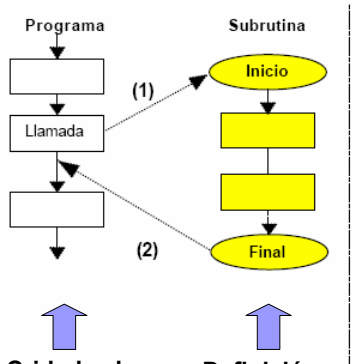
# Funcions en C++

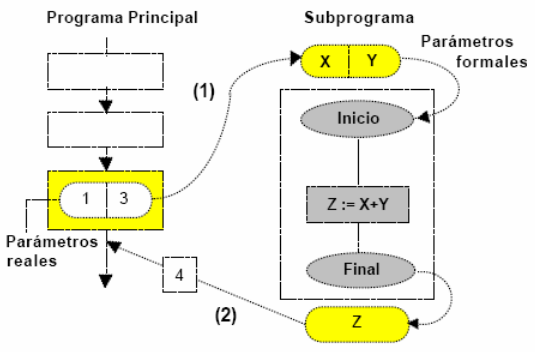


Un subprograma hace el papel de un programa. Puede tener una sección de declaraciones (variables, constantes, etc…) y posee también unos datos de entrada y de salida. Esto permite, como ya veremos, que el subprograma sea totalmente independiente del programa principal.




**Cridada al subpr.**

**Definició del subpr.**




**Subprogrames en C++** ⋯⋯▸ **Funcions**

Una funció es un subprograma que sempre te un paràmetre d'eixida (Ex.: cos(x), pow(2,3) ,.. ).

```
Tipo nom_func (parámetros)      // → Cabecera de la función
{
  Declaracions                  // → Variables, …
    …
  Instruccions                  // → Cos de la funció
    …
   return valor;                // → Valor de tornada
}
```

***Tipo*** es el tipus de la variable d'eixida,
**nom_func** es un identificador que representa el nom de la funció,
**parámetros** es un conjunt de paràmetros separats per comes, on cadascun es declara com una variable normal amb el seu tipus.
Mitjançant la instrucció **return** enviem el valor que tornarà la funció al acabar.

• Cuan fem una cridada a una funció, lo primer que es fa es una asignació dels **paràmetres reals** (els del pp) als **formals** (els de la funció) → **operació de Paso de paràmetres (valor | referencia)**, i després comencen a ejecutar-se les instruccions de la funció …
• Si volem una funció que no torne cap valor, la declarem de tipo ***void***.



```
#include <iostream.h>

//Prototipos de funciones
int recta(int x, int m, int b);

int main()
{
     int m, b; //coeficientes de la ecuacion de la recta
     int r0,r1; //puntos extremos del intervalo
     int x, y; //para el bucle for

     cout << "Introducir coeficientes de la recta: y=mx+b\n";
     cin >> m >> b;


     cout << "Recta: "<< "y=" << m << "x + " << b ;
     cout << "\tRango: [" << r0 <<" " << r1 << "]";
     //Calculo los puntos de la recta
     for (x = r0 ; x <= r1; x++)
     {
       //Calculo valores de y para cada valor de x

       y = recta ( x, m, b);

       //Muestro resultado
       cout << " Valor de x: " << x << ", Valor de y: " << y <<endl;

     }

     return 0;
}
```

Llamada a la función

```
//Función que calcula la coordenada y de la recta (mx + b)
int recta(int x, int m, int b)
{
    int y;

     y = m*x + b;

     return y;
}
```

**Paràmetres per valor**

- **La funció es farà una copia de cadascun d'ells (ej.: int x, int m, int b)**
- **L'àmbit d'aquestes variables es local (podem gastar el mateix nom dins de la funció sense *efectes laterals*, ja que modifiquem una copia)**
- **Per tant, es consideren paràmetres d'entrada a la funció**



```
#include <iostream.h>

float sumar ( float a, float b);
float restar ( float a, float b);
float multiplicar ( float a, float b);
Float dividir (float a, float b);
int main()
{
 float n1, n2, res;
 int opcion;

do
{
  cout << " Elige opcion:\n";
  cout << " 1. Sumar" << endl;
  cout << " 2. Restar" << endl;
  cout << " 3. Multiplicar" << endl;
  cout << " 4. Dividir" << endl;
  cout << " 0. Salir" << endl;

  cin >> opcion;
```

Llamadas a funciones

```
switch ( opcion )
  {

   case 1:
          cout << "Introduce los numeros a sumar" << endl;
          cin >> n1 >> n2;

          res = sumar(n1,n2);

          cout << n1 << " + " << n2 << " = " << res << endl;
          break;
   case 2: cout << "Introduce los numeros a restar" << endl;
          cin >> n1 >> n2;

          res = restar(n1,n2);

          cout << n1 << " - " << n2 << " = " << res << endl;
          break;
   case 3:
          cout << "Introduce los numeros a multiplicar" << endl;
          cin >> n1 >> n2;

          res = multiplicar(n1,n2)

          cout << n1 << " * " << n2 << " = " << res << endl;
          break;
   case 4: cout << "Introduce los numeros a dividir" << endl;
          cin >> n1 >> n2;

          res = dividir(n1,n2);

          cout << n1 << " / " << n2 << " = " << res << endl;
         break;
   }
}while( opcion != 0 );

 return 0;
```

**Exercici 7: Calculadora …**

```
float sumar ( float a, float b)
{
  int res; //Declaración de variables locales

  res = a + b;

  return res; //Valor que devuelve la función
}
```

```
float restar ( float a, float b)
{
  int res; //Declaración de variables locales

  res = a - b;

  return res; //Valor que devuelve la función
}
```

```
float multiplicar ( float a, float b)
{
  int res; //Declaración de variables locales

  res = a * b;

  return res; //Valor que devuelve la función
}
```

```
float dividir ( float a, float b)
{
  int res; //Declaración de variables locales

  res = a / b;

  return res; //Valor que devuelve la función
}
```



# Recordeu…. :

## funcions → problemes independents

Definir: Prototip, cridada, i la propia funció:

- tipus dels parametres: valor (int x) , referencia (int &x)
- Àmbit de les variables i paràmetres.
- Valor/s d'eixida (totes retornen un valor per defecte, pot ser *void*).

## Exercici 8: Escriure una funció que transforme un punt de coordenades polars a cartesianes

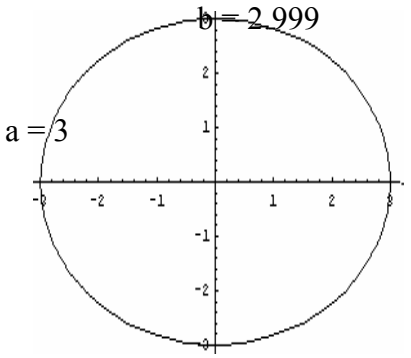
**Entrades:** Un punt amb coordenades polares → **mòdul, àngul**
**Eixides:** Coordenades **(x,y)**
**Análisis :** ¿sabem la tranformació que cal utilitzar?



```
#include <iostream.h>
#include <math.h>

//Prototipo de la función
void PolaresACartesianas(float modulo, float angulo, float &x, float &y);

int main()
{
  float m,ang;
  float posx, posy;

  cout << "Introducir el modulo y el angulo del punto en coordenadas polares:\n";

  cin >> m >> ang;

  //Calculo la posicion x, y

  PolaresACartesianas(m, ang, posx, posy);

  cout << "Las coordenas cartesianas correspondientes a: " <<m << ',' << ang << endl;
  cout << "("<< posx << "," << posy<<")"<< endl;

  return;

}


void PolaresACartesianas(float modulo, float angulo, float &x, float &y)
{
   x = modulo*cos(angulo);
   y = modulo*sin(angulo);

   return;

}
```

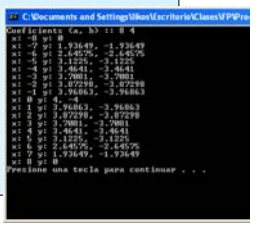
- Exercici 7: Mostrar els punts (coordenades *x,y)* de l'elipse centrada en el (0,0) (demanant a i b pel teclat). Fer el mateix amb les circumferències (r = a, r = b). Definir tres funcions que es cridaran desde el programa principal.

$(x^2/a^2) + (y^2/b^2) = 1$

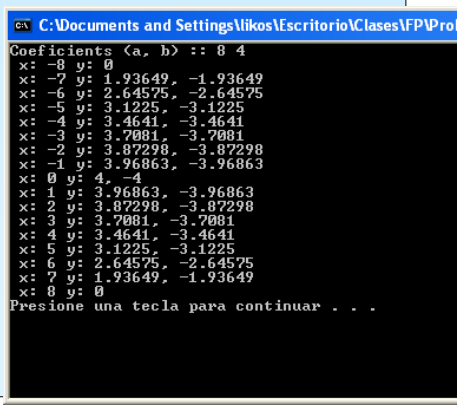





```
#include <iostream>
#include <stdlib.h>

using namespace std;
void elipse(float x, float a, float b, float &y1, float &y2);

int main(int argc, char *argv[])
{
  float a, b;
  float y1, y2;

   cout << "Coeficients (a, b) :: " ;
   cin >> a >> b;

   for (float x=-a; x<=a ; x+=1)
   {
     elipse(x, a, b, y1, y2);
     cout << " x: " << x << " " << "y: " << y1 << " y2: " << y2 << endl;
     // .. Circunferencia …
   }

   system("PAUSE");
   return 0;

}
```



```
void elipse(float x, float a, float b, float &y1, float &y2)
{
  float b2a2, raiz;

  b2a2 = (b*b)/(a*a);

  raiz = (b*b) - b2a2*(x*x);
  if( raiz > 0 )
  {
      y1 = sqrt(raiz) ;
      y2 = y1 * -1;
  }
  else
  {
     y1 = 0;
     y2 = 0;
  }
}
```






- Exercici 9:Escriure una funció que genere punts 2D aleatoris. Introduir la *llavor* (semilla) y el rang per teclat.

**La instrucción srand** (*semilla*) inicializa motor de números pseudo-aleatorios en C++. **srand** usa el argumento recibido como **semilla** para una nueva secuencia de números pseudo-aleatorios, que serán retornados en llamadas posteriores a **rand()**. Si ***srand*** es llamada con el mismo valor semilla, la secuencia de números pseudo-aleatorios será la misma!. Por tanto, si **rand()** es llamada antes de que se haya inicializado (cualquier llamada a *srand)*, la misma secuencia será generada. Para evitar esto, se suele inicializar con valores como la hora de ejcución, de forma que cada secuencia sea diferente.

```
int main()
{
 int x,y, maxRango, minRango, n, semilla, i;

   cout << "Semilla: ";
   cin  >> semilla;
   srand(semilla);

   cout << "Numero de puntos2D a generar";
   cin  >> n;

   do
   {
    cout << "Rango [min..max]: ";
    cin  >> minRango >> maxRango;
   } while(minRango >= maxRango);

   for(i=0; i<n; i++)
   {
     GeneraPunto2D(x, y, minRango, maxRango);
     cout << "Punto2D: (" << x << ", " << y << ")." << endl;
   }


   system("PAUSE");
   return 0;
}
```

**srand** (*semilla*) usa el argumento como una **semilla** para una nueva secuencia de números pseudo-aleatorios, que serán retornados en llamadas posteriores a **rand( )**. Si ***srand*** es llamada con el mismo valor semilla, la secuencia de números pseudo-aleatorios será repetida. Si **rand** es llamada antes de que se haya hecho cualquier llamada a *srand*, la misma secuencia será generada. (inicializar con valores del sistema, como la hora, etc)

```
int Aleatorio(int ini, int fin)
{
   int aleat;
   aleat = ini + rand() % int(fin - ini + 1); // para incluir fin
   return aleat;
}

void GeneraPunto2D(int& x, int& y, int rmin, int rmax)
{
  x = Aleatorio(rmin, rmax);
  y = Aleatorio(rmin, rmax);
}
```



# Funcions recursives

• Dins del codi d'una funció recursiva hi haurà una sentencia o expresió amb una cridada a la mateixa funció.

• Per a no generar cridades infinitas (donat que la funció es crida a si mateixa) necesitem considerar el següent:

• Una funció recursiva resoldrà un problema de **talla N**, considerant resolt el problema en la seua **talla N – 1**.

• Per tant, **els paràmetres de la funció han de variar** :

ej: factorial(N) = N * factorial (N-1);  // problema_de_talla_N = N *problema_de_talla_N-1

• Abans que es produeixi la recursió, deu aparèixer la **condició de parada** (cas base), que estarà relacionada amb algun dels paràmetres de la funció (ej: factorial(1) = 1). En aquest cas, no es produiran més cridades recursivas i es retornarà el valor corresponent**.** .

- Exercici 10: Programar una funció recursiva que calcule el sumatori i productori dels n primers sencers.



```
/************************************************
Funcion que calcula el sumatorio de forma recursiva
************************************************/
int Sumatorio(int n)
{
  int res;

  if (n == 1)
     res = 1;
  else
     res = n + sumatorio( n -1);

  return res;

}
/************************************************
Funcion que calcula el productorio de forma recursiva
************************************************/
int Productorio(int n)
{
   int res;

if ( n == 1)
     res = 1;
else
     res = n * Productorio ( n-1);

return res;
}
```

```
#include <iostream.h>

//Prototipo de la funcion

int Sumatorio(int n);
int Productorio(int n);
int main()
{
  int num;
  int suma, producto;

  cout <<"Este programa calcula el sumatorio  y productorio de  n numeros naturales\n";
  cout << "Utilizando una función recursiva. Introduce n:\n";

  cin >> n;

  suma = Sumatorio(num);

  producto = Productorio(num);

 cout << "El productorio es:" << producto;
 cout << "El sumatorio es:" << suma;

 return 0;
}
```

Llamada a la función



- Ejercicio 11: Realizar una función recursiva que calcule recursivamente la suma de dos números enteros utilizando solamente operaciones de incremento y decremento.

```
/*****************************************************************************
* Funcion que suma dos numero enteros
* Descripcion: Calcula el cociente y el resto de la division entera
* Parámetros:
* Nombre: E/S
*   a      E
*   b      E
*
* Valor devuelto:
* int (valor devuelto)
*****************************************************************************/
int Suma(int a, int b)
{
  if ( a == 0)
     res = b;
  else
     res = suma(a-1,b+1)

  return res;
}
```



- Ejercicio 12: *Recursividad indirecta*. Escribe una función que devuelva cierto si un número entero pasado como parámetro es **par** y falso en caso contrario.

```
#include <iostream.h>

// Prototipos
bool par(int n);
bool impar(int n);

int main()
{
    int a;

    cout << "Introduce un número entero : ";

    cin >> a ;
    cin.ignore();

    if(par(a))
      cout << " ... es par " << endl;
    else
      cout << " ... es impar " << endl;

    system("pause");

    return 0;
}

bool par(int n)
{
  if (n == 0)
    return true;
  else
    return impar(n-1);
}

bool impar(int n)
{
  if (n == 0)
    return false;
  else
    return par(n-1);
}
```