

CS1112: Programación 2

Unidad 2: Funciones y recursividad

Sesión de Teoría 3

Profesor:

José Antonio Fiestas Iquira jfiestas@utec.edu.pe

Material elaborado por:

Maria Hilda Bermejo, José Fiestas, Rubén Rivas, Jaime Farfán



Índice:

- **Unidad 2: Funciones y recursividad**
 - **Ámbito de una variable**
 - **Paso de parámetros a funciones. Transferencia por valor y por referencia**
 - **Recursividad.**
 - **Funciones Lambda**

1

Unidad 2: Funciones y recursividad

UTEC

Logro de la sesión:

Al finalizar la sesión, los alumnos:

- **Entienden el concepto de variables globales.**
- **Comprenden el concepto de transferencia por valor y por referencia.**
- **Usan funciones recursivas y no-recursivas**

Conceptos Previos:

Ámbito o *Scope* de una variable.

Creando tipos de datos.

Scope o ámbito (ejemplo 01)

```
int main()
{
    int a, b, c;           // variables que tienen alcance de función main
    cin >> a;
    cin >> b;
    c = mayor(a,b);
    cout << "El mayor valor es: " << c << endl;

    for (int i = 0; i < 5; i++) // la variable i solo existe en el bucle "for"
    {
        cout << "Valor de i : " << i << endl;
    }
    // A partir de esta parte del código la variable i no existe.

    if (a < b)
    {
        int temporal = a;    // el alcance de la variable temporal es el bloque if
        a = b;
        b = temporal;
    }
}
```

Scope o ámbito - Variables Globales (ejemplo 2)

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Numero de llamadas : " << linea;
    return 0;
}
```

línea y contador

Son variables globales.

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

0

contador

0

Lo que imprime:

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

0

contador

0

Lo que imprime:

n

1

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

0

1

contador

0

Lo que imprime:

n

1

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

0

1

contador

0

1

n

1

Lo que imprime:

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

0

1

contador

0

1

n

1

Lo que imprime:

Valor : 1

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

0

1

contador

0

1

Lo que imprime:

Valor : 1

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

contador

0

0

1

1

Lo que imprime:

Valor : 1

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

0

1

contador

0

1

n

2

Lo que imprime:

Valor : 1

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

contador

0

0

1

1

2

3

n

2

Lo que imprime:

Valor : 1

Valor : 3

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

contador

0

0

1

1

2

3

Lo que imprime:

Valor : 1

Valor : 3

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

contador

0

0

1

1

2

3

Lo que imprime:

Valor : 1

Valor : 3

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

0

1

2

contador

0

1

3

n

5

Lo que imprime:

Valor : 1

Valor : 3

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

0

1

2

3

contador

0

1

3

n

5

Lo que imprime:

Valor : 1

Valor : 3

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

contador

0

0

1

1

2

3

n

5

3

8

Lo que imprime:

Valor : 1

Valor : 3

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

contador

0

0

1

1

2

3

3

8

n

5

Lo que imprime:

Valor : 1

Valor : 3

Valor : 8

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

contador

0

0

1

1

2

3

3

8

Lo que imprime:

Valor : 1

Valor : 3

Valor : 8

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de Llamadas : " << linea;
    return 0;
}
```

linea

contador

0

0

1

1

2

3

3

8

Lo que imprime:

Valor : 1

Valor : 3

Valor : 8

Numero de Llamadas : 3

Scope o ámbito - Variables Globales:

funciones.h

```
#include <iostream>
using namespace std;
int linea = 0;
int contador=0;
void imprimir(int n);
```

funciones.cpp

```
#include "funciones.h"
void imprimir(int n){
    linea++;
    contador = contador + n;
    cout << "Valor : " << contador << "\n";
}
```

main.cpp

```
#include "funciones.h"
int main(){
    imprimir(1);
    imprimir(2);
    imprimir(5);
    cout << "Número de llamadas : " << linea;
    return 0;
}
```

linea

contador

0

0

1

1

2

3

3

8

Lo que imprime:

Valor : 1

Valor : 3

Valor : 8

Type Aliases:

Algunas veces se necesita nuevos nombres para un tipo dato, porque facilita los cambios o mantenimiento en los programas:

Ej1:

`typedef int int32_t; //`-- luego se puede declarar la variable como sigue:

`int32_t numero=34;`

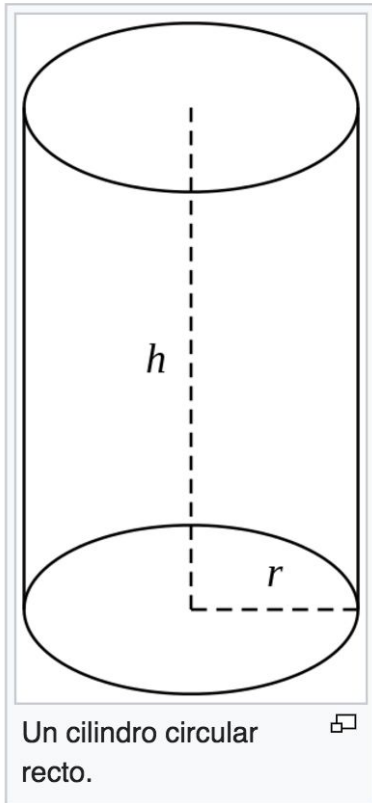
Ej2:

`typedef double tipoReal; //`-- luego se puede declarar la variable como si

`tipoReal dato=3.5643;`

Ejemplo 3 :

Escribir un programa que permita hallar el área total y el volumen de un cilindro circular recto, si se conoce el valor del radio y la altura.



$$\text{areaDeLaBase} = \pi r^2$$

$$\text{areaLateral} = 2 \pi r h$$

$$\text{areaTotal} = 2 \pi r h + 2 \pi r^2$$

$$\text{volumen} = \pi r^2 h$$

Solución 1: Todo el código está en un solo archivo main.cpp

```
#include <iostream>
using namespace std;
const double PI=3.1415;

double LeeDato(string mensaje)
{
    //-----
    double dato;
    do
    {
        cout << mensaje;
        cin >> dato;
    } while (dato <= 0);
    return dato;
}

double areaDeLaBase(double r)
{
    //-----
    return (PI*r*r);
}

double areaTotal(double r, double h)
{
    //-----
    return (2*PI*r*h + 2*areaDeLaBase(r));
}

double volumen(double r, double h)
{
    //-----
    return (areaDeLaBase(r) * h);
}
```

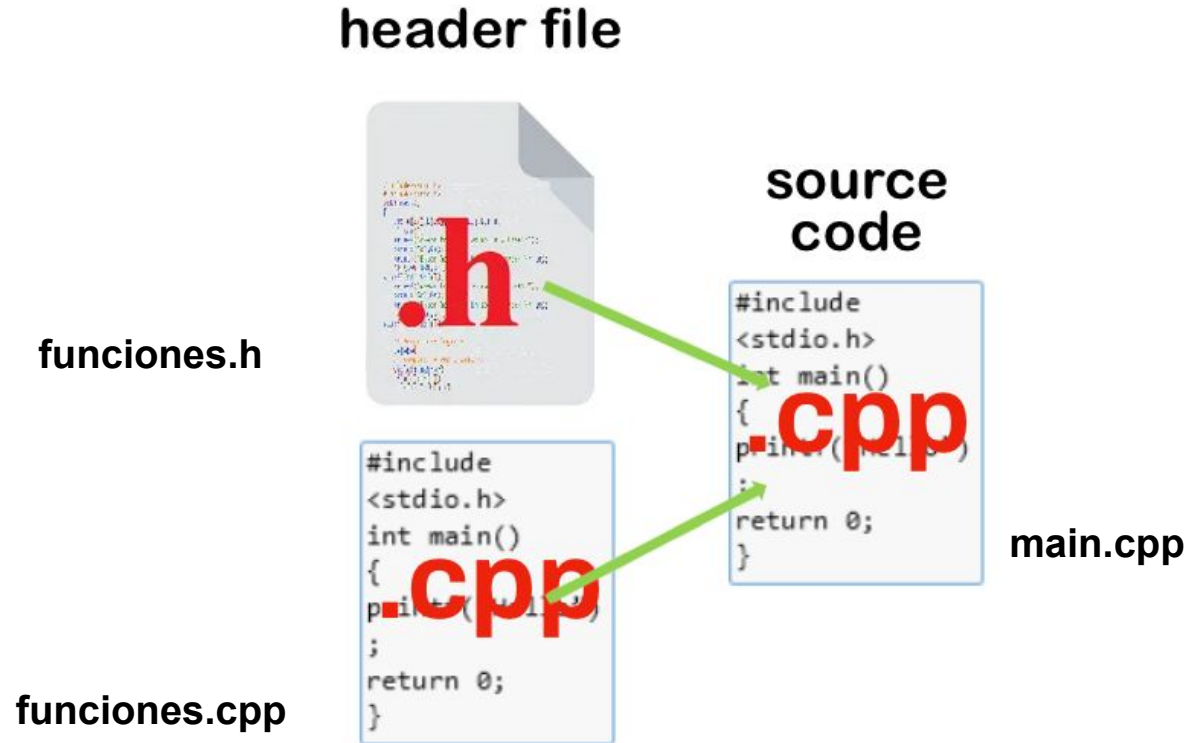
```
int main()
{
    double radio, altura;

    radio=LeeDato("Radio: ");
    altura=LeeDato("Altura: ");
    cout << "Area total: " << areaTotal(radio, altura);
    cout << "\n";
    cout << "Volumen : " << volumen(radio, altura);
    return 0;
}
```

Pantalla de salida :

```
Radio: 0
Radio: -1
Radio: 10
Altura: 3
Area total: 816.79
Volumen : 942.45
```

El código puede también estar distribuido en varios archivos:



y utilizar tipos genéricos. Esta es una buena práctica que facilita el mantenimiento del código

main.cpp

```
#include <iostream>
#include "UFunciones.h"
using namespace std;

int main()
{tipo_Real radio, altura;

    radio=LeeDato("Radio: ");
    altura=LeeDato("Altura: ");
    cout << "Area total: " << areaTotal(radio,
altura);
    cout << "\n";
    cout << "Volumen    : " <<
volumen(radio,altura);
    return 0;
}
```

UFunciones.h

```
#ifndef CILINDRO1_UFUNCIONES_H
#define CILINDRO1_UFUNCIONES_H

#include <iostream>
using namespace std;

typedef double tipo_Real;

const tipo_Real PI=3.1415;

tipo_Real LeeDato(string mensaje);
tipo_Real areaDeLaBase(tipo_Real r);
tipo_Real areaTotal(tipo_Real r, tipo_Real
h);
tipo_Real volumen(tipo_Real r, tipo_Real h);

#endif //CILINDRO1_UFUNCIONES_H
```

UFunciones.h

```
#ifndef CILINDRO1_UFUNCIONES_H
#define CILINDRO1_UFUNCIONES_H

#include <iostream>
using namespace std;

typedef double tipo_Real;

const tipo_Real PI=3.1415;

tipo_Real LeeDato(string mensaje);
tipo_Real areaDeLaBase(tipo_Real r);
tipo_Real areaTotal(tipo_Real r, tipo_Real h);
tipo_Real volumen(tipo_Real r, tipo_Real h);

#endif //CILINDRO1_UFUNCIONES_H
```

UFunciones. cpp

```
#include "UFunciones.h"

tipo_Real LeeDato(string mensaje)
{
    //-----
    tipo_Real dato;
    do
    {
        cout << mensaje;
        cin >> dato;
    }while(dato<=0);
    return dato;
}

tipo_Real areaDeLaBase(tipo_Real r)
{
    //-----
    return(PI*r*r);
}

tipo_Real areaTotal(tipo_Real r, tipo_Real h)
{
    //-----
    return(2*PI*r*h + 2*areaDeLaBase(r));
}

tipo_Real volumen(tipo_Real r, tipo_Real h)
{
    //-----
    return(areaDeLaBase(r) * h);
}
```

Transferencia de parámetros por referencia

Paso por referencia: se modifica la variable (en su espacio de memoria). *E.g. void fillCup (int &cup);*

Paso por valor: se crea una copia local de la variable en la función *E.g. void fillCup (int cup);*

pass by reference



fillCup()

pass by value



fillCup()

Ejemplo 4 : paso por valor y referencia

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

Lo que se imprime Programa

```
void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{  auto auxiliar = a;
   a= b;
   b=auxiliar;
}

int main()
{  int a=10, b=33;

   cout << "a = " << a << " " << "b = " << b << "\n";
   AumentaenCinco(a,b);
   cout << "a = " << a << " " << "b = " << b << "\n";
   cout << "\n";
   a=30; b=88;
   //--- ahora usamos transferencia por referencia
   cout << "Valores antes del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   IntercambiarValores(a,b);
   cout << "Valores después del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   return 0;
}
```

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

Lo que se imprime Programa

```
void AumentaenCinco(int a,int b)
```

```
//-----
{ a = a + 5;
  b = b + 5;
}
```

```
void IntercambiarValores(int &a, int &b)
```

```
//-----
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}
```

```
int main()
```

```
{ int a=10, b=33;
```

```
  cout << "a = " << a << " " << "b = " << b << "\n";
  AumentaenCinco(a,b);
  cout << "a = " << a << " " << "b = " << b << "\n";
  cout << "\n";
  a=30; b=88;
```

```
  //--- ahora usamos transferencia por referencia
```

```
  cout << "Valores antes del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  IntercambiarValores(a,b);
  cout << "Valores después del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  return 0;
}
```

a

10

b

33

```

#include <iostream>
using namespace std;

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{  auto auxiliar = a;
   a= b;
   b=auxiliar;
}

int main()
{  int a=10, b=33;

   cout << "a = " << a << " " << "b = " << b << "\n";
   AumentaenCinco(a,b);
   cout << "a = " << a << " " << "b = " << b << "\n";
   cout << "\n";
   a=30; b=88;
   //--- ahora usamos transferencia por referencia
   cout << "Valores antes del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   IntercambiarValores(a,b);
   cout << "Valores después del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   return 0;
}

```

Lo que se imprime Programa

a=10 b=33

a

10

b

33

```

#include <iostream>
using namespace std;

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{  auto auxiliar = a;
   a= b;
   b=auxiliar;
}

int main()
{  int a=10, b=33;

   cout << "a = " << a << " " << "b = " << b << "\n";
   AumentaenCinco(a,b);
   cout << "a = " << a << " " << "b = " << b << "\n";
   cout << "\n";
   a=30; b=88;
   //--- ahora usamos transferencia por referencia
   cout << "Valores antes del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   IntercambiarValores(a,b);
   cout << "Valores después del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   return 0;
}

```

Lo que se imprime Programa

a=10 b=33

a

10

b

33

```

#include <iostream>
using namespace std;

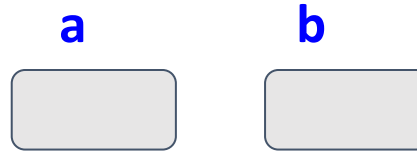
void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{  auto auxiliar = a;
   a= b;
   b=auxiliar;
}

int main()
{  int a=10, b=33;

   cout << "a = " << a << " " << "b = " << b << "\n";
   AumentaenCinco(a,b);
   cout << "a = " << a << " " << "b = " << b << "\n";
   cout << "\n";
   a=30; b=88;
   //--- ahora usamos transferencia por referencia
   cout << "Valores antes del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   IntercambiarValores(a,b);
   cout << "Valores después del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   return 0;
}

```



Lo que se imprime Programa

a=10 b=33



```

#include <iostream>
using namespace std;

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{  auto auxiliar = a;
   a= b;
   b=auxiliar;
}

int main()
{  int a=10, b=33;

   cout << "a = " << a << " " << "b = " << b << "\n";
   AumentaenCinco(a,b);
   cout << "a = " << a << " " << "b = " << b << "\n";
   cout << "\n";
   a=30; b=88;
   //--- ahora usamos transferencia por referencia
   cout << "Valores antes del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   IntercambiarValores(a,b);
   cout << "Valores después del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   return 0;
}

```

a	b
10	33

Lo que se imprime Programa

a=10 b=33

a	b
10	33

```

#include <iostream>
using namespace std;

void AumentaenCinco(int a,int b)
//-----
{  a = a + 5;
   b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{  auto auxiliar = a;
   a= b;
   b=auxiliar;
}

int main()
{  int a=10, b=33;

   cout << "a = " << a << " " << "b = " << b << "\n";
   AumentaenCinco(a,b);
   cout << "a = " << a << " " << "b = " << b << "\n";
   cout << "\n";
   a=30; b=88;
   //--- ahora usamos transferencia por referencia
   cout << "Valores antes del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   IntercambiarValores(a,b);
   cout << "Valores después del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   return 0;
}

```

a	b
15	33

Lo que se imprime Programa

a=10 b=33

a	b
10	33


```

#include <iostream>
using namespace std;

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{  auto auxiliar = a;
   a= b;
   b=auxiliar;
}

int main()
{  int a=10, b=33;

   cout << "a = " << a << " " << "b = " << b << "\n";
   AumentaenCinco(a,b);
   cout << "a = " << a << " " << "b = " << b << "\n";
   cout << "\n";
   a=30; b=88;
   //--- ahora usamos transferencia por referencia
   cout << "Valores antes del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   IntercambiarValores(a,b);
   cout << "Valores después del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   return 0;
}

```

a	b
15	38

Lo que se imprime Programa

a=10 b=33

a	b
10	33

```

#include <iostream>
using namespace std;

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{  auto auxiliar = a;
   a= b;
   b=auxiliar;
}

int main()
{  int a=10, b=33;

   cout << "a = " << a << " " << "b = " << b << "\n";
   AumentaenCinco(a,b);
   cout << "a = " << a << " " << "b = " << b << "\n";
   cout << "\n";
   a=30; b=88;
   //--- ahora usamos transferencia por referencia
   cout << "Valores antes del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   IntercambiarValores(a,b);
   cout << "Valores después del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   return 0;
}

```

Lo que se imprime Programa

a=10 b=33

a	b
10	33

Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{  auto auxiliar = a;
   a= b;
   b=auxiliar;
}

int main()
{  int a=10, b=33;

   cout << "a = " << a << " " << "b = " << b << "\n";
   AumentaenCinco(a,b);
   cout << "a = " << a << " " << "b = " << b << "\n";
   cout << "\n";
   a=30; b=88;
   //--- ahora usamos transferencia por referencia
   cout << "Valores antes del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   IntercambiarValores(a,b);
   cout << "Valores después del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   return 0;
}
```

Lo que se imprime Programa

a=10 b=33

a=10 b=33

a

10

b

33

Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{  auto auxiliar = a;
   a= b;
   b=auxiliar;
}

int main()
{  int a=10, b=33;

   cout << "a = " << a << " " << "b = " << b << "\n";
   AumentaenCinco(a,b);
   cout << "a = " << a << " " << "b = " << b << "\n";
   cout << "\n";
   a=30; b=88;
   //--- ahora usamos transferencia por referencia
   cout << "Valores antes del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   IntercambiarValores(a,b);
   cout << "Valores después del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   return 0;
}
```

Lo que se imprime Programa

a=10 b=33

a=10 b=33

a

10

b

33

Transferencia por valor y Transferencia por Referencia

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

```
void AumentaenCinco(int a,int b)
```

```
//-----
```

```
{ a = a + 5;
  b = b + 5;
}
```

```
void IntercambiarValores(int &a, int &b)
```

```
//-----
```

```
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}
```

```
int main()
```

```
{ int a=10, b=33;
```

```
  cout << "a = " << a << " " << "b = " << b << "\n";
```

```
  AumentaenCinco(a,b);
```

```
  cout << "a = " << a << " " << "b = " << b << "\n";
```

```
  cout << "\n";
```

```
  a=30; b=88;
```

```
  //--- ahora usamos transferencia por referencia
```

```
  cout << "Valores antes del intercambio\n";
```

```
  cout << "a = " << a << " " << "b = " << b << "\n";
```

```
  IntercambiarValores(a,b);
```

```
  cout << "Valores después del intercambio\n";
```

```
  cout << "a = " << a << " " << "b = " << b << "\n";
```

```
  return 0;
```

```
}
```

Lo que se imprime Programa

a=10 b=33

a=10 b=33

a

30

b

88

Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}

int main()
{ int a=10, b=33;

  cout << "a = " << a << " " << "b = " << b << "\n";
  AumentaenCinco(a,b);
  cout << "a = " << a << " " << "b = " << b << "\n";
  cout << "\n";
  a=30; b=88;
  //--- ahora usamos transferencia por referencia
  cout << "Valores antes del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  IntercambiarValores(a,b);
  cout << "Valores después del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  return 0;
}
```

Lo que se imprime Programa

a=10 b=33

a=10 b=33

Valores antes del intercambio

a

30

b

88

Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}

int main()
{ int a=10, b=33;

  cout << "a = " << a << " " << "b = " << b << "\n";
  AumentaenCinco(a,b);
  cout << "a = " << a << " " << "b = " << b << "\n";
  cout << "\n";
  a=30; b=88;
  //--- ahora usamos transferencia por referencia
  cout << "Valores antes del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  IntercambiarValores(a,b);
  cout << "Valores después del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  return 0;
}
```

Lo que se imprime Programa

a=10 b=33

a=10 b=33

Valores antes del intercambio

a

30

b

88

Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{  auto auxiliar = a;
   a= b;
   b=auxiliar;
}

int main()
{  int a=10, b=33;

   cout << "a = " << a << " " << "b = " << b << "\n";
   AumentaenCinco(a,b);
   cout << "a = " << a << " " << "b = " << b << "\n";
   cout << "\n";
   a=30; b=88;
   //--- ahora usamos transferencia por referencia
   cout << "Valores antes del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   IntercambiarValores(a,b);
   cout << "Valores después del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   return 0;
}
```

Lo que se imprime Programa

a=10 b=33

a=10 b=33

Valores antes del intercambio

a=30 b=88

a

30

b

88

Transferencia por valor y Transferencia por Referencia

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

```
void AumentaenCinco(int a,int b)
```

```
//-----
{ a = a + 5;
  b = b + 5;
}
```

```
void IntercambiarValores(int &a, int &b)
```

```
//-----
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}
```

```
int main()
```

```
{ int a=10, b=33;
```

```
  cout << "a = " << a << " " << "b = " << b << "\n";
  AumentaenCinco(a,b);
  cout << "a = " << a << " " << "b = " << b << "\n";
  cout << "\n";
  a=30; b=88;
```

```
//--- ahora usamos transferencia por referencia
```

```
  cout << "Valores antes del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  IntercambiarValores(a,b);
```

```
  cout << "Valores después del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  return 0;
}
```

Lo que se imprime Programa

a=10 b=33

a=10 b=33

Valores antes del intercambio

a=30 b=88

a

b

a

30

b

88

Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}

int main()
{ int a=10, b=33;

  cout << "a = " << a << " " << "b = " << b << "\n";
  AumentaenCinco(a,b);
  cout << "a = " << a << " " << "b = " << b << "\n";
  cout << "\n";
  a=30; b=88;
  //--- ahora usamos transferencia por referencia
  cout << "Valores antes del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  IntercambiarValores(a,b);
  cout << "Valores después del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  return 0;
}
```

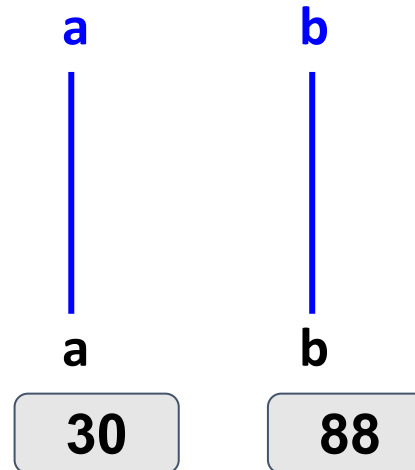
Lo que se imprime Programa

a=10 b=33

a=10 b=33

Valores antes del intercambio

a=30 b=88



Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{  auto auxiliar = a;
   a= b;
   b=auxiliar;
}

int main()
{  int a=10, b=33;

   cout << "a = " << a << " " << "b = " << b << "\n";
   AumentaenCinco(a,b);
   cout << "a = " << a << " " << "b = " << b << "\n";
   cout << "\n";
   a=30; b=88;
   //--- ahora usamos transferencia por referencia
   cout << "Valores antes del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   IntercambiarValores(a,b);
   cout << "Valores después del intercambio\n";
   cout << "a = " << a << " " << "b = " << b << "\n";
   return 0;
}
```

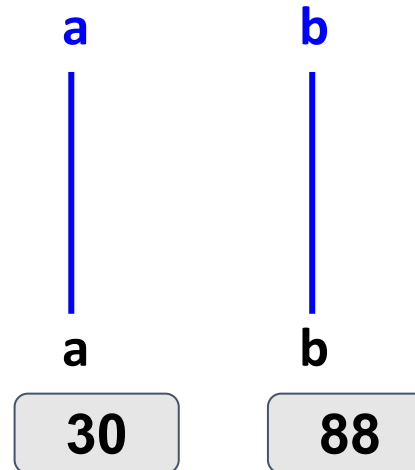
Lo que se imprime Programa

a=10 b=33

a=10 b=33

Valores antes del intercambio

a=30 b=88



Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}

int main()
{ int a=10, b=33;

  cout << "a = " << a << " " << "b = " << b << "\n";
  AumentaenCinco(a,b);
  cout << "a = " << a << " " << "b = " << b << "\n";
  cout << "\n";
  a=30; b=88;
  //--- ahora usamos transferencia por referencia
  cout << "Valores antes del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  IntercambiarValores(a,b);
  cout << "Valores después del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  return 0;
}
```

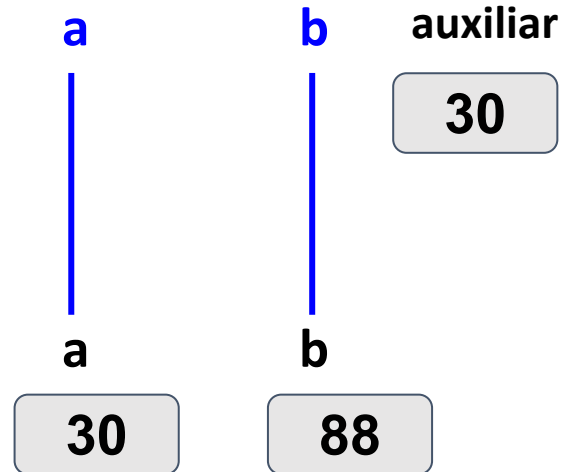
Lo que se imprime Programa

a=10 b=33

a=10 b=33

Valores antes del intercambio

a=30 b=88



Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}

int main()
{ int a=10, b=33;

  cout << "a = " << a << " " << "b = " << b << "\n";
  AumentaenCinco(a,b);
  cout << "a = " << a << " " << "b = " << b << "\n";
  cout << "\n";
  a=30; b=88;
  //--- ahora usamos transferencia por referencia
  cout << "Valores antes del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  IntercambiarValores(a,b);
  cout << "Valores después del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  return 0;
}
```

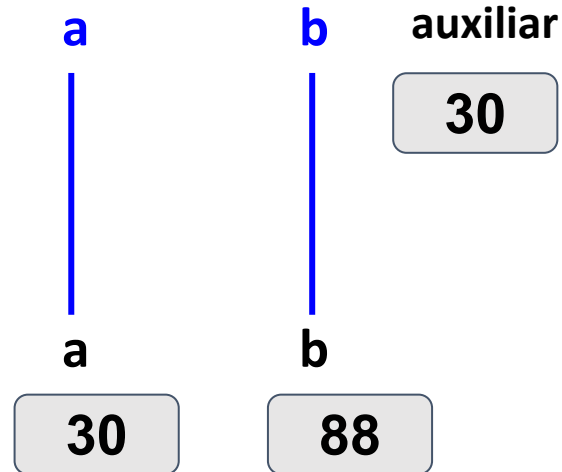
Lo que se imprime Programa

a=10 b=33

a=10 b=33

Valores antes del intercambio

a=30 b=88



Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}

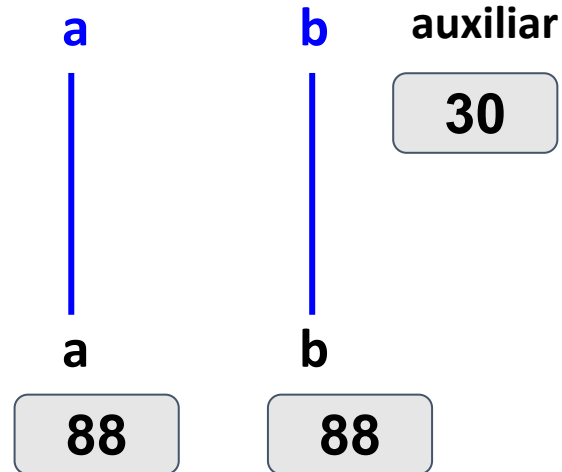
int main()
{ int a=10, b=33;

  cout << "a = " << a << " " << "b = " << b << "\n";
  AumentaenCinco(a,b);
  cout << "a = " << a << " " << "b = " << b << "\n";
  cout << "\n";
  a=30; b=88;
  //--- ahora usamos transferencia por referencia
  cout << "Valores antes del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  IntercambiarValores(a,b);
  cout << "Valores después del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  return 0;
}
```

Lo que se imprime Programa

a=10 b=33
a=10 b=33

Valores antes del intercambio
a=30 b=88



Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}

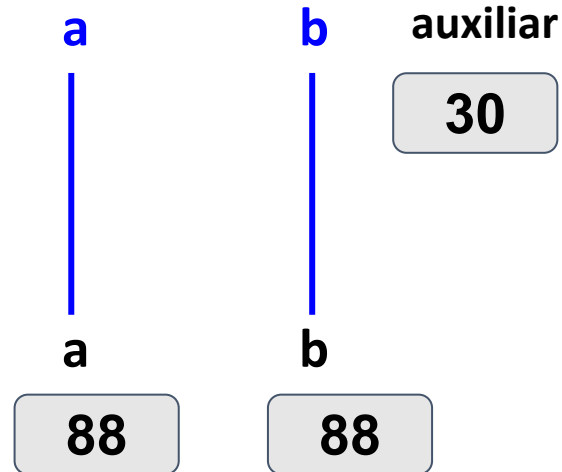
int main()
{ int a=10, b=33;

  cout << "a = " << a << " " << "b = " << b << "\n";
  AumentaenCinco(a,b);
  cout << "a = " << a << " " << "b = " << b << "\n";
  cout << "\n";
  a=30; b=88;
  //--- ahora usamos transferencia por referencia
  cout << "Valores antes del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  IntercambiarValores(a,b);
  cout << "Valores después del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  return 0;
}
```

Lo que se imprime Programa

a=10 b=33
a=10 b=33

Valores antes del intercambio
a=30 b=88



Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}

int main()
{ int a=10, b=33;

  cout << "a = " << a << " " << "b = " << b << "\n";
  AumentaenCinco(a,b);
  cout << "a = " << a << " " << "b = " << b << "\n";
  cout << "\n";
  a=30; b=88;
  //--- ahora usamos transferencia por referencia
  cout << "Valores antes del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  IntercambiarValores(a,b);
  cout << "Valores después del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  return 0;
}
```

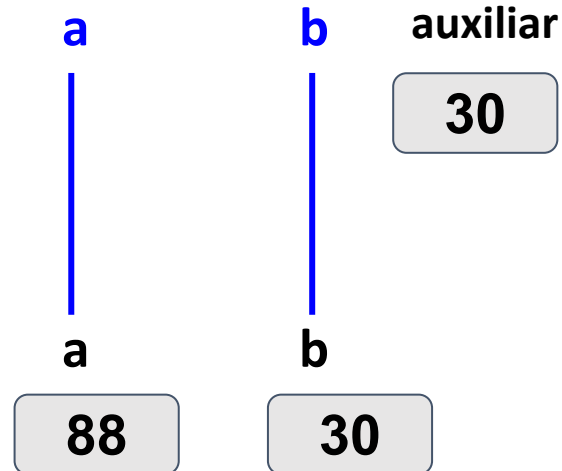
Lo que se imprime Programa

a=10 b=33

a=10 b=33

Valores antes del intercambio

a=30 b=88



Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}

int main()
{ int a=10, b=33;

  cout << "a = " << a << " " << "b = " << b << "\n";
  AumentaenCinco(a,b);
  cout << "a = " << a << " " << "b = " << b << "\n";
  cout << "\n";
  a=30; b=88;
  //--- ahora usamos transferencia por referencia
  cout << "Valores antes del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  IntercambiarValores(a,b);
  cout << "Valores después del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  return 0;
}
```

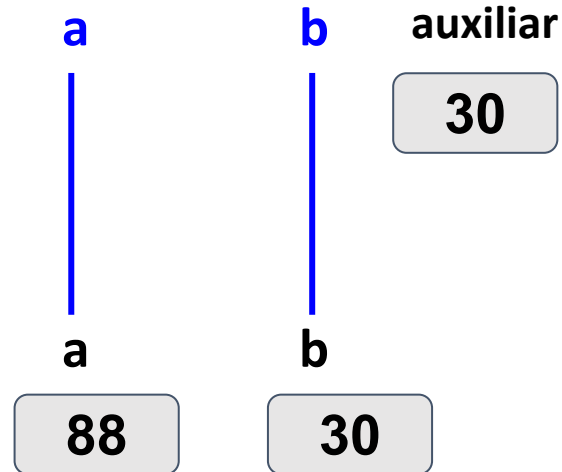
Lo que se imprime Programa

a=10 b=33

a=10 b=33

Valores antes del intercambio

a=30 b=88



Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}

int main()
{ int a=10, b=33;

  cout << "a = " << a << " " << "b = " << b << "\n";
  AumentaenCinco(a,b);
  cout << "a = " << a << " " << "b = " << b << "\n";
  cout << "\n";
  a=30; b=88;
  //--- ahora usamos transferencia por referencia
  cout << "Valores antes del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  IntercambiarValores(a,b);
  cout << "Valores después del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  return 0;
}
```

Lo que se imprime Programa

a=10 b=33

a=10 b=33

Valores antes del intercambio

a=30 b=88

a

88

b

30

Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}

int main()
{ int a=10, b=33;

  cout << "a = " << a << " " << "b = " << b << "\n";
  AumentaenCinco(a,b);
  cout << "a = " << a << " " << "b = " << b << "\n";
  cout << "\n";
  a=30; b=88;
  //--- ahora usamos transferencia por referencia
  cout << "Valores antes del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  IntercambiarValores(a,b);
  cout << "Valores después del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  return 0;
}
```

Lo que se imprime Programa

a=10 b=33

a=10 b=33

Valores antes del intercambio

a=30 b=88

Valores después del intercambio

a

88

b

30

Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}

int main()
{ int a=10, b=33;

  cout << "a = " << a << " " << "b = " << b << "\n";
  AumentaenCinco(a,b);
  cout << "a = " << a << " " << "b = " << b << "\n";
  cout << "\n";
  a=30; b=88;
  //--- ahora usamos transferencia por referencia
  cout << "Valores antes del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  IntercambiarValores(a,b);
  cout << "Valores después del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  return 0;
}
```

Lo que se imprime Programa

a=10 b=33
a=10 b=33

Valores antes del intercambio
a=30 b=88
Valores despues de intercambio
a=88 b=30

a	b
88	30

Transferencia por valor y Transferencia por Referencia

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

void AumentaenCinco(int a,int b)
//-----
{ a = a + 5;
  b = b + 5;
}

void IntercambiarValores(int &a, int &b)
//-----
{ auto auxiliar = a;
  a= b;
  b=auxiliar;
}

int main()
{ int a=10, b=33;

  cout << "a = " << a << " " << "b = " << b << "\n";
  AumentaenCinco(a,b);
  cout << "a = " << a << " " << "b = " << b << "\n";
  cout << "\n";
  a=30; b=88;
  //--- ahora usamos transferencia por referencia
  cout << "Valores antes del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  IntercambiarValores(a,b);
  cout << "Valores después del intercambio\n";
  cout << "a = " << a << " " << "b = " << b << "\n";
  return 0;
}
```

Lo que se imprime Programa

a=10 b=33
a=10 b=33

Valores antes del intercambio

a=30 b=88

Valores después del intercambio

a=88 b=30



Ejemplo 5:

Desarrolla un programa que permite leer como dato una cantidad de segundos (número mayor a 1) y el programa realice la conversión a **través del uso de una función** a horas, minutos y segundos.



main. cpp

```
#include <iostream>
#include "UMisFunciones.h"
using namespace std;

int main()
{tipo_Entero segundos;
  tipo_Entero Horas, Min, Seg;

  segundos=LeeSegundos();
  ConvertirAHorasMinSeg(segundos, Horas, Min, Seg);
  cout<<"Equivale a ";
  cout<< Horas<<" horas " << Min <<" minutos y " << Seg << "
segundos";
  return 0;
}
```

UMisFunciones. h

```
#ifndef SEGUNTOSAHORASMINSEG_UMISFUNCIONES_H
#define SEGUNTOSAHORASMINSEG_UMISFUNCIONES_H

#include <iostream>
using namespace std;

typedef long int tipo_Entero;

tipo_Entero LeeSegundos();
void ConvertirAHorasMinSeg(tipo_Entero Segundos, tipo_Entero &horas,
tipo_Entero &min, tipo_Entero &seg);

#endif //SEGUNTOSAHORASMINSEG_UMISFUNCIONES_H
```

```
#include "UMisFunciones.h"

tipo_Entero LeeSegundos()
{//-----
    tipo_Entero n;

    do{
        cout <<"Segundos <dato mayor a 1>: ";
        cin>>n;
    }while(n<=1);;
    return n;
}

void ConvertirAHorasMinSeg(tipo_Entero Segundos, tipo_Entero &horas, tipo_Entero &min, tipo_Entero &seg)
{//-----
    horas = Segundos/3600;
    Segundos %= 3600;
    min = Segundos/60;
    seg = Segundos%60;
}
```

La función *ConvertirAHorasMinSeg*, en su identificador o nombre no retorna ningún valor, recibe 4 parámetros: el primero por valor y los 3 siguientes por referencia.

Ejemplo adicional para practicar :

Existe un tipo de notación conocida como notación polaca inversa, cuya ventaja es que realiza operaciones compuestas sin necesidad de utilizar paréntesis.

Por ejemplo la siguiente operación infijo:

$$(4 + 2) * (7 - 5)$$

Se expresaría en notación polaca inversa de la siguiente forma:

$$4\ 2\ +\ 7\ 5\ -\ *$$

Como se observa en notación polaca inversa se coloca primero los 2 números seguidos por la operación, conservando la precedencia.

Ejemplo:

$$(4 - 2) * 10$$

En notación polaca inversa se expresaría:

$$4\ 2\ -\ 10\ *$$

Diseñar y escribir una función que recepcione una expresión en notación polaca inversa que se denominará:

simple_reverse_polish_calculator

Esta función recibe como parámetro una expresión aritmética en notación polaca del tipo Texto y mostrará el resultado de la expresión.

main. cpp

```
#include <iostream>
#include "SimpleReversePolishCalculator.h"

int main()
{
    std::cout << "The first result is: ";
    simple_reverse_polish_calculator("5 2 - 10 *"); // Equivalente a (5 - 2) * 10
    std::cout << "The second result is: ";
    simple_reverse_polish_calculator("10 0 /"); // Equivalente a 10 / 0
    std::cout << "The third result is: ";
    simple_reverse_polish_calculator("10 5 ^"); // Equivalente a 10 ^ 5, No existe la operación ^
    std::cout << "The fourth result is: ";
    simple_reverse_polish_calculator("10.5 5 / 3.5 2 * +"); // Equivalente a (10.5 / 5) + (3.5 * 2)
    return 0;
}
```

Salida del programa:

```
The first result is: 30
The second result is: Error: Division by Zero
The third result is: Error: Unknown command: ^
The fourth result is: 9.1
```

```
#ifndef POLISH_NOTATION_SIMPLEREVERSEPOLISHCALCULATOR_H
#define POLISH_NOTATION_SIMPLEREVERSEPOLISHCALCULATOR_H

#include <iostream>
#include <string>
#include <vector>
#include <utility>
// Tipos Genericos
enum class TermType
{
    Number = '0',
    Sum = '+',
    Rest = '-',
    Multiplication = '*',
    Division = '/',
    Empty = '\0'
};

typedef double Number;
typedef std::string Text;
typedef char Character;
typedef void Void;
typedef std::pair<TermType, Text> Token;
typedef std::vector<double> Stack;

// Funciones
Token get_token(Text&); // Retorna los token de una expresión
Void push(Stack&, Number); // Push en un Stack
Number pop(Stack&); // Pop de un Stack
Void simple_reverse_polish_calculator(Text); // Funcion Solicitada

#endif //POLISH_NOTATION_SIMPLEREVERSEPOLISHCALCULATOR_H
```

```
#include "SimpleReversePolishCalculator.h"
```

```
Void push(Stack& s, Number n) {  
    s.push_back(n);  
}
```

```
Number pop(Stack& s) {  
    auto result = s.back();  
    s.pop_back();  
    return result;  
}
```

```
Token get_token(Text& expr) {  
    Text term = Text();  
    size_t i = 0;  
    Text rest;
```

```
    // Limpiando de posibles espacios iniciales  
    if (isblank(expr[i]))  
        while (isblank(expr[++i]));
```

```
    // Evaluando si expresión queda vacía  
    if (i >= expr.size()) {  
        return {TermType::Empty, term};  
    }
```

```
    // Evaluando si termino no es numérico  
    if (!isdigit(expr[i]) && expr[i] != '.') {  
        char c = expr[i++];  
        while (i < expr.size())  
            rest += expr[i++];  
        expr = rest;  
        return {TermType(c), term};  
    }
```

```
    // Evaluando si termino es numérico  
    if (isdigit(expr[i]))  
        while (isdigit(expr[i]))  
            term += expr[i++];
```

```
    if (expr[i] == '.') {  
        term += expr[i++];  
        while (isdigit(expr[i]))  
            term += expr[i++];  
    }
```

```
    // Actualizando expresión con resto  
    while (i < expr.size())  
        rest += expr[i++];  
    expr = rest;
```

```
    // Retornando token numérico  
    return {TermType::Number, term};  
}
```

```

Void simple_reverse_polish_calculator(Text expression) {

    // Variables requeridas
    Token token;
    Number oper2;
    Stack stack;

    // Recorriendo expresiones
    while ((token = get_token(expression)).first != TermType::Empty)
    {
        switch (token.first) {

            // Push numero en stack
            case TermType::Number:
                push(stack, stod(token.second));
                break;

            // Pop 2 números y Push Suma de Los 2 números
            case TermType::Sum:
                push(stack, pop(stack) + pop(stack));
                break;

            // Pop 2 numeros y Push Multiplication
            // de Los 2 números
            case TermType::Multiplication:
                push(stack, pop(stack) * pop(stack));
                break;

            // Pop 2 números y Push Resta de Los 2 números
            case TermType::Rest:
                push(stack, -pop(stack) + pop(stack));
                break;
        }
    }
}

```

```

        // Pop 2 números y Push Resta de Los 2 números
        case TermType::Rest:
            push(stack, -pop(stack) + pop(stack));
            break;

        // Pop 2 numeros y Push Division si segundo
        // numero no es CERO
        case TermType::Division:
            oper2 = pop(stack);
            if (oper2 != 0)
                push(stack, pop(stack) / oper2);
            else {
                std::cout << "Error: Division by Zero\n";
                return;
            }
            break;

        // Si tipo de término es desconocido envía
        // mensaje
        default:
            std::cout << "Error: Unknown command: " <<
                char(token.first) << std::endl;
            return;
        }
    }
    // Mostrando resultado
    std::cout << pop(stack) << std::endl;
}

```

2

Unidad 2: Funciones y recursividad

- Recursividad.

La recursividad ocurre cuando

Para hallar el resultado una función se invoca a si misma.



El algoritmo para hallar el factorial de un número es un ejemplo clásico del uso de recursividad.

$$\text{fact}(7) = 7*6*5*4*3*2*1 \quad \text{ó} \quad \text{fact}(7) = 7 * \text{fact}(6)$$

$$\text{fact}(9) = 9*8*7*6*5*4*3*2*1 \quad \text{ó} \quad \text{fact}(8) = 8 * \text{fact}(7)$$

$$\text{fact}(14) = 14 * \text{fact}(13)$$

Generalizando:

$$\text{fact}(n) = n * \text{fact}(n-1)$$

Veamos cómo funciona la recursividad, hallando el factorial de 5

```
fact(5) = 5 * fact(4)
```

Recursión:

`fact(5) = 5 * fact(4)`


`4 * fact(3)`

Recursión:

`fact(5) = 5 * fact(4)`



`4 * fact(3)`



`3 * fact(2)`

Recursión:

`fact(5) = 5 * fact(4)`


`4 * fact(3)`


`3 * fact(2)`


`2 * fact(1)`

Recursión:

`fact(5) = 5 * fact(4)`


`4 * fact(3)`


`3 * fact(2)`


`2 * fact(1)`


1

Recursión:

fact(5) = 5 * fact(4)

4 * fact(3)

3 * fact(2)

2 * fact(1)

1

Recursión:

fact(5) = 5 * fact(4)

 **4 * fact(3)**

 **3 * fact(2)**

 **2 * 1**

Recursión:

$\text{fact}(5) = 5 * \text{fact}(4)$
 ↓
 4 * $\text{fact}(3)$
 ↑
 3 * 2

Recursión:

`fact(5) = 5 * fact(4)`
 ↑
 4 * 6

Recursión:

fact(5) = 5 * 24

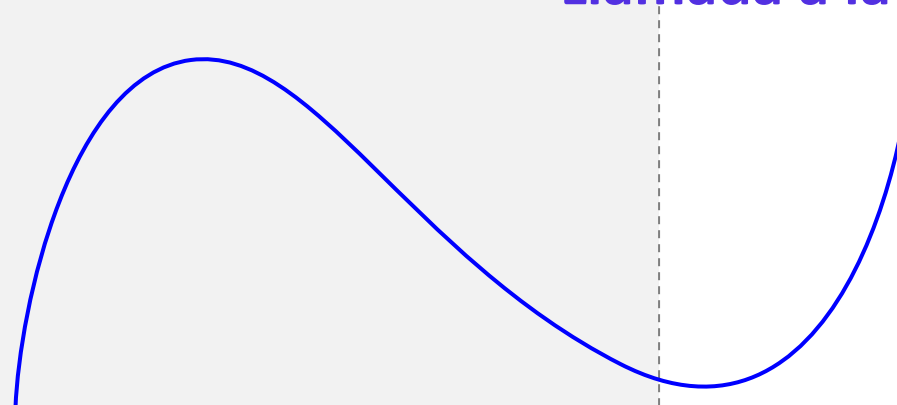
Recursión:

```
fact(5) = 120
```

Recursión:

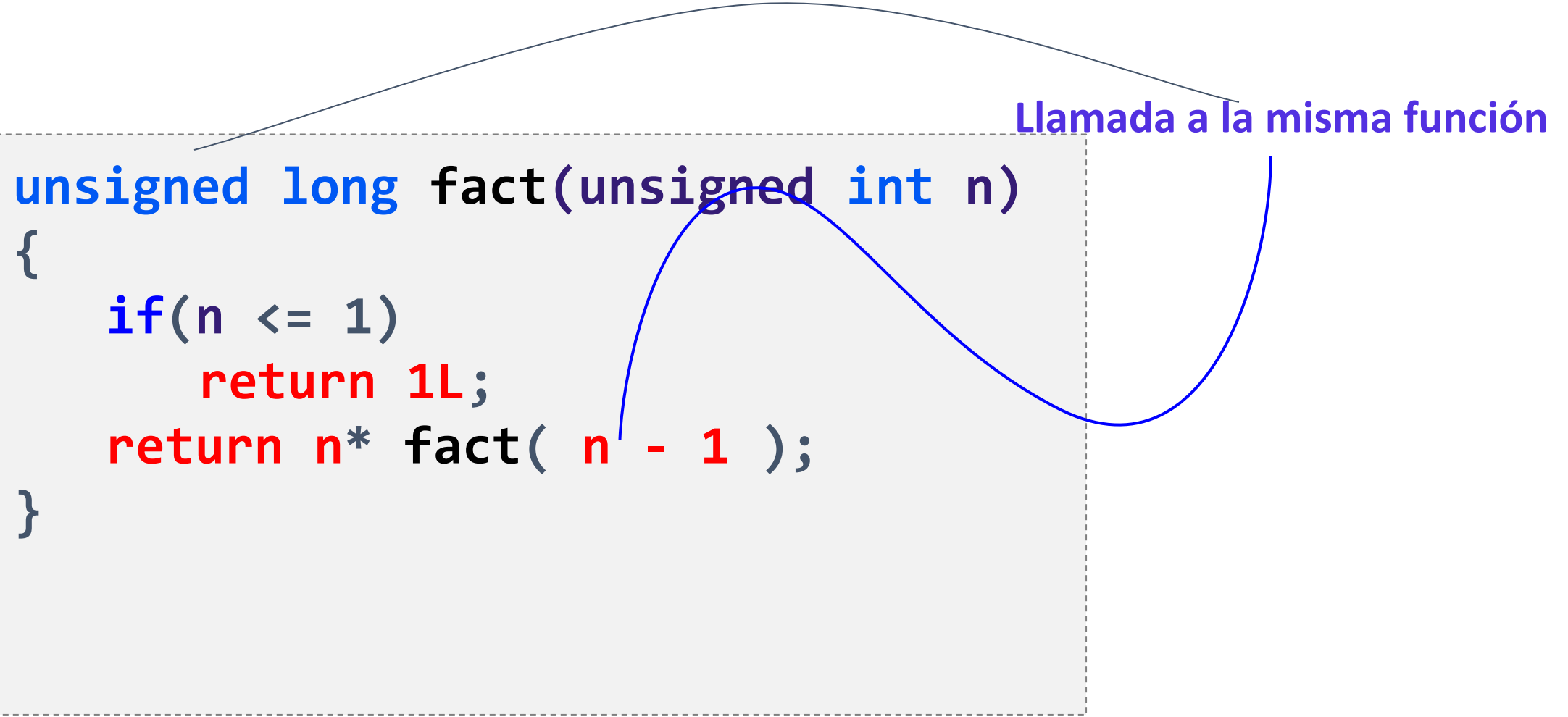
```
int fact(int n)
{
    if(n <= 1)
        return 1;
    else
        return n* fact( n - 1 );
}
```

Llamada a la misma función



Recursión:

```
unsigned long fact(unsigned int n)
{
    if(n <= 1)
        return 1L;
    return n* fact( n - 1 );
}
```



Llamada a la misma función

Ejemplo 6: recursión

Factorial - Iterativo

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

unsigned long factorial(unsigned int n)
{unsigned long f;

    f=1;
    for(int i=2;i<=n; i++)
        f*=i;
    return f;
}

int main()
{
    unsigned int n;
    cout << "Numero: ";
    cin >> n;
    cout << "Factorial ("<< n <<") = ";
    cout << factorial(n);
    return 0;
}
```

Factorial - Recursivo

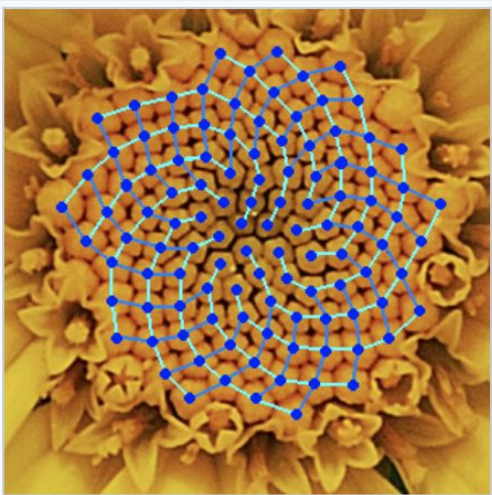
```
#include <iostream>
using namespace std;

unsigned long factorial(unsigned int n)
{
    if (n<=1)
        return 1L;
    return (n * factorial(n-1));
}

int main()
{
    unsigned int n;
    cout << "Numero: ";
    cin >> n;
    cout << "Factorial ("<< n <<") = ";
    cout << factorial(n);
    return 0;
}
```

Serie de Fibonacci:

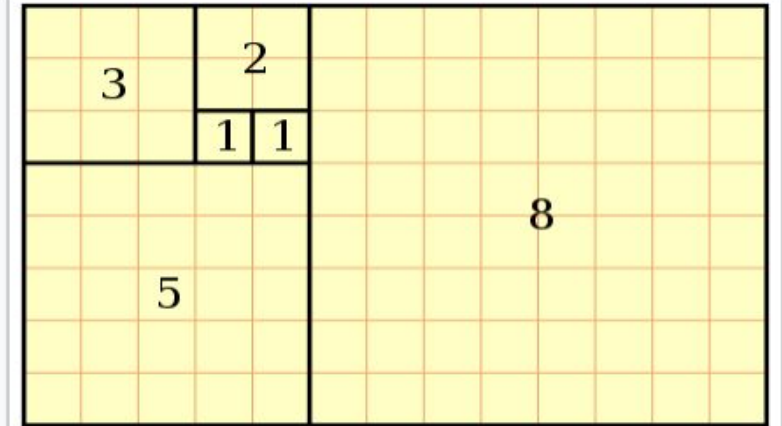
0 1 1 2 3 5 8 13 21 34 55 89 144 233 ...



Botón de [Camomila amarilla](#) mostrando la ordenación en espirales de módulos 21 (color azul) y 13 (color cian). Este tipo de arrollamientos utilizando números consecutivos de Fibonacci aparecen en una gran variedad de plantas.



Espiral de Fibonacci en la sección de la concha de un [nautilus](#).



Al construir bloques cuya longitud de lado sean números de Fibonacci se obtiene un dibujo que se asemeja al [rectángulo áureo](#) (véase [Número áureo](#)).

Serie de Fibonacci:

Los números de Fibonacci quedan definidos por la ecuación:

$$f_n = f_{n-1} + f_{n-2}$$

partiendo de dos primeros valores predeterminados:

$$f_0 = 0$$

$$f_1 = 1$$

se obtienen los siguientes números:

- $f_2 = 1$
- $f_3 = 2$
- $f_4 = 3$
- $f_5 = 5$
- $f_6 = 8$
- $f_7 = 13$
- $f_8 = 21$

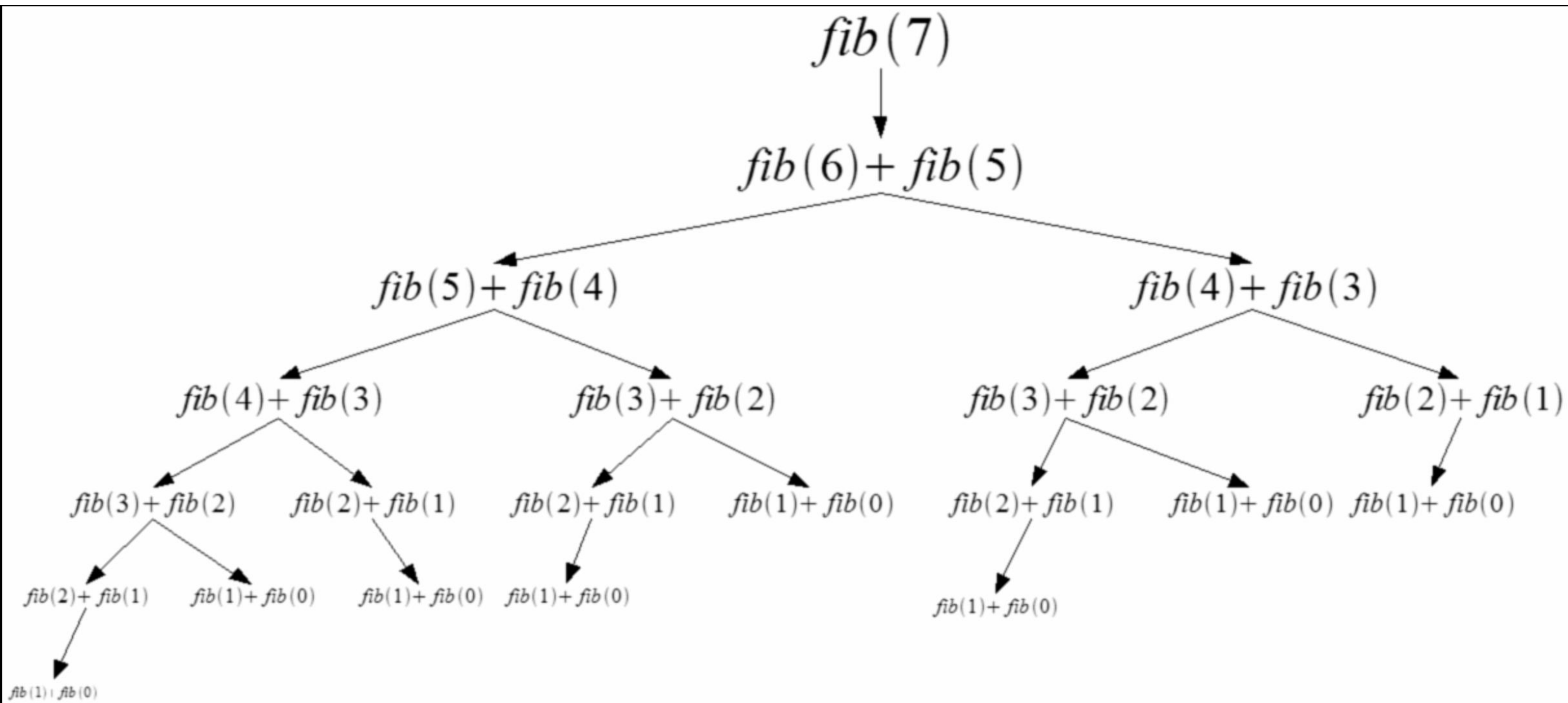
para $n = 2, 3, 4, 5, \dots$

$$f(n) = f(n-1) + f(n-2)$$

Partiendo de:

$$f(0) = 0$$

$$f(1) = 1$$



Ejemplo 6: recursión

Fibonacci - Iterativo

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

long int fibonacci (long int  n)
{
    if(n<2)
        return n;
    long int a=0,b=1, c;
    for(int i=2;i<=n;i++)
    {
        c = a+b;
        a = b;
        b = c;
    }
    return c;
}

int main()
{ long int n;
  cout <<"Termino : ";
  cin >> n;
  cout << "Fibonacci ("<< n <<") = ";
  cout << fibonacci(n);
  return 0;
}
```

Fibonacci - Recursivo

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

long int fibonacci (long int  n)
{
    if(n<2)
        return n;
    return( fibonacci(n-1) + fibonacci(n-2));
}

int main()
{ long int n;
  cout <<"Termino : ";
  cin >> n;
  cout << "Fibonacci ("<< n <<") =";
  cout << fibonacci(n);
  return 0;
}
```

3

Unidad 2: Funciones y recursividad

- Funciones Lambda

Ejemplo 7:

Halla la suma de los 10 primeros números naturales, utiliza una función Lambda.

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

void SumaNumero(int &s,int x)
{
    s+=x;
}

int main()
{
    int sum = 0;
    for (int i = 0; i < 10; i++) {
        SumaNumero(sum,i);
    }
    cout<<"La suma de los 10 primeros numeros : "<<sum<<"\n";
    return 0;
}
```

Ejemplo :

Halla la suma de los 10 primeros números naturales, utiliza una función Lambda.

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

int main() {
    //[capture list](parameter list) {function body}

    int sum = 0;
    for (int i = 0; i < 10; i++) {
        [&sum](int x){sum+=x;}(i);
    }
    cout<<"La suma de los 10 primeros numeros : "<<sum<<"\n";

    [](){cout<<"Hello World!"<<endl;}();

    return 0;
}
```

The following example uses a [lambda function](#) to increment all of the elements of a vector and then uses an overloaded operator() in a functor to compute their sum. Note that to compute the sum, it is recommended to use the dedicated algorithm `std::accumulate`.

Run this code

```
#include <vector>
#include <algorithm>
#include <iostream>

struct Sum
{
    Sum(): sum{0} { }
    void operator()(int n) { sum += n; }
    int sum;
};

int main()
{
    std::vector<int> nums{3, 4, 2, 8, 15, 267};

    auto print = [](const int& n) { std::cout << " " << n; };

    std::cout << "before:";
    std::for_each(nums.begin(), nums.end(), print);
    std::cout << '\n';

    std::for_each(nums.begin(), nums.end(), [](int &n){ n++; });

    // calls Sum::operator() for each number
    Sum s = std::for_each(nums.begin(), nums.end(), Sum());

    std::cout << "after: ";
    std::for_each(nums.begin(), nums.end(), print);
    std::cout << '\n';
    std::cout << "sum: " << s.sum << '\n';
}
```

Output:

```
before: 3 4 2 8 15 267
after:  4 5 3 9 16 268
sum: 305
```

¿Qué crees que hace el siguiente programa?

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

int queNombreTieneEstaFuncion(vector<int> v);

int main()
{vector<int> vec ={1,2,3,4,5,45,7,4,28,30,21,11};

  cout << "El resultado es : " << queNombreTieneEstaFuncion(vec);
  return 0;
}

int queNombreTieneEstaFuncion(vector<int> v) {
  int elemento;
  if (v.size() == 1) {
    return v[0];
  } else {
    elemento = queNombreTieneEstaFuncion(vector<int>(v.begin() + 1, v.end()));
    return elemento > v[0] ? elemento : v[0];
  }
}
```


Resumen

- 1. Las funciones nos permiten realizar una tarea específica.**
- 2. Las funciones reciben valores en los parámetros de entrada y retornan un valor.**
- 3. La recursividad es una característica que permite a un subprograma se invoque así mismo.**
- 4. Cualquier proceso iterativo puede expresarse en forma recursiva y viceversa.**

**¡Nos vemos en la
siguiente clase!**

