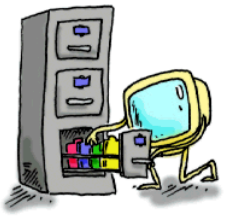
Tema 5: Programació estructurada

- 5.1 Programació estructurada i modular
- 5.2 Procediments i funcions
- 5.3 Funcions
- 5.4 Procediments
- 5.5 Variables i pas de paràmetres
- 5.6 Pas de paràmetres per valor
- 5.7 Pas de paràmetres per referència
- 5.8 Pas de paràmetres per referència amb apuntador
- 5.9 La recursivitat
- 5.10 Exemples



5.1 Programació estructurada i modular (I)

- Programació modular → Un programa consta de la unió de mòduls independents.
- Programació estructurada → Cada mòdul es construeix utilitzant mètodes estructurats.

Programació estructurada

- La programació estructurada es fonamenta en l'ús exclusiu de tres estructures de control: seqüència, selecció i iteració.
  - Un programa estructurat no utilitza sentències de salt incondicional. Exemple: goto
- Ús de recursos abstractes →
- Permet descomposar accions complexes en funció d'instruccions (accions simples) que són executades per la computadora
- Disseny top-down →
- Tot problema pot descomposar-se en nivells successius on cada nivell següent (inferior) és un refinament de l'anterior.
- Estructures bàsiques →
- Tal com demostraren Böhm i Jacopini (1966) emprant estructures seqüencials, selectives i iteratives es pot escriure qualsevol programa. A més, aquest programa compleix les característiques de tenir:
  - Un únic punt d'entrada i un de sortida (o fi) per control del programa.
  - Existeixen camins des de l'entrada del programa a la sortida que es poden seguir i que passen per totes les parts del mateix.
  - Totes les instruccions són executables i no existeixen bucles infinits (sense fi).

5.1 Programació estructurada i modular (II)

Programació modular

- La programació modular ajuda a construir programes 'ben fets': correctes (donen el resultat esperat), llegibles (per qualsevol programador), modificables i depurables.
  - Un mòdul és un conjunt d'instruccions que realitzen una tasca específica i tenen una interfície ben definida: un punt d'entrada i un de sortida
  - Interfície és el conjunt d'entitats necessàries per a utilitzar el mòdul
  - En un disseny modular és important descompondre el problema en subproblemes.
  - En la major part de llenguatges això passa per implementar el programa emprant funcions i procediments (subrutines): tots dos realitzen una tasca determinada retornant el control a qui els ha cridat.

Exemple de programació estructurada:

1. Posar el títol (comentari) del programa.
2. Directives del processador.
3. Explicar (capçalera) què fa el programa.
4. Definició de variables.
5. Declaració de constants numèriques.
6. Declaració de funcions.
7. Funció principal (main()).
8. Altres definicions (funcions i procediments).

Acompanyar el programa amb els comentaris que calguin

Exemple d'estructura modular simple.

Mòdul principal

- → B.1. Mòdul impressió de capçalera.
- → B.2. Mòdul de procés de dades.
- → B.3. Mòdul d'impressió.

5.1 Programació estructurada i modular (III)

Exemple

- Programa que calcula la hipotenusa d'un triangle rectangle.

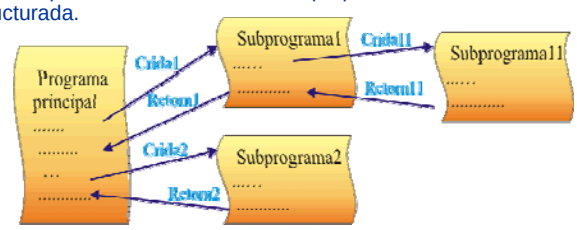
```
//Algorisme hipotenusa
/*Aquest programa calcula la hipotenussa
d'un triangle.
Consta del programa principal i d'una funció (retorna un
valor real) que calcula la hipotenusa*/
Real calculH(Real a, b)        //Declaració funció
Real x, y, h                   //Declaració variables
Inici                          //Programa principal
    Escriure("Programa calcul hipotenusa")
    Llegir(x,y)
    h <- calculH(x,y)
    Escriure(h);
Fi
Real calculH(Real a, b)        //Definició funció
Inici
    Tornar (arrel_quadrada(a^2+b*2))
Fi
```

```
//Algorisme hipotenusa
/*Aquest programa calcula la hipotenussa
d'un triangle.
Consta del programa principal i d'una funció (retorna un
valor real) que calcula la hipotenusa*/
float calculH(float , float )      //Declaració funció
float x, y, h                      //Declaració variables
void main()                        //Programa principal
{
    printf("Programa calcul hipotenusa\n")
    printf("      Entra x i y: \n")
    scanf("%f%f", &x, &y)
    h = calculH(x,y)
    printf("      h = %f\n", h);
}
float calculH(float a, float b)    //Definició funció
{
    return (arrel_quadrada(a^2+b*2))
}
```

- Es pot observar clarament que, en la realització de la funció, hi ha tres etapes:
  - Declaració
  - Definició
  - Crida a la funció

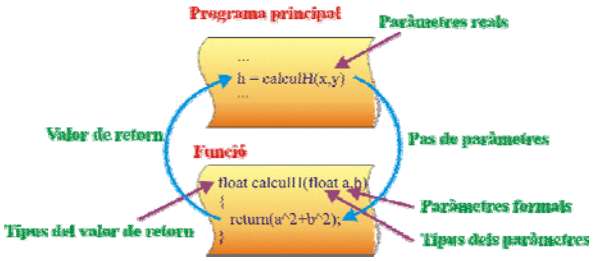
5.2 Procediments i funcions

- Disseny descendent o top-down →
  - És una forma de resoldre problemes complexos: divideix i conqueriràs
  - Al conjunt de subprogrames que resolen el problema principal se'ls anomena funcions i procediments
- Funcions i procediments →
  - Compleixen els axiomes prèviament descrits en la programació estructurada:
    - Tots els blocs (funcions i procediments) tenen un únic punt d'entrada.
    - Tots els blocs tenen un únic punt de sortida.
    - Quan s'arriba al final d'un bloc es continua amb un altre o acaba.
    - Un programa pot tenir diferents nivells de subprogrames:
  - En l'exemple anterior es fa ús de les propietats de la modularitat i la programació estructurada.



5.3 Funcions

- És una operació que rep un o més valors, anomenats arguments, i genera **un** valor de retorn anomenat resultat
  - Certs llenguatges (com el C) requereixen de la declaració prèvia de la funció abans del seu ús.
- La funció s'ha de declarar i definir. La declaració s'ha de fer previ crida de la mateixa.
- S'ha d'especificar el tipus de variables dels arguments i del resultat.
  - Exemple: com en l'exemple del càlcul de la hipotenusa.
- Paràmetres formals i reals.
  - Els paràmetres reals són els paràmetres que s'utilitzen en el càlcul en l'algorisme.
  - Els paràmetres formals són els emprats en la definició de la funció.
  - Les variables emprades pels paràmetres formals i els reals no tenen per què ser les mateixes.



5.4 Procediments (I)

- És un subprograma que executa un procés determinat.
- Es diferencia de les funcions en:
  - Les funcions retornen un valor. Els procediments en poden retornar zero, un o més .
  - El nom de procediment no està lligat a cap tipus de valor .
- A l'igual que les funcions:
  - Hi ha els paràmetres formals i els reals.
  - Les variables emprades pels paràmetres formals i els reals no tenen per què ser les mateixes.
  - Donat que no retorna valor de forma explícita, en C, s'indica amb el tipus de variable void.
- Exemple: Cas típic d'una rutina d'impressió de resultats.
  - Un procediment (funció) es pot fer servir en múltiples ocasions (és una rutina).

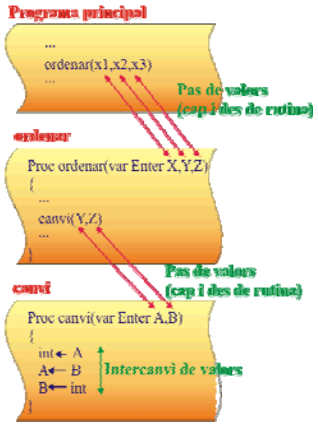
```
void imprimir(char[]);                  //Declaració
void main()
{
    imprimir("Comencem... \n");         //Ús
    ...
    imprimir("...aqui s'ha acabat! \n"); //Ús
    imprimir("Adeu!!\n");               //Ús
}
void imprimir(char[20] cadena);         //Definició
{
    printf("%s\n", cadena);
}
```


5.4 Procediments (II)

- Exemple
  - El següent exemple ordena de major a menor tres nombres. Observar:
    - Com el programa principal només fa de controlador de successos, cridant als tres procediments llegir_xs (entrada de dades), ordenar (mòdul de procés) i escriure (sortida de dades). Al seu temps, ordenar crida al procediment canvi per a efectuar la tasca rutinària d'ordenar els valors de dues dades.
    - El pas de paràmetres posicional en els procediments ordenar i canvi. Sense canviar el nom ni l'ordre de les variables emprades efectuem la ordenació dels nombres (continguts de les variables)!.
    - x1, x2 i x3 són variables globals del programa. La variable intern (en el proc. canvi) és local.

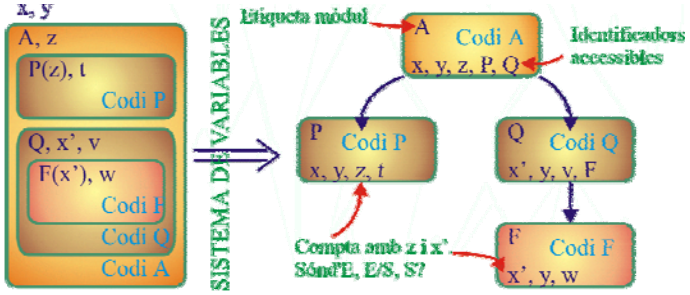
```
//Algoritme ordenació
Enter: x1, x2, x3
Inici
        llegir_xs()
        ordenar(x1, x2, x3)
        escriure()
Fi
Procediment llegir_xs
Inici
        Llegir (x1)
        Llegir (x2)
        Llegir (x3)
Fi llegir_xs
Procediment ordenar (var int: X, Y, Z)
Inici
        Si (X<Y) Fer canvi (X, Y)
        Si (Y<Z) Fer canvi (Y, Z)
        Si (X<Y) Fer canvi (X, Y)
Fi ordenar
```

```
Procediment canvi (var Enter: A, B)
var Enter: intern
Inici
        intern ← A
        A ← B
        B ← A
Fi canvi
Procediment escriure_xs
Inici
        Escriure (x1)
        Escriure (x2)
        Escriure (x3)
Fi escriure_xs
```



5.5 Variables i pas de paràmetres (I)

- Variables globals i variables locals.
  - Identificador global → Àmbit tot el programa. És a dir, es pot utilitzar en tots els mòduls. Estan declarades en nivells alts.
  - Identificador local al mòdul → Quan el seu ús queda restringit al mòdul. Són les declarades en cada mòdul. El seu àmbit és en el mòdul i/o en els mòduls 'fills'.
  - El terme identificador fa referència tant a noms de funcions/procediments com de variables.
- Els identificadors que es passen entre mòduls poden tenir diferent pas de paràmetres segons el llenguatge que s'utilitzi. S'ha de vigilar, sobretot, quan les variables globals poden ser modificades en el submòdul.
- El pas de paràmetres estableix la comunicació entre procediments i subprocediments



5.5 Variables i pas de paràmetres (II)

- Àmbit de les variables en el C.
  - En C hi ha variables locals (dintre de procediments) i globals (definides abans del *main()*).
  - S'ha d'anar en compta amb les variacions de les variables globals dintre dels procediments.
- Exemple: Segons on es defineix la variable A la sortida és diferent.

```
① ← int A=0;
void procediment(void);
void main()
{
      ② ← int A = 1;
      printf("1. A = %i\n", A);
      procediment();
      A += 3;
      printf("2. A = %i \n", A);
}
void procediment(void)
{
      ③ ← int A = 7;
      A += 1;
      printf("3. A = %i \n", A);
}
```

En ① → El resultat serà: 1. A = 0.
3. A = 1.
2. A = 4.
El procediment modifica A → A és variable global.

Només en ② → Error de compilació: procediment() no té definida A.

En ② i en ③ → 1. A = 1.
3. A = 8.
2. A = 4.
A és diferent variable en procediment() i en main().
En ocupar posicions de memòria diferents emmagatzemen diferents valors.

En ① i en ③ → 1. A = 1.
3. A = 8.
2. A = 3.
Passa com en el cas anterior.

5.5 Variables i pas de paràmetres (III)

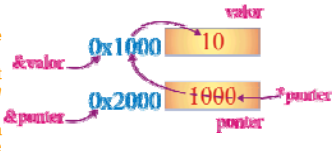
- Tipus de variables en C
  - Constants → No canvien en tot el programa.
    És aconsellable l'ús de variables globals constant.
  - Automàtiques → Per defecte, qualsevol variable definida en una funció es considera automàtica.
    Totes les emprades fins ara han estat automàtiques.
  - Externes → Variable global llur definició apareix en una altra part del programa (o en un altre arxiu font del programa)
  - Estàtiques → Variable local, però que té vida mentre duri el programa.
    La variable estàtica declarada en una funció manté el seu valor entre diferents invocacions de la funció.
  - De registre → Indica al compilador que s'ha d'emmagatzemar en un registre (d'accés més ràpid que la memòria) de la CPU.

  - Exemple:
    - const float PI = 3.141592; Constant.
    - (auto) int valor; Variable automàtica.
    - extern char caracter; Variable externa.
    - static int fusible; Variable estàtica.
    - register int comptador; Variable registre.

5.5 Variables i pas de paràmetres (IV)

- Emmagatzament de variables en memòria
  - Fins ara s'ha vist que les variables es declaren i es defineixen (s'usen).
  - Quan es declara no es fa res més que reservar l'espai de memòria que cal per al tipus de variable declarada.
  - Exemples:
    - int total; → Reserva 2 bytes per a un enter al que s'accedeix mitjançant la constant total
    - float volt; → Reserva 4 bytes per a un real al que s'accedeix mitjançant la constant volt
  - A partir d'ara, per passar paràmetres, sovint parlarem de la posició de memòria de la variable i del contingut de la variable. O, dit més normalment, d'adreces i apuntadors.
  - Per parlar de l'adreça d'una variable emprarem l'operador d'adreça &. Per parlar del contingut de la variable emprarem l'operador d'indirecció *.

- Així, en l'exemple següent:
  - valor i punter són dos identificadors que referencien dues posicions de memòria.
  - Les adreces que referencien són: &valor=0x1000 i &punter=0x2000
  - Quan es fa valor=10 s'assigna el valor (contingut) 10 a la posició de memòria referenciada per la variable valor.
  - Si ara es defineix la variable punter com a punter=&valor, s'està dient que a la *posició de memòria referenciada per punter hi posarem l'adreça de la posició de memòria referenciada per valor*.
  - En conseqüència la variable punter apunta a la posició de memòria 0x1000 i *punter=10 (llegeixi's com el contingut de la variable referenciada per l'apuntador *punter.



5.6 Pas per paràmetre valor

- Els pas de paràmetres més coneguts són el pas per valor i el pas per referència. En C el pas per apuntador sol substituir el pas per referència.
- Característiques del pas de paràmetres per valor:
  - Emprat en llenguatges com C, Pascal, Algol, ...
  - Correspondència (entre procediment i subprocediment) posicional.
  - Els paràmetres són tractats com a variables locals.
  - Valors inicials obtinguts per còpia en la correspondència posicional.
  - No retorna informació (el valor de la variable no ha canviat) en el retorn del subprograma.

Pseudocodi

```
//Exemple: Elevar al cub
Real cub (Real n)
Inici
    Real x
    Escriure (“Entra numero: “)
    Llegir(x)
    Escriure(cub(x))
Fi
Real cub(Real n)
Inici
    n=n^3
    Tornar(n)
Fi cub
```

Diagrama de flux

Codificació C

```
//Exemple: Elevar al cub
float cub(float n);      //Declaració funció cub
void main()              //Funció principal
{
    float x;
    printf("Entra numero: ");
    scanf("%f", &x);
    printf(“%6.2f ^3 es %6.2f.\n", x, cub(x));
}
 float cub(float n)      //definició funció cub
{                        //n es el valor formal
    n = n*n*n;
    return(n);
}
```


5.7 Pas per paràmetre per referència (I)

- Característiques del pas per paràmetre referència:
  - La unitat que crida passa l'adreça del paràmetre variable. És una referència a la posició de memòria de la variable.
  - Així el subprogama pot modificar el contingut de la variable passada. En tornar al procediment que ha fet la crida, es manté el valor modificat (en el subprograma) de la variable.
  - Es poden considerar variables d'E/S.
  - És un pas directe en FORTRAN, COBOL, Pascal, ... És més delicat en C.
  - En pseudocodi s'indica posant la paraula var al davant de la declaració del tipus de paràmetre.
  - Exemple:

```
...
A ← 5
B ← 7
proc_1(A, 18, B*3+4)              //Crida a procediment
...

procediment proc_1 (var Enter x, y, z)   //x=5, y=18, z=25
Inici
    ...
    x = x+2
    ...
Fi proc_1                         //En retornar, A = 7, i B = 7
```

5.7 Pas per paràmetre per referència (II)

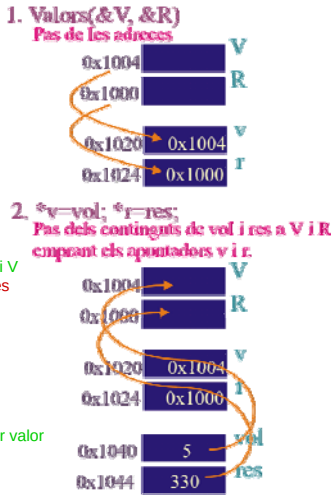
- En C/C++ s'ha de fer passant l'adreça de les variables o passant un apuntador. En aquest exemple es veu el pas per referència recollint les adreces de les variables &.

```
//Càlcul del corrent donats V i R
#include <stdio.h>
void Valors(float&, float&);          //Procediment amb pas d'adreces
float Calcul(float, float);           // Funció: retorna un valor float
void main()
{
    float V, R, I;                    //Variables locals
    Valors(V, R);                     //La crida passa les adreces
    I = Calcul(V, R);                 //Càlcul d'I
    printf("El corrent de V/R = %2.1f/%2.1f = %2.1f\n",V, R, I);
}
void Valors(float& v, float& r)       // Es treballa amb les adreces de R i V
{
    float vol, res;                   v i r recullen les adreces passades de V
    printf(" Dona V i R: ");          i R a on s'hi posaran  els nous valors!
    scanf("%f%f", &vol, &res);
    v = vol;                          vol i v (res i r) són dues constants referenciant
    r = res;                          a la mateixa posició de memòria!
}
float Calcul(float vo, float re)      //Paràmetres volt i resist passats per valor
{
    return(vo/re);
}
```

5.8 Pas de paràmetres amb apuntador

- L'ús d'apuntadors és la forma més usual de pas de paràmetres en C. En aquest cas es pot observar com es faria en l'exemple anterior.

```
//Càlcul del corrent donats V i R
#include <stdio.h>
void Valors(float*, float*);          //Procediment amb pas d'adreces
float Calcul(float, float);           // Funció: retorna un valor float
void main()
{
    float V, R, I;                    //Variables locals
    Valors(&V, &R);                   //La crida passa les adreces
    I = Calcul(V, R);                 //Càlcul d'I
    printf("El corrent de V/R = %2.1f%2.1f = %2.1f\n",V, R, I);
}
void Valors(float* v, float* r)       // Es treballa amb les adreces de R i V
{                                     Tot i que es pot fer sense emprar les
    float vol, res;                   variables intermitges vol i res
    printf(" Dona V i R: ");              scanf("%f%f", &(*v), &(*r)) =
    scanf("%f%f", &vol, &res);            scanf("%f%f", v, r),
    *v = vol;                         per claretat, és millor emprarles.
    *r = res;
}
float Calcul(float vo, float re)      //Paràmetres volt i resist passats per valor
{
    return(vo/re);
}
```



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
```

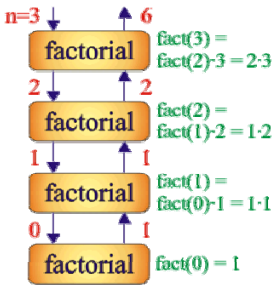
```
//Funció càlcul factorial (no recursiu)
Enter i
Enter factorial (Enter n)
Inici
    fact=1;
    Per (i=n; i>1) Fer
        fact=fact·i;
    Tornar(fact)
Fi factorial
```

```
//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
}
```

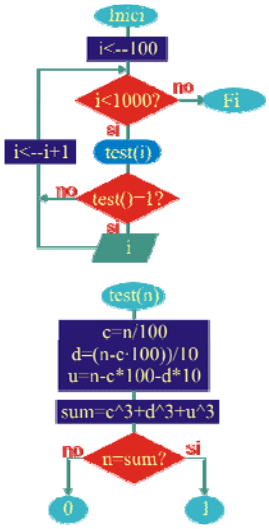


- Fins a quin valor de n proporcionarà, aquest algorisme, valors vàlids?

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
}
```

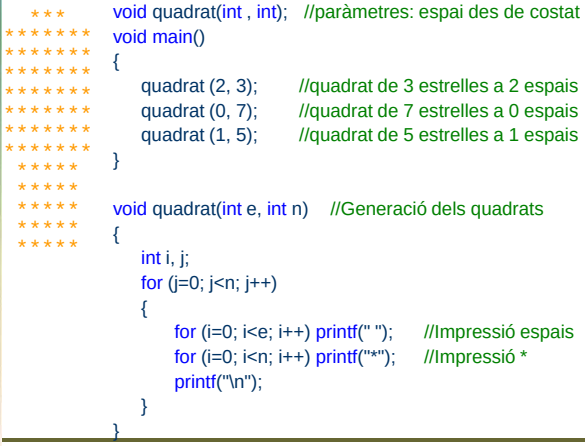


5.10 Exemples (II)

Exemple de programació modular amb pas per valor: Realitzar un programa que generi la figura de l'esquerra.

- La funció generada imprimeix cada quadrat

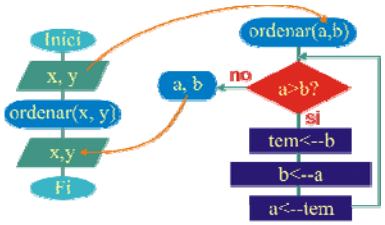
```
//Generant quadrats
#include <stdio.h>
void quadrat(int , int);   //paràmetres: espai des de costat i número d'*
void main()
{
    quadrat (2, 3);        //quadrat de 3 estrelles a 2 espais
    quadrat (0, 7);        //quadrat de 7 estrelles a 0 espais
    quadrat (1, 5);        //quadrat de 5 estrelles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