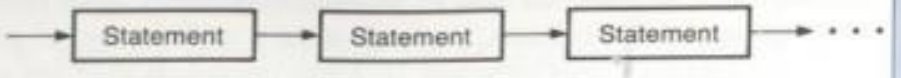
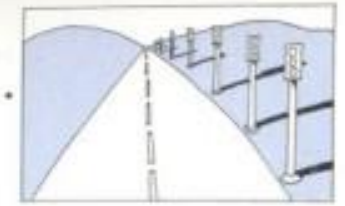
Funcții

Introducere

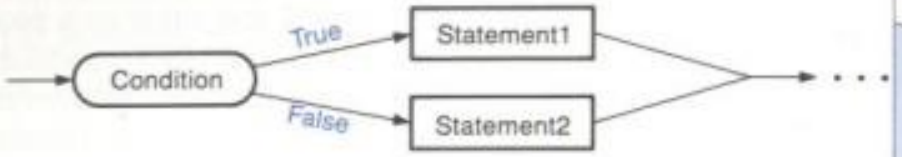
SEQUENCE

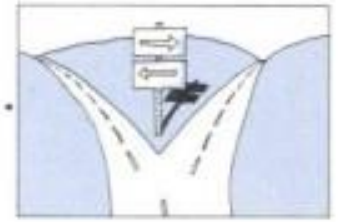




SELECTION (also called branch and decision)

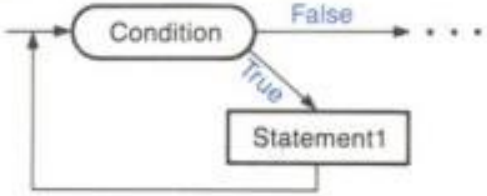
IF condition THEN statement1 ELSE statement2



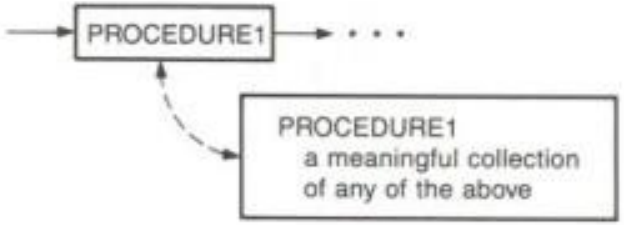


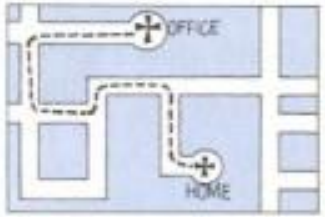
LOOP (also called repetition and iteration)

WHILE condition DO statement1



PROCEDURE (also called subprogram and subroutine)





Principiile programării structurate

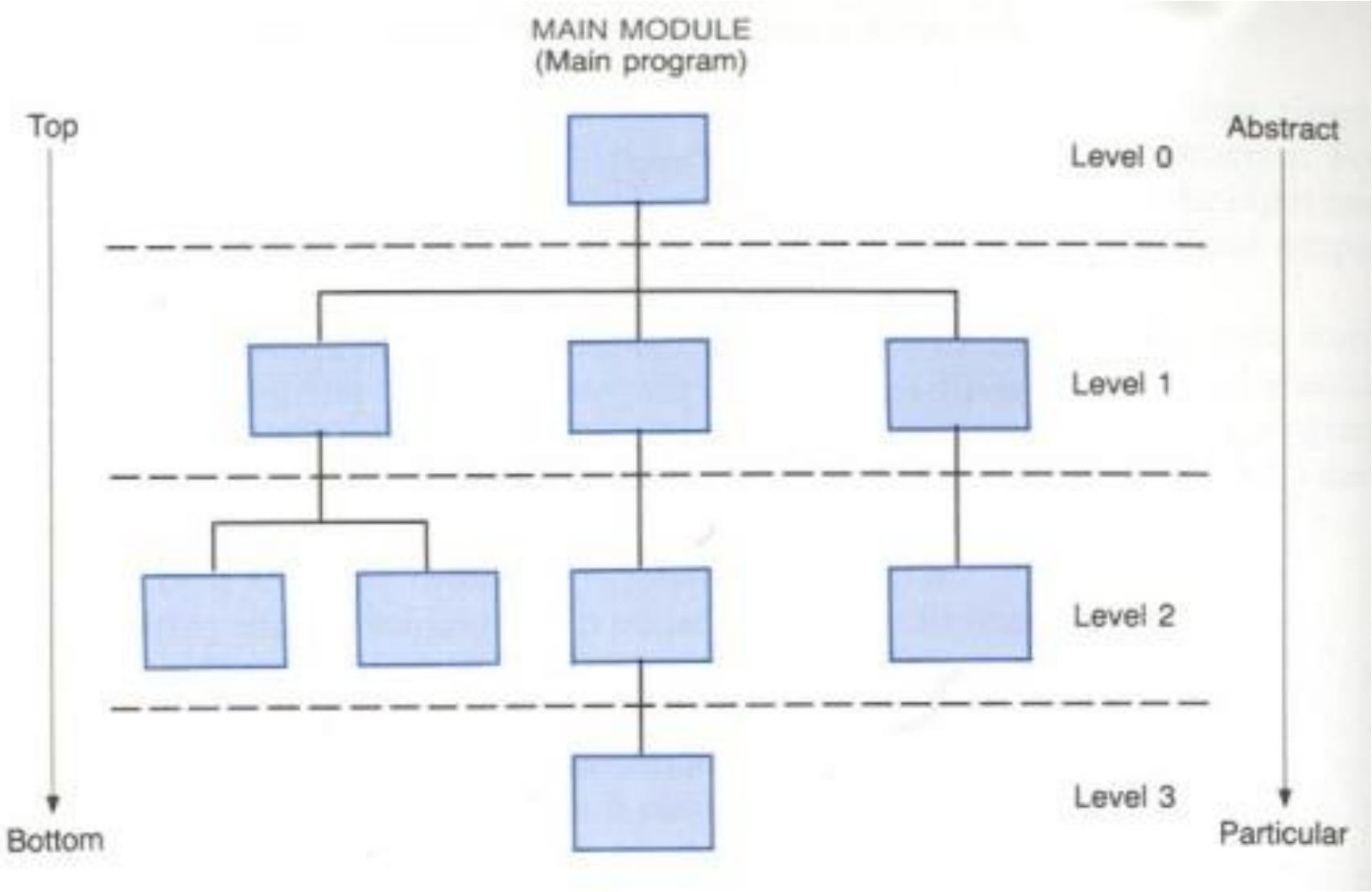
- **TOP-DOWN DESIGN**

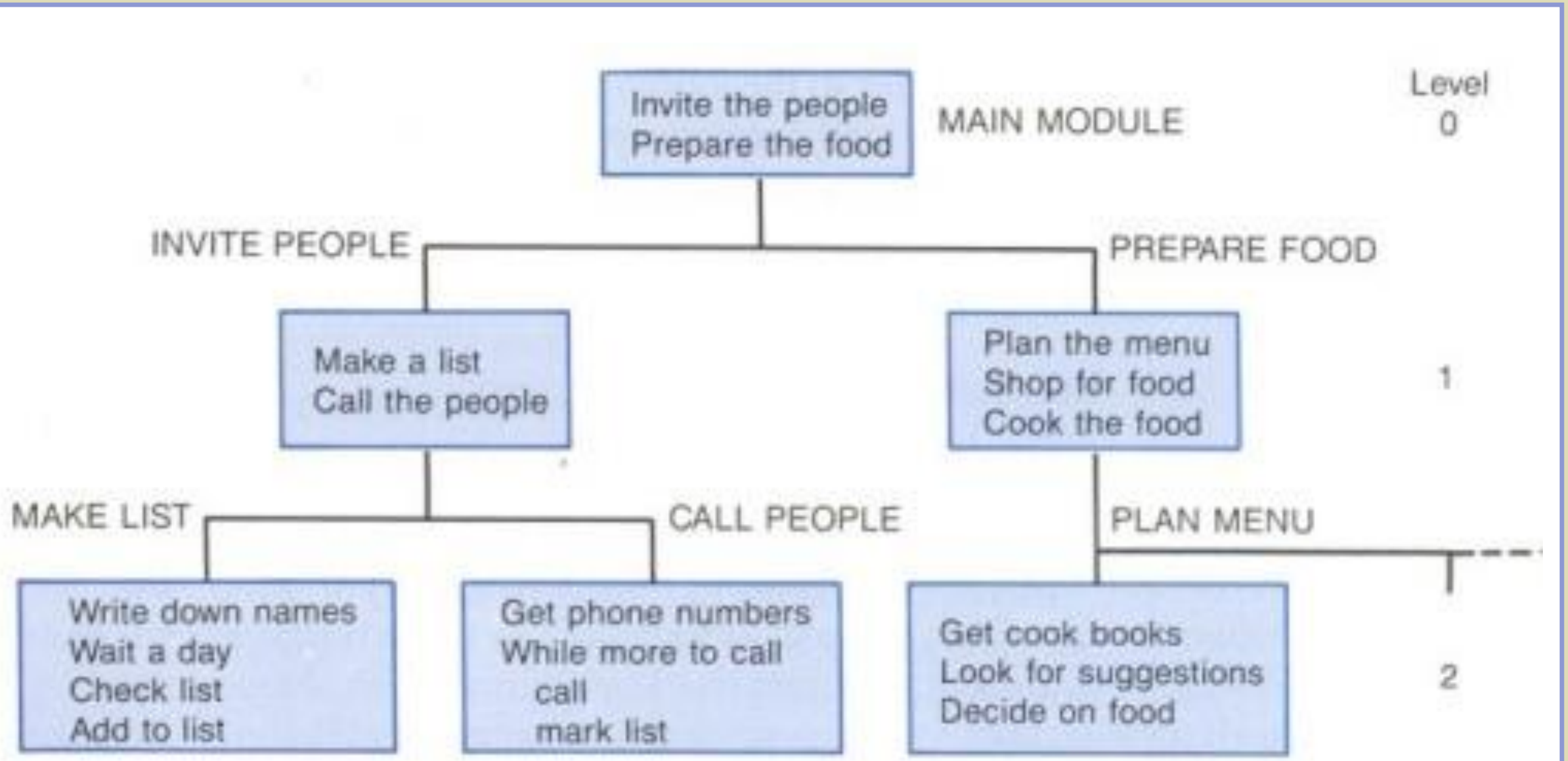
Rezolvarea problemelor mari este mai eficientă în cazul divizării acestora în probleme elementare (standard). Procesul de analiză,divizare a problemei generale în probleme elementare şi de stabilire a relațiilor între problemele determinate

În cadrul programului procesul de divizare este realizat prin construirea "blocurilor" destinate pentru rezolvarea problemelor elementare. Deoarece una şi aceeaşi subproblemă poate să fie realizată de mai multe ori în procesul de rezolvare a problemei principale, dar cu date inițiale diferite, este rațional a "blocul" să se poată adapta la setul de date inițiale. Astfel apar "modulele" de program, rezultatul funcționării cărora este determinat nu numai de algoritmul realizat, dar şi de setul de date inițiale. În limbajele de programare modulele sunt realizate prin subprograme (funcții).

Principiile:

1. Divizarea problemei în subprobleme elementare
2. Stabilirea relațiilor între subprobleme
3. Realizarea modulelor pentru fiecare dintre subprobleme
4. "Asamblarea" problemei inițiale într-un "modul principal"


MAIN MODULE
(Main program)
Top
Bottom
Level 0
Level 1
Level 2
Level 3
Abstract
Particular


Invite the people
Prepare the food
MAIN MODULE
Level
0
INVITE PEOPLE
PREPARE FOOD
Make a list
Call the people
Plan the menu
Shop for food
Cook the food
1
MAKE LIST
CALL PEOPLE
PLAN MENU
Write down names
Wait a day
Check list
Add to list
Get phone numbers
While more to call
call
mark list
Get cook books
Look for suggestions
Decide on food
2

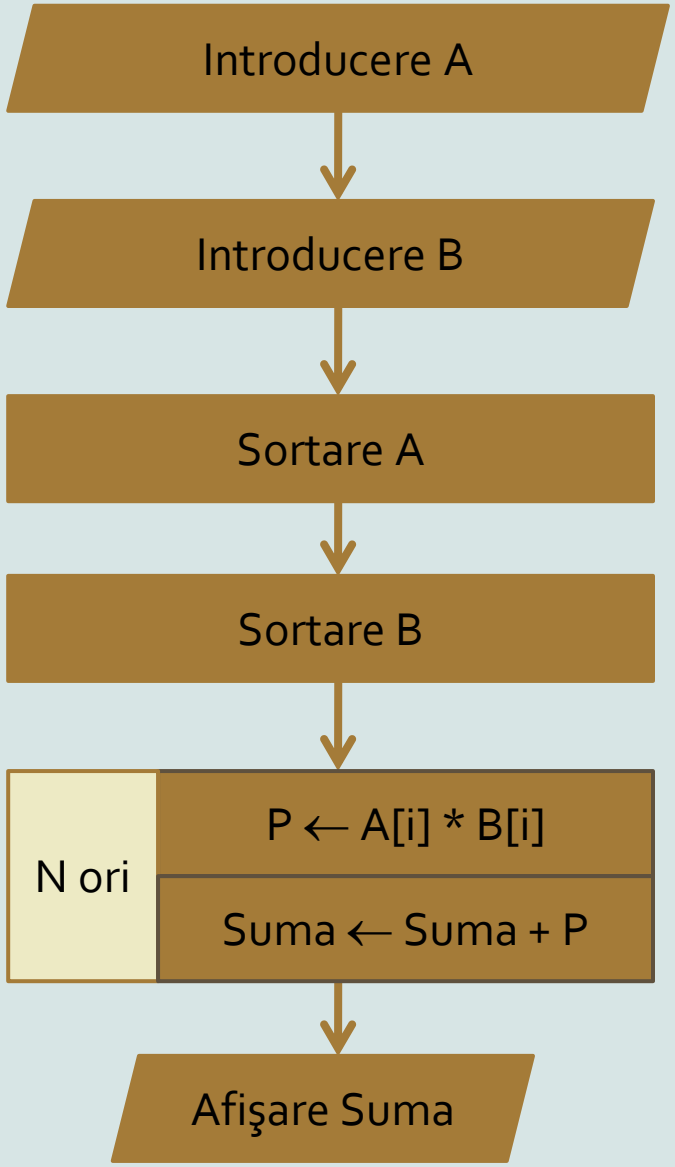
Exemplu

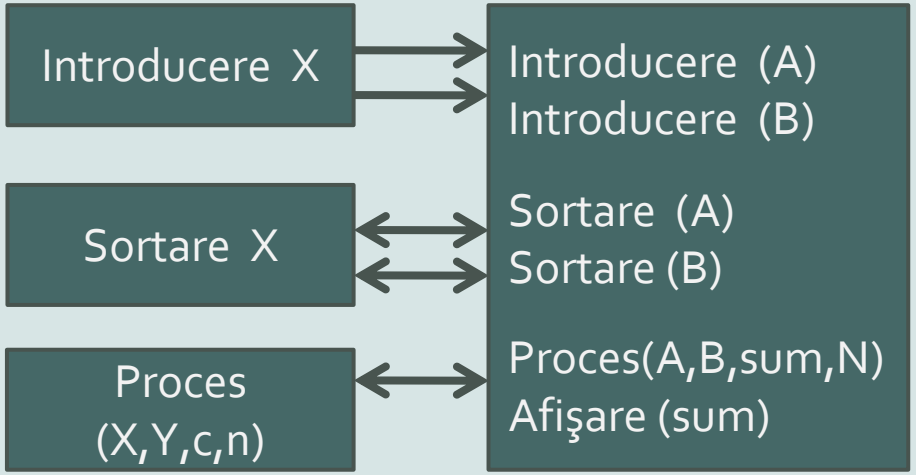
Fie două şiruri a câte N elemente - numere întregi. Să se formeze N perechi cu elemente distincte din şiruri diferite, astfel încât suma produselor formate de elementele perechii să fie maximă. `[ 2 5 1] [4 3 6]`
`[5*6+2*4+1*3]`

Subprobleme:

1. Introducerea valorilor elementelor tabloului
2. Sortarea elementelor tabloului
3. Calculul produsului a două elemente
4. Calculul sumei a două elemente
5. Afişarea sumei

Schema logică





Noțiuni generale

subprogram (funcție) – modul al programului, identificat prin numele său, care rezolvă o problemă elementară, prin intermediul unui apel la numele său.

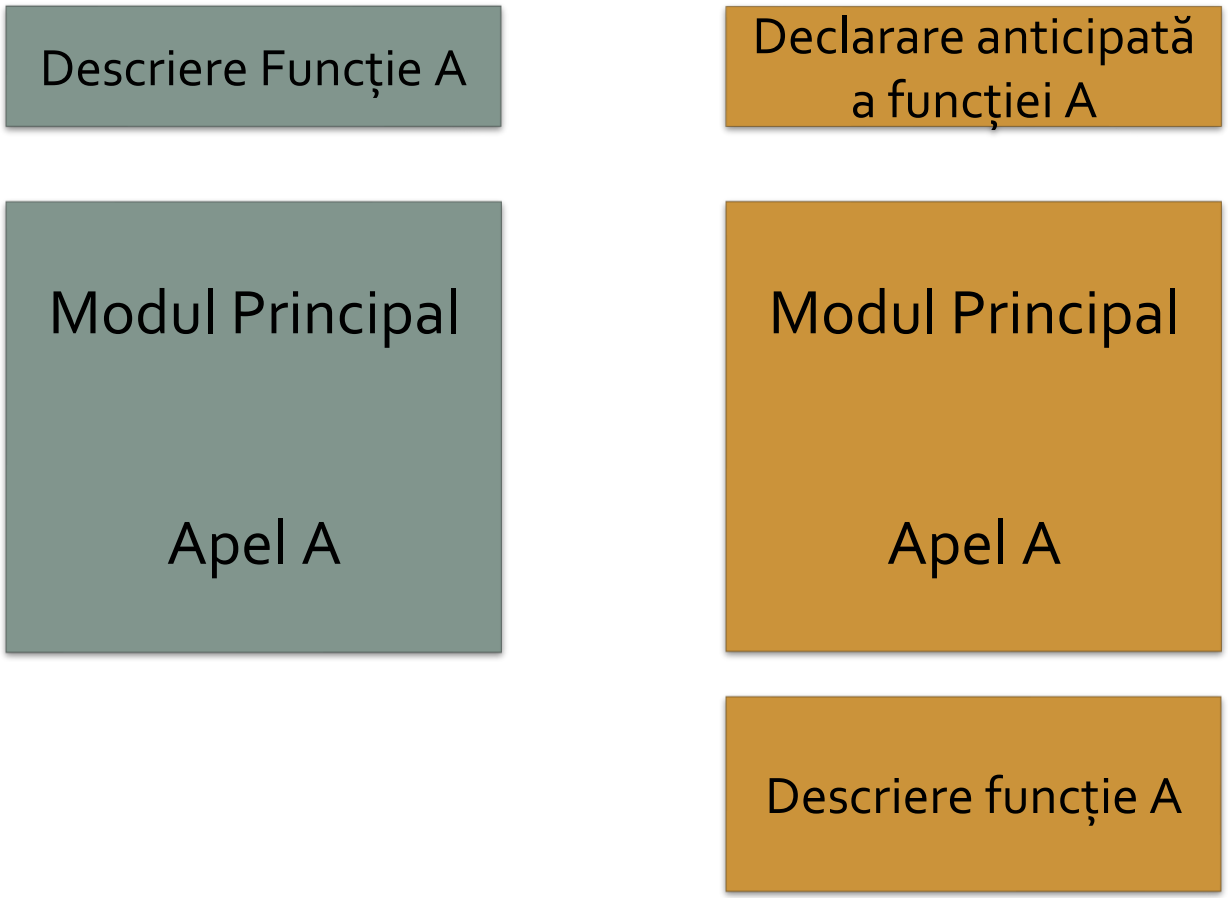
Parametri ai subprogramului - setul de valori, care urmează a fi transmise din modulul principal în subprogram, pentru funcționarea acestuia

Parametri formali – parametrii folosiți în descrierea subprogramului

Parametri actuali – parametrii folosiți în apelul subprogramului

Tipul funcției – tipul rezultatului returnat de subprogram

Scheme de apel



Exemple

```
# include <iostream.h>

int max(int x, int y)
{
  if (x > y) return x;
      else return y;
}

int main()
{
   int a, b;
   cin >> a >> b;
   cout << max (a, b);
}
```

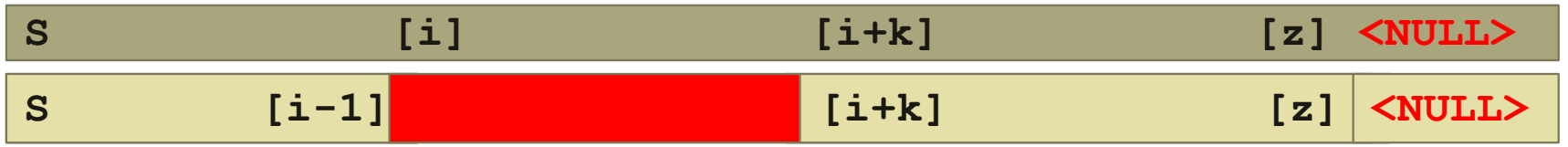
```
# include <iostream.h>

int max (int x, int y);

int main()
{
   int a, b;
   cin >> a >> b;
   cout << max (a, b);
}

int max(int x, int y)
{
  if (x > y) return x;
      else return y;
}
```

Tablouri ca parametri

Array, array of char

Exemplul 1

Modelarea funcției de lichidare a **k** caractere dintr-un şir **s**, începând cu cel din poziția **i**.




```
# include <stdlib.h>
int k, i;
void del(char *s, int k, int i)
{
  int l, z;
  z = strlen(s); // Exercitiu: scrieti propria fuctiie strlen
  for (l = i; l < z - k; l++) s[l] = s[l + k];
  s[l] = NULL;
}

int main()
{
  char *s;
  s=((char *) malloc(100));
  printf("introduceti S:"); gets(s);
  printf("introduceti k,i:"); scanf("%d%d", &k,&i);
  del(s, k, i);
  puts(s);
}
```

Exemplul 2

Sortarea unui tablou de numere întregi

.

```
#include <stdio.h>

int a[15],n,i,q,j,k;
FILE *in,*out;

void readdata(int *a)
{    int i;
     in = fopen("data.in","r");
     fscanf(in, "%d", &n);
     for (i = 0; i < n; i++)
     fscanf(in, "%d", &a[i]);
     fclose(in);
}

void sort(int *x, int m)
{ int i, j, k;
  for (j = 0; j < m - 1; j++)
    for (i = 0; i < m - 1; i++)
      if (x[i] > x[i + 1])
        { k = x[i];
          x[i] = x[i + 1];
          x[i + 1] = k; }
}
```

```
void printdata(int *x)
{
 int i;
 out = fopen("result.out","w");
 for (i = 0; i < n; i++)
 fprintf(out,"%d ",x[i]);
 fclose(out);
}

void main()
{   readdata(a);
    sort(a,n);
    printdata(a);
}
```

Recursia

Funcții elementare

Noțiuni

Spunem că o noțiune este definită recursiv, dacă în definirea ei apare însăşi noțiunea care se defineşte.

In informatică numim **recursivitate directă**, proprietatea funcțiilor de a se autoapela.

Structura unei funcții recursive:

```
tip nume_funcţie(lista de parameri)
{
    ..................
    if (condiţie de oprire) return <valoarea pentru condiția de oprire>
    else
        {   ..................
            return <formula recurentă> ... nume_funcţie(lista de parametri);
        }
}
```

Observații

Orice funcție recursivă **trebuie să conțină o condiție de reluare a apelului recursiv** (sau de oprire)**.** Fără această condiție, funcția teoretic se reapelează la infinit.

- La fiecare reapel al funcției se execută aceeaşi secvență de instruciuni.

- **Ținând seama de observațiile anterioare, pentru a implementa o funcție recursivă, trebuie să:**
  - **Identificăm relația de recurența (ceea ce se execută la un moment dat şi se reia la fiecare reapel)**
  - **Identificăm conditiile de oprire ale reapelului**

- În cazul în care funcția are **parametrii,** aceştia **se memorează ca şi variabilele locale pe stiva,** astfel:
  - parametrii transmişi prin valoare se memorează pe stivă cu valoarea din acel moment
  - pentru parametrii transmişi prin referință se memorează adresa lor

Exemple

E1:

Problema 1 Factorial

Să se scrie un program recursiv, care calculează n!

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

int n;
long f;
long factor (int k)
{  if (k == 1) return 1;
        else return k * factor(k - 1);
}

void main()
{
  printf("\input N: "); scanf("%d", &n);
  printf("\n%d", factor(n));
}
```

E2:

Problema 2 Fibonacci

Să se scrie un program recursiv, care calculează termenul cu numărul de ordine n al şirului Fibonacci

(caz în care recursia este ineficientă)

Exercițiu: de ce?

```
# include <stdio.h>

int n,k;

long fibonacci (n)
{
  if (n == 0 || n == 1) return n;
        else return fibonacci(n - 1) + fibonacci(n - 2);
}

int main()
{     printf("\input N: "); scanf("%d", &n);
      printf("\n%d", fibonacci(n));
      return 0;
}
```

E3:

Problema 3 Qsort

Să se scrie un program recursiv, care ordonează eficient un șir de numere

```
#include <stdio.h>;

 void qsort(int st, int dr);
 void print(int st, int dr);

 int i,n,a[100];
 FILE *in, *out;

void main()
{ in = fopen("sort.in", "r");    // read data
  fscanf(in, "%d\n", &n);
  for(i = 0; i < n; i++)
       fscanf(in, "%d", &a[i]);
  fclose(in);
  print (0, n - 1);              // print initial array
  qsort (0, n - 1);              // sort array
  print (0, n - 1);              // print sorted array
}
```

E3:

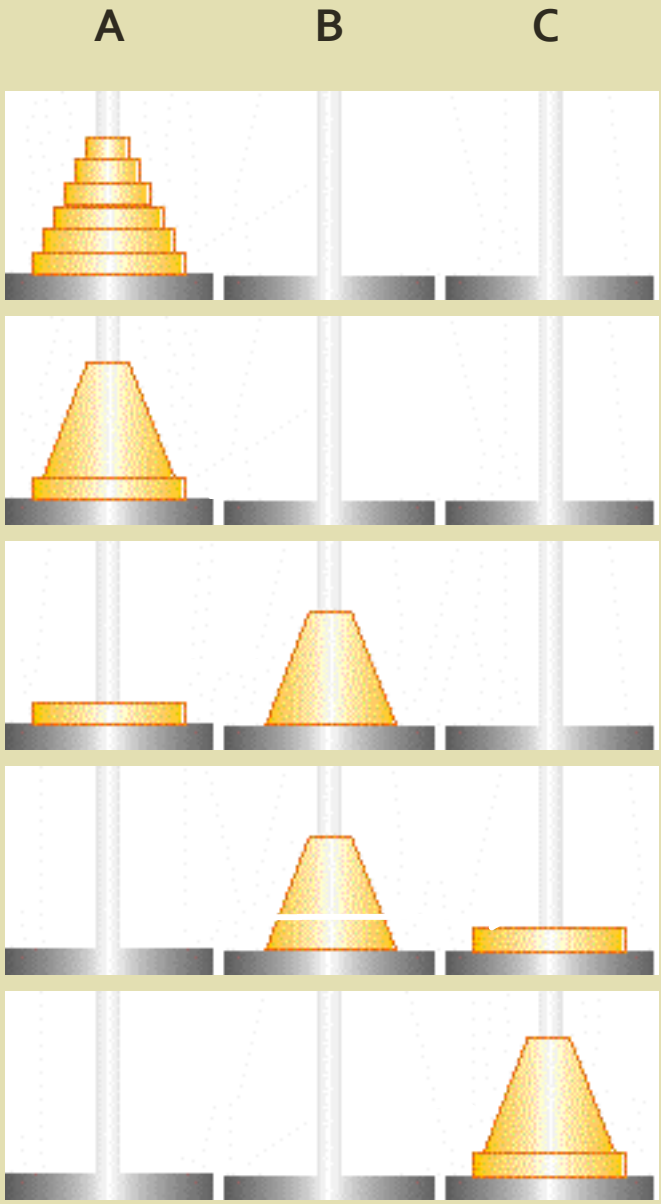
Continuare

```
void qsort(int st, int dr)
{ int i, t, k;
  if (st < dr)
       { print(st, dr);
         k = st;
         for (i = st + 1; i <= dr; i++)
           if (a[i] < a[st])
                {k++; t = a[i]; a[i] = a[k]; a[k] = t;}
         t = a[st]; a[st] = a[k]; a[k] = t;
         qsort(st,k - 1);
         qsort(k + 1,dr);
       }
}

void print(int st, int dr)
{ int i;
  printf("\n");
  for (i = 0; i < n; i++)
       if (i >= st && i <= dr) printf("%3d " ,a[i]);
       else printf("    ");
}
```

E4:

Problema turnurilor din Hanoi

Exercițiu: identificați și citiți legenda asociată problemei



Fie trei tije A, B, C. Pe tija A sunt plasate n discuri, cu diametre distincte. Fiecare disc sa poate afla doar pe un disc de diametru mai mare.
Să se transfere discurile, câte unul, de pe A pe C, respectând regula de amplasare (orice disc se pune doar pe un disc mai mare), folosind în calitate de tijă ajutătoare B. (**hanoi(A,C,B,n-1)**)

Primele n-1 discuri se consideră ca un "megadisc", care poate fi transferat direct prin o "megamutare" (aceeaşi problemă de dimensiune n-1)

Prin o "megamutare" se deplasează n-1 discuri de pe tija A pe tija B. (**hanoi(A,B,C,n-1)**)

Prin o mutare simplă se deplasează 1 disc de pe tija A pe tija C.

Prin o "megamutare" se deplasează n-1 discuri de pe tija B pe tija C. (**hanoi(B,C,A,n-1)**)

E4:

Continuare

Exercițiu: scrieți propria funcție muta (int *x, int * y) care deplasează primul element nenul din x în ultimul element cu valoare nulă din y.

```
void hanoi(int *a, int *b, int *c, int n)
{
  if (n == 1)
            { muta(a, b); print (m);}
  else
            { hanoi (a, c, b, n - 1);
              muta (a, b);
              print(m);
              hanoi(c, b, a, n - 1);
            }
 }


void main()
{
   printf("\n N: "); scanf("%d", &m);
   for (i=0; i < m ; i++)
             {a[i] = i + 1; b[i] = c[i] = 0;}
   print(m);
   hanoi(a, b, c, m);
}
```

Headers

The question:

is it possible to create your own header file?

The answer is **yes**. header files are simply files in which you can declare your own functions that you can use in your main program or these can be used while writing large C programs.

NOTE: Header files generally contain definitions of data types, function prototypes and C preprocessor commands.

Creating myhead.h : Write the functions to be included code and then save the file as **myhead.h** or you can give any name but the extension should be .h indicating its a header file.

Including

Including the .h file in other program : Now as we need to include stdio.h as #include in order to use printf() function. We will also need to include the above header file myhead.h as **#include”myhead.h”**. The ” ” here are used to instructs the preprocessor to look into the present folder and into the standard folder of all header files if not found in present folder. So, if you wish to use angular brackets instead of ” ” to include your header file you can save it in the standard folder of header files otherwise. If you are using ” ” you need to ensure that the header file you created is saved in the same folder in which you will save the C file using this header file.

Important Points:

The creation of header files are needed generally while writing large C programs so that the modules can share the function definitions, prototypes etc.

Function and type declarations, global variables, structure declarations and in some cases, inline functions; definitions which need to be centralized in one file.

In a header file, do not use redundant or other header files; only minimal set of statements.

Don’t put function definitions in a header. Put these things in a separate .c file.

Include Declarations for functions and variables whose definitions will be visible to the linker. Also, definitions of data structures and enumerations that are shared among multiple source files.

In short, Put only what is necessary and keep the header file concised.

More – in practice session