangos: otro ejemplo

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    vector<int> a = {1,2,3,4,5,6,7};

    //--- imprime todos los valores del vector a
    cout << "Valores del vector \"a\" \n";
    for(auto i:a)
        cout << i << " ";
    //--- modifica los valores del vector a, reemplazando el valor por su cuadrado
    for( auto &i:a ) //--- note que al usar &i, i es una referencia.
        i *= i;
    //--- imprime todos los valores del vector a
    cout << "\nValores del vector \"a\" modificados\n";
    for(auto i:a)
        cout << i << " ";
    return 0;
}
```

El programa imprime:

```
Valores del vector "a"
1 2 3 4 5 6 7
Valores del vector "a" modificados
1 4 9 16 25 36 49
```

Ejemplo 5:

Realiza un programa en C++, que permita leer como dato un número entero positivo que sea de al menos 3 dígitos y el programa indique:

- La cantidad de dígitos que tiene el número,
- Cuántos dígitos son pares
- Cuántos dígitos son impares.

Ejemplo de la Corrida del programa:

```
Numero de 3 digitos por favor :56  
Numero de 3 digitos por favor :3  
Numero de 3 digitos por favor :5678432087
```

```
El numero que ingresaste 5678432087 tiene  
Numero de digitos : 10  
Numero de digitos pares : 6  
Numero de digitos impares: 4
```

```
#include <iostream>
using namespace std;
int main()
{ unsigned long int numero=0;

do{
    cout << "Numero de 3 digitos por favor :";
    cin >> numero;
}while(numero<1000);

unsigned long int numDeDigitos=0;
unsigned long int numDeDigitosImpares=0;
unsigned long int numDeDigitosPares=0;

for(unsigned long int copiaDelNumero=numero, digito=0; copiaDelNumero>0; copiaDelNumero/=10)
{
    digito = copiaDelNumero%10;
    numDeDigitos++;
    if(digito%2 == 0)
        numDeDigitosPares++;
    else
        numDeDigitosImpares++;
}
```

Continua...


```
cout << "\n\n";  
cout << "El numero que ingresaste " << numero << " tiene \n";  
cout << "Numero de digitos : " << numDeDigitos << "\n";  
cout << "Numero de digitos pares : " << numDeDigitosPares << "\n";  
cout << "Numero de digitos impares: " << numDeDigitosImpares;  
return 0;  
}
```

Averigua para qué sirve:

break

continue

exit

? :

En esta sesión aprendiste:

- 1. Las partes de un programa en C++,**
- 2. Tipos de datos, variables y constantes.**
- 3. Instrucciones de I/O**
- 4. Estructuras de control selectivas y repetitivas**



TECHSUYO LIMA

¿Qué harías si tuvieras a Silicon Valley por 24 horas?



Victor Cadenas
Apple



Rosalva Gallardo
Google



Victor Laguna
Facebook



Renzo Sanchez-Silva
Netflix



Mayra Soto
HoloBuilder



Omar Florez
Capital One

3 de abril, 2019
Swissôtel Lima

¡Regístrate ahora!
www.techsuyo.org/registration

Organizadores:



Patrocinadores:



Partners:



<https://www.techsuyo.org/stem>

CS1102 – PROGRAMACIÓN ORIENTADA A OBJETOS 1

CICLO 2019-1

Unidad 1:

Elementos de Programación y Estructuras de Control

<http://bit.ly/2HRBWgq>

Profesores:

Ernesto Cuadros- Vargas, PhD. ecuadros@utec.edu.pe

María Hilda Bermejo, M. Sc. mbermejo@utec.edu.pe