SUBPROGRAMAS
FUNCIONES

INTRODUCCIÓN

En la mayoría de los casos, un determinado problema complejo lo podemos (y debemos) dividir en problemas más sencillos. (Módulos o Subprogramas)

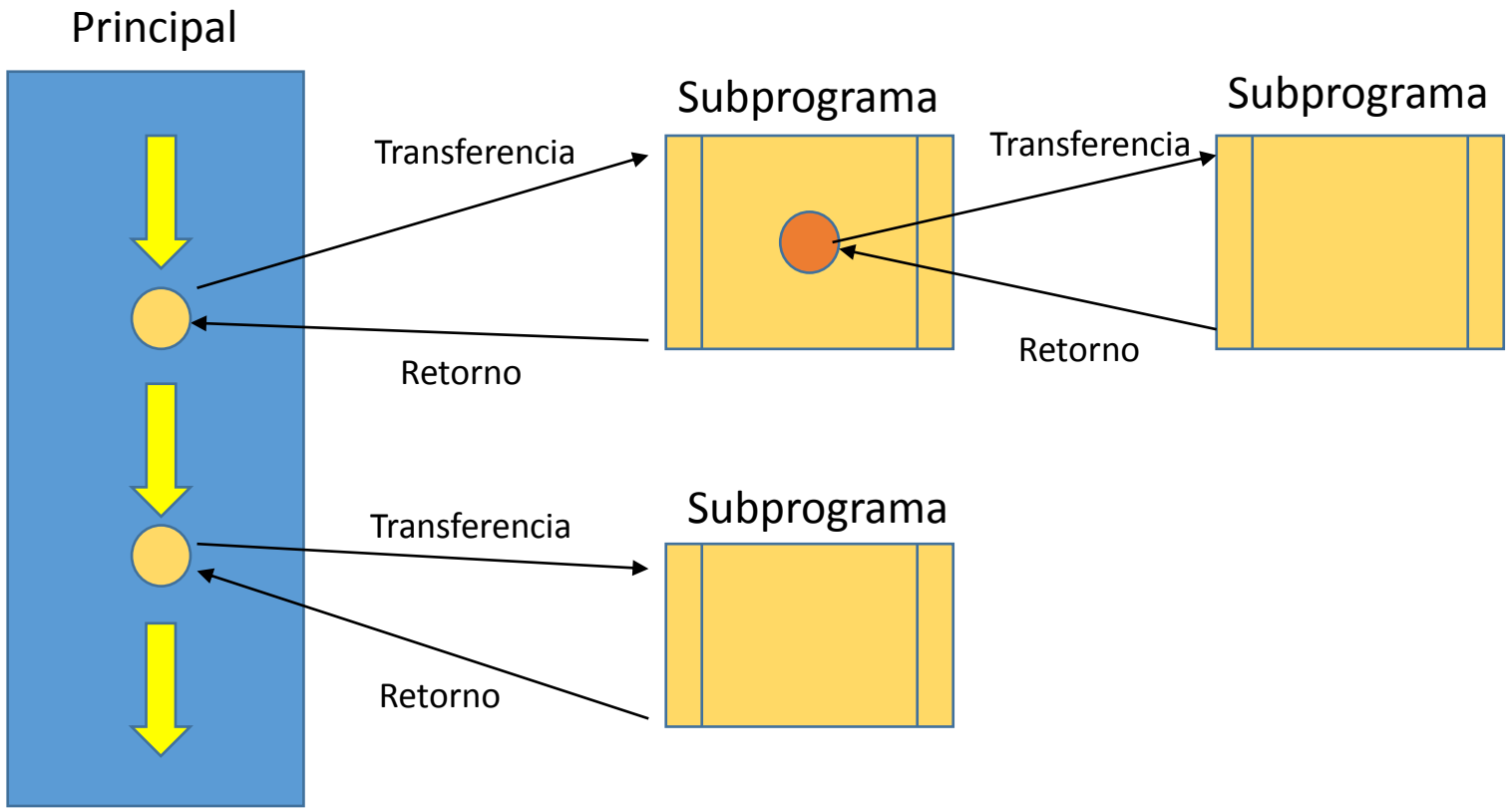
TÉCNICA DE DISEÑO TOP DOWN

- ✓ Se tratará de descomponer el problema original en partes.
- ✓ Se pueden codificar de forma independiente e incluso por diferentes personas.
- ✓ El problema final queda resuelto y estructurado en forma de módulos, lo que hace más sencilla su lectura y mantenimiento.

Ventajas:

- Ahorro de espacio.
- Más fácil entender el algoritmo.
- Más fácil de codificar y mantener.
- Se evita repetir código

Un **subprograma** es una serie de instrucciones escritas independientemente del programa principal. Está ligado al programa principal mediante un proceso de **transferencia/retorno.**



C++ es un lenguaje modular, y por ésta razón, se puede dividir en varios módulos, cada uno de los cuales realiza una tarea determinada. Cada módulo es un subprograma llamado **FUNCIÓN**. Una función "especial" y que se usa siempre es la función main().

Estructura de una función

```
<tipo_resultado> <nombre_función> ( lista_de_parámetros )
{
        cuerpo_de_la_función ;

        return <expresión>;
}
```

<tipo_resultado> tipo de dato que devuelve la función.

lista_de_parámetros cada parámetro <tipo><identificador>. Son valores que se pasan a la función.

return <expresión> punto de retorno de la función. Devuelve un valor del tipo declarado en la función.

El tipo de datos de retorno puede ser tipo void. En este caso la función no devuelve ningún valor y se denominan *PROCEDIMIENTOS*.

Puede tener cualquier número de sentencias return, pero al menos debe de haber una.

EJEMPLOS

//funcion01.cpp

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

int suma(int x, int y); //prototipo de la función

int main () //función main
{
   int a,b;
   a=5;
   b=13;
            cout<<suma(a,b);
   return 0;
}

int suma(int x, int y) //función suma
{
            return x+y;
}
```

//funcion02.cpp

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

void mostrar_caracter(char c); //prototipo de procedimiento

int main () //función main
{
   int i=10;
   for (int i=1;i<10;i++)
                        mostrar_caracter('x');
   return 0;
}

void mostrar_caracter(char c) //Procedimiento
{
            cout<<c;
            return;
}
```

Las funciones trabajan con dos tipos de datos:

1. Variables locales: declaradas en el cuerpo de la función. Estas variables solo son conocidas dentro de la función y se crean y se destruyen con la función.

2. Lista de Parámetros: Permiten la comunicación de la función con el resto del programa mediante transferencia de datos.

C++ proporciona dos métodos para realizar ésta transferencia de datos a la función:

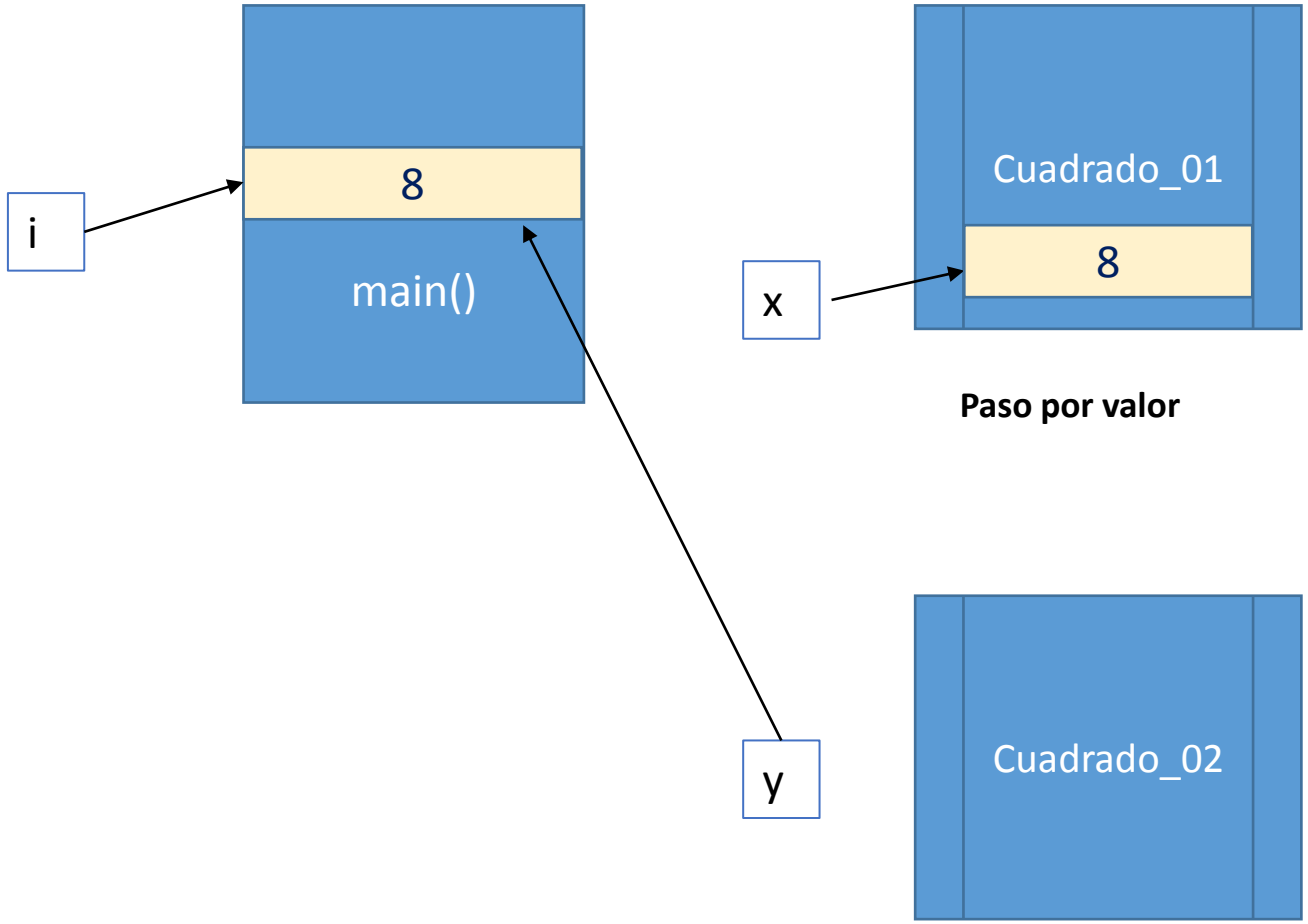
Paso de parámetros por valor. El compilador reserva memoria y hace copia para los parámetros de la función y se destruyen en el retorno. Un cambio de valor de estos parámetros **NO AFECTA** a los usados en la llamada. Por defecto el paso de parámetros es por valor *salvo en los tipos arrays*

Paso de parámetros por referencia. El compilador no reserva memoria para los parámetros de la función sino que usa la misma dirección de memoria de los parámetros en la llamada. Un cambio de valor de estos parámetros **AFECTA** a los usados en la llamada. Para este caso se utiliza el símbolo **&** en la declaración de los parámetros (int & x)

```
//funcion03.cpp
#include <iostream>
using namespace std;
int cuadrado_01(int x);
int cuadrado_02(int & x);

int main () //función main
{
   int i=8;
   cout<<cuadrado_01(i)<<endl;
   cout<<i<<endl;
   cout<<cuadrado_02(i)<<endl;
   cout<<i;

}
int cuadrado_01(int x) // Paso parámetros por valor
{
            return x*=x;
}
int cuadrado_02(int & y) //Paso parámetros por referencia
{
            return y*=y;
}
```



Los arrays se pasan siempre por referencia aunque no lleven &.

```
//funcion04.cpp
#include <iostream>
using namespace std;
void cuadrados_vector(int v[],int elementos);
void muestra_vector(int v[],int elementos);

int main ()
{
   int numeros[5]={0,1,2,3,4};
   muestra_vector(numeros,5);
   cuadrados_vector(numeros,5);
   muestra_vector(numeros,5); //Los elementos del vector han cambiado

   return 0;
}
void cuadrados_vector(int v[],int elementos)
{
                int i;
                for (i=0;i<elementos;i++)
                                v[i]*=v[i]; //El paso de parámetros es por referencia en los arrays
}
void muestra_vector(int v[],int elementos)
{
                int i;
                for (i=0;i<elementos;i++)
                                cout<<v[i]<<endl;
}
```

BIBLIOTECAS FUNCIONES EN C++ (Predefinidas)

Biblioteca <math.h>

Función	Sintaxis	Descripción
abs(i)	int abs(int i);	Devuelve el valor absoluto de i
acos(d)	double acos(double d);	Devuelve el arco coseno de d
asin(d)	double asin(double d);	Devuelve el arco seno de d
atan(d)	double atan(double d);	Devuelve la arco tangente de d.
ceil(d)	double ceil(double d);	Devuelve un valor redondeado por exceso al siguiente entero mayor
cos(d)	double cos(double d);	Devuelve el coseno de d
cosh(d)	double cos(double d);	Devuelve el coseno hiperbólico de d
exp(d)	double exp(double d);	Eleve *e* a la potencia
fabs(d)	double fabs(double d);	Devuelve el valor absoluto de d
floor(d)	double floor(double d);	Devuelve un valor redondeado por defecto al entero menor mas cercano
fmod(d1, d2)	double fmod(double d1, double d2);	Devuelve el resto de d1/d2 (con el mismo signo que d1)
labs(l)	long int labs(long int l);	Devuelve el valor absoluto de 1
log(d)	double log(double d);	Devuelve el logaritmo natural de d
log10(d)	double log10(double d);	Devuelve el logaritmno (en base 10) de d
pow(d1, d2)	double pow(double d1, double d2);	Devuelve d1 elevado a la potencia d2
sin(d)	double sin(double d);	Devuelve el seno de d
sinh(d)	double sinh(double d);	Devuelve el seno hiperbolico de d
sqrt(d)	double sqrt(double d);	Devuelve la raiz cuadrada de d
tan(d)	double tan(double d);	Devuelve la tangente de d
tanh(d)	double tanh(double d);	Devuelve la tangente hiperbolica de d

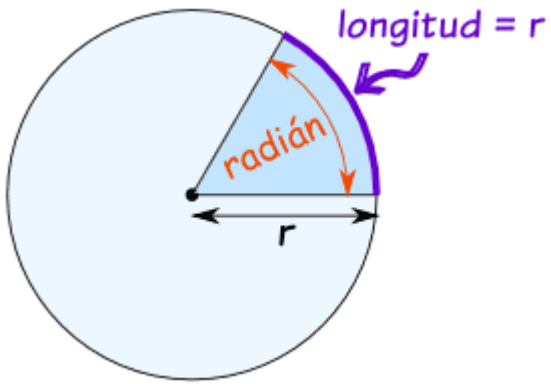
```
//funcion05.cpp   Estudio de una ecuación de segundo grado
#include <iostream>
#include <math.h>
using namespace std;
void entrada_coeficientes(double & a, double & b, double & c);
double discriminante(double a, double b, double c);
double solucion_1(double a, double b, double c);
double solucion_2(double a, double b, double c);

int main ()
{
   double a,b,c;
   entrada_coeficientes(a,b,c);
   if (discriminante(a,b,c) < 0)
                cout<<"no hay solución real";
   else if (discriminante(a,b,c) == 0)
                cout<<"solución doble"<<solucion_1(a,b,c);
   else  {
                cout<<"Primera solucion:"<<solucion_1(a,b,c)<<endl;
                cout<<"Segunda solucion:"<<solucion_2(a,b,c);
         }
}
```

```
void entrada_coeficientes(double & a, double & b, double & c)
{
   cout<<endl<<"valor de a=";
   cin>>a;
   cout<<endl<<"valor de b=";
   cin>>b;
   cout<<endl<<"valor de c=";
   cin>>c;
}
double discriminante(double a, double b, double c)
{
                return pow(b,2)-4*a*c;
}
double solucion_1(double a, double b, double c)
{
                return (-b+sqrt(discriminante(a,b,c)))/(2*a);
}
double solucion_2(double a, double b, double c)
{
                return (-b-sqrt(discriminante(a,b,c)))/(2*a);
}
```

```
//funcion06.cpp  Trigonométricas
#include <iostream>
#include <math.h>
using namespace std;
const double pi=3.141592;
int main ()
{
   double angulo, radianes;
   cout<<"Valor del ángulo en grados:";
   cin >> angulo;
   radianes = angulo*pi/180;
   cout<<"Seno de "<<angulo<<"="<<sin(radianes)<<endl;
   cout<<"Coseno de "<<angulo<<"="<<cos(radianes)<<endl;
   cout<<"Tangente de "<<angulo<<"="<<tan(radianes)<<endl;

}
```



```
//tipostruct02.cpp
#include <iostream>
using namespace std ;
struct complejo
{
        float real ;
        float imaginaria ;
} ;

void leer_complejo (complejo & z)
{
        complejo z1 ;
        cout<<"Parte real:"; cin>>z.real;
        cout<<"Parte imaginaria:"; cin>>z.imaginaria;
}

void mostrar_complejo (complejo z)
{
        cout<<"("<<z.real<<","<<z.imaginaria<<"i)"<<endl;
}

complejo suma_complejos(complejo a, complejo b)
{
        complejo z;
        z.real=a.real+b.real;
        z.imaginaria=a.imaginaria+b.imaginaria;
        return z;
}

complejo producto_complejos(complejo a, complejo b)
{
        complejo z;
        z.real=a.real*b.real - a.imaginaria*b.imaginaria;
        z.imaginaria=a.real*b.imaginaria+a.imaginaria*b.real;
        return z;
}

int main ()
{
        complejo z1, z2 ;
        leer_complejo(z1); leer_complejo(z2);
        mostrar_complejo(z1); mostrar_complejo(z2);
        mostrar_complejo(suma_complejos(z1,z2));
        mostrar_complejo(producto_complejos(z1,z2));
}
```

Ordenación Burbuja

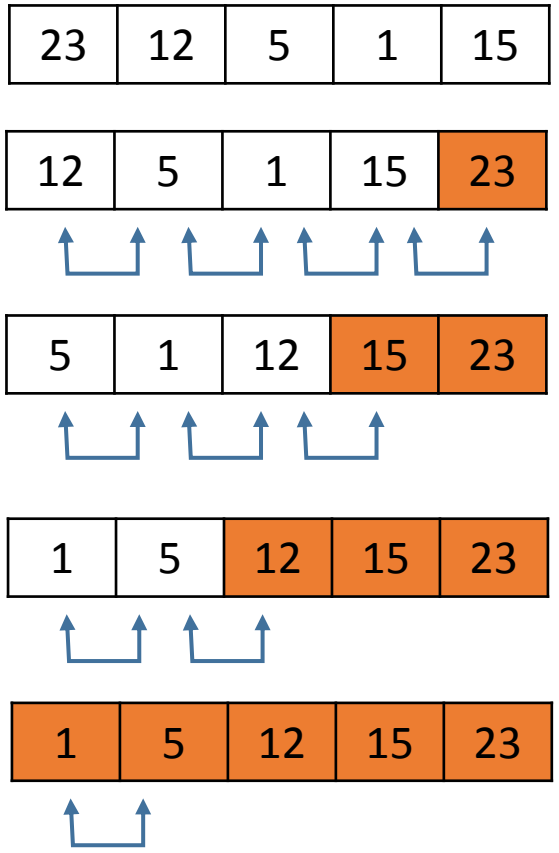


Tabla incial de 5 elementos

1º Iteración. (j=1)
Desde 1º al último-1.

2º Iteración. (j=2)
Desde 1º al último-2.

3º Iteración. (j=3)
Desde 1º al último-3.

4º Iteración. (j=4)
Desde 1º al último-4.

Iterar tantas veces como elementos-1 tenga el vector. En cada iteración j, se compara el elemento i con i+1 desde el **primero** hasta el **último-j** quedando ordenado el elemento **último-j+1.**

```
//Método de ordenación de la burbuja
#include <iostream>
using namespace std;
void entrada_vector(int v[],int elementos);
void muestra_vector(int v[],int elementos);
void ordenacion_burbuja(int v[],int elementos);

int main()
{
	int numeros[10];
	cout<<"Entrada de datos";
	entrada_vector(numeros,10);
	ordenacion_burbuja(numeros,10);
	cout<<"Dastos ordenados";
	muestra_vector(numeros,10);
	return 0;
}

void entrada_vector(int v[],int elementos)
{
	int i;
	for (i=0;i<elementos;i++)
	{
		cout<<"Elemento "<<i<<"=";
		cin >> v[i];
	}
}
```

```
void muestra_vector(int v[],int elementos)
{
	for (int i=0;i<elementos;i++)
		cout<<v[i]<<endl;
}


void ordenacion_burbuja(int v[],int elementos)
{
	int i,j,aux;
	for ( i=1;i<=elementos-1;i++)
		for (j =0; j<elementos-i;j++)
			if (v[j]>v[j+1])
			{
				aux=v[j];
				v[j]=v[j+1];
				v[j+1]=aux;
			}
}
```

FUNCIONES RECURSIVAS

Se dice que una función es recursiva cuando se define en función de si misma.

Para definir la función:

1. Definir valor devuelto en el caso base (Evitar bucles infinitos)
2. Definir el caso recursivo (Nos debe llevar al caso base)

Función factorial. Definida para números naturales.

Factorial(0)=1
Factorial(1)=1
Factorial (2)=2*1=2
Factorial (3)=3*2*1=6
Factorial (4)=4*Factorial(3)=24

Caso Base

Factorial(n)=Factorial(n-1)*n

Caso recursivo

```
//Función factorial recursivo
#include <iostream>
using namespace std;
long double factorial (int n);

int main(void)
{
   int a;
   cout<<"Ingrese el número=";
   cin >> a;
   if (a >= 0)
            cout<<"El factorial es "<<factorial(a);
   else
            cout<<"No definido para números negativos";

   return 0;
}

long double factorial (int n)
{
            if (n==0)
                        return 1;
            else
                        return n*factorial(n-1);
}
```