5.9 La recursivitat (I)

- Els algorismes que empren la recursivitat contenen funcions que es criden a sí mateixes.
- Normalment la recursivitat permet la construcció de programes complexes amb poques instruccions. Com a contrapartida, (sovint) costa d'entendre i (sovint) realitza un consum alt de memòria (fet que no passa amb un bucle). Això es deu a què la recursivitat en el C empra una pila en temps d'execució que, cada cop que la funció és cridada, emmagatzema les variables i els valors dels paràmetres de les funcions en execució. Només quan se surt de les funcions la pila es buida. Per tant, una funció que no tingui parada omplirà tota la pila fins que finalment produirà un error.
- Per tant, és imprescindible tenir una condició de sortida.

- Un exemple. Càlcul del factorial d'un nombre.
  - n! = n·(n-1)·(n-2)·...·1 = n·(n-1)! si n>0, on n! = 1 si n = 0.

```
//Càcul del factorial de n
//Programa principal
Enter n, f
Inici
    Llegir(n)
    f=factorial(n)
    Escriure(f)
Fi
```

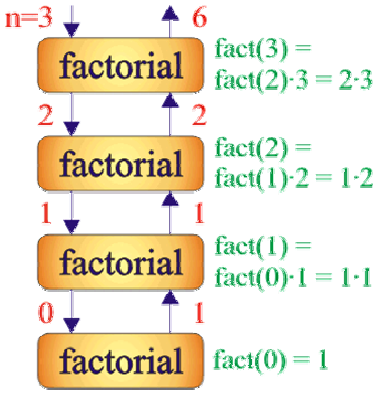
```
//Funció càlcul factorial (no recursiu)
Enter i
Enter factorial (Enter n)
Inici
    fact=1;
    Per (i=n; i>1) Fer
        fact=fact·i;
    Tornar(fact)
Fi factorial
```

```
//Funció càlcul factorial (recursiu)
Enter factorial (Enter n)
Inici
    Si n=0 Llavors Tornar(1)
    SiNo Llavors
        Tornar( n·factorial(n-1) )
Fi factorial
```


5.9 La recursivitat (II)

- La codificació en C quedaria de la forma següent:

```
//Càlcul del factorial
int fact (int);                 //Declaració de la funció
void main ()
{
    int n, f;
    printf("Entra n: ");        //Entrada del número
    scanf("%i", &n);
    f=fact(n);                  //Crida de la funció
    printf("Fact(%i)=%i\n",n,f); //Impressió del resultat
}
int fact(int x)                 //Definició de la funció
{
    if (x==0) return (1);       //Condició de retorn
    else return (x*fact(x-1));  //Recursivitat
}
```

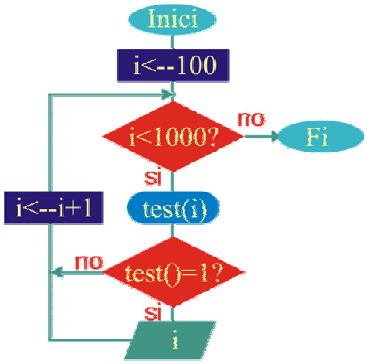


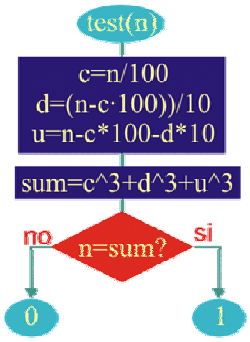
- Fins a quin valor de n proporcionarà, aquest algorisme, valors vàlids?

5.10 Exemples (I)

- Imprimir per pantalla tots els nombres de tres xifres tals que la suma dels seus cubs doni el mateix número.

```
//Trobar els n=c^3+d^3+u^3
#include <stdio.h>
int test (int);                    //Declaració funció test
void main()                        //Funció principal
{
      int i;
      for (i=100; i<1000; i++)
            if (test(i)) printf("%i ", i);  //Bucle principal
      printf("\n");
}
int test (int n)                   //Definició funció test
{
      int u, d, c, sum;
      c=n/100;                     //xifra centenes
      d=(n-c*100)/10;              //xifra desenes
      u=n-c*100-d*10;              //xifra unitats
      sum=c*c*c+d*d*d+u*u*u;
      if (n==sum) return(1);       //Es compleix
      else return(0);              //No es compleix
}
```



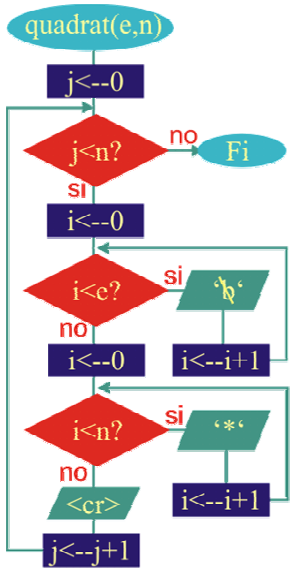


5.10 Exemples (II)

- Exemple de programació modular amb pas per valor: Realitzar un programa que generi la figura de l'esquerra.
  - La funció generada imprimeix cada quadrat

```
  * * *
   * * *
   * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
  * * * * *
  * * * * *
  * * * * *
  * * * * *
  * * * * *
```

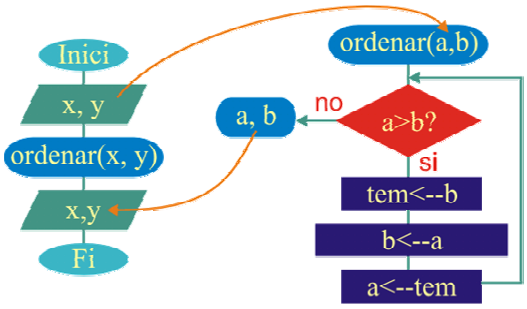
```
//Generant quadrats
#include <stdio.h>
void quadrat(int , int);   //paràmetres: espai des de costat i número d'*
void main()
{
    quadrat (2, 3);        //quadrat de 3 estrelles a 2 espais
    quadrat (0, 7);        //quadrat de 7 estrelles a 0 espais
    quadrat (1, 5);        //quadrat de 5 estrelles a 1 espais
}

void quadrat(int e, int n)    //Generació dels quadrats
{
    int i, j;
    for (j=0; j<n; j++)
    {
        for (i=0; i<e; i++) printf(" ");     //Impressió espais
        for (i=0; i<n; i++) printf("*");     //Impressió *
        printf("\n");
    }
}
```



5.10 Exemples (III)

- Exemple d'aplicació pas per referència.
- Entrar dos valors en dues variables. Emprant un procediment d'ordenació que treballi sobre les dues variables imprimir-los ordenats.

```
//Treballant amb pas de paràmetres
#include <stdio.h>
void ordenar(int &, int &);
void main()
{
    int x, y;
    printf("Entra dos enters: ");
    scanf("%i%i", &x, &y);
    ordenar(x,y);
    printf("x=%i, y=%i\n", x, y);
}
void ordenar(int &a, int &b)
{
    int tem;
    if (a>b)
    {
        tem=b;
        b=a;
        a=tem;
    }
}
```



5.10 Exemples (IV)

- Exemple d'aplicació pas per referència.
- És interessant intentar visualitzar el resultat d'aquest exemple de memòria sense executar-lo.

```
//Treballant amb pas de paràmetres
#include <stdio.h>
void ProcN(int i, int *j);
void main()
{
    int A=2;
    int B=3;
    printf(" A = %i, B = %i\n", A, B);
    ProcN(A, &B);
    printf(" A = %i, B = %i\n", A, B);
}

void ProcN(int i, int *j)
{
    i += 10;
    *j += 10;
    printf(" A = %i, B = %i\n", i, *j);
}
```

- Que passaria si féssim printf(" A = %i, B = %i\n", i, j); ?

5.10 Exemples (V)

- Exemple que mostra la limitació d'emprar funcions recursives.
  - Aquest programa mostra com s'omple la pila d'execucio quan s'empra la recursivitat. Tant funcio1 (normal) com funcio2 (recursiu) no fan res mes que incrementar una variable. Es fàcil comprovar que funcio1 no falla per a tot el rang d'enters. En canvi, funcio2, no passarà, en alguns ordinadors, gaire mes enlla d'11286!

```
#include <stdio.h>
int funcio1 (int, int); //declaracio funció recursiva
int funcio2 (int, int); //declaracio funció recursiva
void main()             // funció principal
{
    int n, x1, x2;
    do
    {
        printf("Numero? "); //entrar número de test
        scanf("%i", &n);
        x1=funcio1(0, n);       //crida funció normal
        printf("\n");
        x2=funcio2(0, n);       //crida funció recursiva
        printf("\n");
    }while(n!=0);
}
```

```
int funcio1(int a, int n)   //declaració funció recursiva 1
{
    for (int y=a; y<=n; y++) printf("\r\t\tFuncio1 = %i", y);;
    return (y);
}
int funcio2(int a, int n)   //declaració funció recursiva 2
{
    printf("\r\t\tFuncio2 = %i", a);
    if (a<n) return(funcio2((a+1),n));
    else return (a);
}
```