

# INFORMATICĂ

LIMBAJUL C



MATERIAL ELABORAT CORESPUNZÂND  
CERINTELOR DE BACALAUREAT 2016

© 2016 PRESSTERN SOLUTIONS

# Cuprins

<b>Gândirea algoritmică .....</b>	<b>1</b>
Problema căutării .....	1
Problema intrării într-un cabinet medical.....	3
<b>Structura unui program și a unei funcții C .....</b>	<b>5</b>
Reprezentarea numărului în binar .....	7
Transmiterea parametrilor în cadrul unei funcții.....	8
<b>Construcțiile de bază ale limbajului C .....</b>	<b>9</b>
Caracterele .....	9
Identificatorii.....	9
Cuvinte cheie.....	10
Tipuri de date.....	10
Tipuri aritmetice .....	11
Tipul întreg .....	11
Tipul flotant .....	12
Tipul caracter .....	13
Constante și variabile.....	13
Comentarii .....	15
Operatori.....	16
Operatori aritmetici .....	16
Operatorul de conversie explicită (cast) .....	17
Operatorul dimensiune (sizeof) .....	17
Operatorul condițional .....	18
<b>Structuri de date.....</b>	<b>20</b>
Lista liniară simplu înlănțuită .....	20
Structura de tip stivă (LIFO Last In First Out) .....	26
Structura de tip coadă.....	30
<b>Instrucțiuni C .....</b>	<b>33</b>
Instrucțiunea vidă.....	33
Instrucțiunea de atribuire.....	33
Instrucțiunea compusă .....	34
Instrucțiunea if .....	34

Instrucțiunea while .....	35
Cel mai mare divizor comun.....	36
Instrucțiunea do while .....	36
Instrucțiunea for .....	37
Programul C care calculează n! .....	37
Instrucțiunea return.....	38
O funcție care returnează maximul a 2 numere .....	38
Numărul de elemente ale unei șir.....	39
Comparația celor 2 șiruri.....	39
Concatenare .....	40
Instrucțiunea break.....	40
Funcția exit.....	41
Instrucțiunea continue.....	42
Numerarea cifrelor existente într-un șir .....	42
Instrucțiunea goto.....	43
Instrucțiunea switch.....	44
<b>Pointeri C .....</b>	<b>45</b>
Operatori specifici pointerilor .....	46
Pointeri și tablouri.....	47
Pointeri la funcții.....	49
Tipuri structurate de date .....	50
Structuri.....	50
Câmpuri de biți.....	51
Uniuni.....	52
Enumerări.....	53
<b>Funcții de bibliotecă .....</b>	<b>55</b>
Funcția printf .....	55
Funcția scanf .....	57
Funcții de conversie .....	59
Funcții generatoare de numere aleatoare.....	59
Funcții pentru operații cu caractere și șiruri de caractere.....	59
<b>Operații cu fișiere .....</b>	<b>62</b>
<b>Calcul Matriceal.....</b>	<b>65</b>
Produsul a două matrici .....	65
Inversa unei matrici .....	66
Metoda lui Gauss.....	69

<b>Metode de sortare .....</b>	<b>72</b>
Sortare ordinară.....	72
Sortare prin selecție (Selection Sort) .....	74
Sortarea prin inserție directă (Direct Insertion Sort) .....	77
Sortare prin inserție binară (Binary Insertion Sort).....	79
Sortare prin inserție directă folosind o santinelă .....	81
Sortarea prin metoda bulelor (Bubble Sort) .....	83
Sortare rapidă (Quick Sort) .....	85
Sortare prin interclasare (Merge Sort) .....	87
<b>Recursivitate.....</b>	<b>90</b>
Calculul valorii $n!$ .....	90
Algoritmul lui Euclid recursiv.....	90
Să se genereze primele $n$ numere din șirul lui Fibonacci .....	91
<b>Metoda backtracking.....</b>	<b>93</b>
Permutările .....	93
Problema aranjamentelor .....	95
Problema combinărilor.....	95
Problema reginelor .....	97
Problema labirintului.....	99
Problema calului .....	102
Problema mingii .....	104
<b>Metoda Divide et Impera.....</b>	<b>108</b>
Suma elementelor unui șir.....	108
Problema "Turnurilor din Hanoi".....	109
Elementul maxim într-un șir .....	110
Problema căutării binare .....	111
<b>Grafuri neorientate .....</b>	<b>113</b>
Implementarea parcurgerii în lățime pentru un graf neorientat ....	113
Implementarea parcurgerii în adâncime pentru un graf neorientat ...	115
Drumuri într-un graf .....	117
Grafuri ponderate .....	118
Graful hamiltonian .....	120
Grafuri euleriene.....	122
Implementarea unui graf utilizând matricea de adiacență....	125
Implementarea unui graf utilizând pointeri .....	127
Drumul optim într-un graf .....	128

# Gândirea algoritmică

## Problema căutării

Există și cazul care, pentru o aceeași problemă putem prezenta două soluții, în care una este mai rapidă ca alta. De pildă, fie un șir de numere naturale oarecare, de exemplu 4,2,10,1,8,15,7. Vrem să testăm dacă un număr dat (să zicem numărul 8) se află în această secvență sau nu.

O astfel de căutare, prin parcurgerea de la stânga la dreapta a întregii secvențe de numere, până se găsește numărul dorit sau se epuiează toate elementele din secvență, se numește căutarea secvențială. Dacă avem un șir  $S$  de  $n$  elemente (notat  $S[1..n]$ ), atunci căutarea secvențială a lui  $x$  în  $S$  se descrie prin:

Căutare\_secvențială( $x, S[1..n]$ ) înseamnă

Început

Fie  $elemental\_curent = primul\_element$ ;

Atât timp cât ( $elemental\_curent <> x$ ) și (poziția elementului  $current \leq$  poziția ultimului element ( $n$ )) execută

Început

Dacă  $elemental\_curent = x$  atunci mesaj("găsit")

Altfel treci la următorul element

Sfârșit

Sfârșit.

În continuare să presupunem că numerele erau deja ordonate crescător 1, 2, 4, 7, 8, 10, 15. În acest caz particular, căutarea secvențială a unui număr cum este 8 nu este prea eficientă, deoarece 8 se află în a doua jumătate a secvenței, deci ar fi de preferat să nu-l căutăm în prima jumătate. Acest procedeu este mai rapid:

- Dacă numărul din mijloc este mai mic decât numărul căutat, atunci căutăm a doua jumătate;
- Dacă numărul din mijloc este mai mare ca numărul căutat, atunci căutăm în prima jumătate;
- Dacă numărul din mijloc este egal cu numărul căutat, înseamnă că am găsit numărul în cauză și trebuie să oprim căutarea.

Căutarea în jumătatea aleasă se face tot la fel, deci se va înjumătăți și această zonă...

Procedul anterior se numește căutare binară, deoarece, de fiecare dată, o secvență de numere este divizată în două jumătăți. Putem descrie căutarea binară a unui număr  $x$  într-o secvență  $S[1..n]$  astfel.

Căutarea\_binară( $x, S[1..n]$ ) înseamnă

Început

Stabilește elemental\_curent ca fiind elemental din mijlocul secvenței;

Dacă  $n=1$  atunci

Dacă  $x = \text{elemental\_curent}$  atunci mesaj("găsit")

Altfel mesaj("negăsit")

Altfel

Început

Împarte secvența în cele două jumătăți ( $S[1..mijloc]$  și  $S[mijloc+1..n]$ );

Dacă  $\text{elemental\_curent} = x$  atunci mesaj("găsit")

Altfel

Dacă  $\text{elemental\_curent} < x$  atunci

Căutare\_binară( $x, S[mijloc+1..n]$ )

Altfel căutarea\_binară( $x, S[1..mijloc]$ )

Sfârșit

Sfârșit.

## Problema intrării într-un cabinet medical

Dacă un om vrea să intre pentru consult la un medic, atunci, mai întâi va bate la ușă: dacă medicul răspunde prin „poftim!”, atunci va intra, altfel va aștepta până când pacientul dinăuntru va ieși; abia după ce cabinetul va fi liber, va intra.

Intrare\_la\_medic înseamnă

Început

Bate\_la\_ușa;

Dacă\_răspunsul = "poftim!" atunci

Întră\_în\_cabinet;

Altfel

Început

Atât\_timp\_cât\_cabinetul\_este\_ocupat\_de\_alt\_pacient

execută

Citeste\_un\_ziar;

Sfârșit

Sfârșit.

O instrucțiune compusă este formată dintr-o secvență de instrucțiuni, încadrate de cuvinte început și sfârșit, care pot conține, la rândul lor, blocuri de alternativă și repetiție.

Programarea este tehnica realizării de algoritmi descriși prin proceduri și programe. Ea devine o artă, atunci când se folosesc cele trei elemente de structurate și se numește programare structurată. Pentru a vedea ce ar însemna programare nestructurată, să reconsiderăm procedura de intrare în cabinetul medical:

Intare\_la\_medic înseamnă

Început

Bate\_la\_ușa;

Dacă\_răspunsul = "poftim!" atunci intră\_în\_cabinet;

Altfel Început

Atât\_timp\_cât\_cabinetul\_este\_ocupat\_de\_alt\_pacient

execută

Citeste\_un\_ziar;  
Întră\_în\_cabinet  
Sfârșit  
Sfârșit.



# Instrucțiuni C

Instrucțiunile C implementează structurile de bază ale programării structurate:

- *structura secvențială* (instrucțiunea compusă)
- *structura alternativă* (instrucțiunea if)
- *structura repetitivă* condiționată anterior (instrucțiunea while) și condiționată posterior (instrucțiunea do while)

Toate instrucțiunile limbajului C se termină cu ‘;’ excepție făcând instrucțiunile care se termină cu } (instrucțiunea compusă și instrucțiunea switch) după care nu se pune ‘;’.

## Instrucțiunea vidă

Instrucțiunea vidă se reduce la caracterul ; și se folosește în acele construcții în care este necesară prezența unei instrucțiuni fără a fi nevoie să se realizeze anumite operații.

## Instrucțiunea de atribuire

Instrucțiunea de atribuire se obține punând ; după o expresie de atribuire. Deci o expresie devine o instrucțiune dacă este urmată de ; (punct și virgulă).

## Instrucțiunea compusă

Instrucțiunea compusă este o succesiune de declarații urmate de instrucțiuni, succesiune care se include între acolade. Dacă există declarații atunci ele definesc variabile care sunt valabile numai în cadrul instrucțiunii compuse respective.

### Instrucțiunea if

În limbajul C, instrucțiunea condițională de bază este instrucțiunea if. *Formatul general* al instrucțiunii if este:

```
if (expresie) instrucțiune1;  
else instrucțiune2;
```

*Efectul* acestei instrucțiuni este următorul: se evaluează expresia din paranteze. Dacă valoarea este adevărat (diferită de 0) atunci se execută instrucțiune1 altfel se execută instrucțiune2, apoi în ambele situații se trece la instrucțiunea următoare instrucțiunii if.

Deoarece instrucțiunea if testează valoarea numerică a unei expresii, următoarele construcții sunt echivalente:

```
if(expresie!=0) ...
```

```
if(expresie) ...
```

Exemplu:

```
if(x)  
  if(y)  
    printf("1");  
else  
  printf("2");
```

În acest caz, ramura else care tipărește valoarea 2 este legată de instrucțiunea if(y).

```
if(x)
{
    if (y) printf("1");
}
else
    printf("2");
```

În acest caz, datorită instrucțiunii compuse, ramura else care tipărește valoarea 2 aparține instrucțiunii if(x).

Orice instrucțiune de atribuire închisă între parantezele expresiei din cadrul instrucțiunii if este de fapt o expresie, a cărei valoare de adevăr este egală cu valoarea atribuită.

Exemplul:

```
if(i=0) printf("Variabila i are valoarea 0");
else printf("variabila i are o valoare diferită de 0");
```

este greșit deoarece în loc să se folosească operatorul == se folosește operatorul de atribuire iar în acest caz valoarea de adevăr a expresiei de atribuire este 0 (fals), deci întotdeauna se va afișa mesajul "variabila i are o valoare diferită de 0" indiferent de valoarea inițială a lui i.

## Instrucțiunea while

Instrucțiunea while implementează structura repetitivă condiționată anterior și are următorul *format general*:

```
while(expresie) instrucțiune;
```

*Efectul* acestei instrucțiuni este următorul: Se evaluează valoarea expresiei din paranteză. Dacă valoarea expresiei este diferită de 0, atunci se execută instrucțiunea și se revine la

evaluarea expresiei. Dacă valoarea expresiei este fals (0) atunci se trece la instrucțiunea următoare instrucțiunii while.

## Cel mai mare divizor comun

Să se scrie programul C care implementează algoritmul lui Euclid de determinare a celui mai mare divizor comun a două numere naturale.

```
#include<stdio.h>

void main(void)
{
    int a=18, b=12, r;
    while (r=a%b) a=b,b=r;
    //putem folosi operatorul virgulă
    //în locul unei instrucțiuni compuse
    printf("cmmdc este %d\n",b);
}
```

## Instrucțiunea do while

Instrucțiunea do while implementează structura repetitivă condiționată posterior. *Formatul general* al instrucțiunii este:

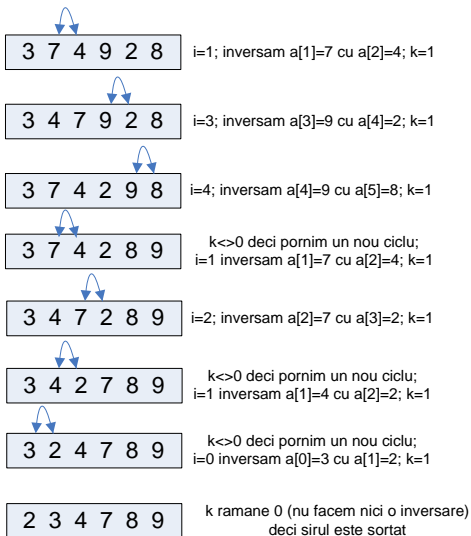
```
do
    instrucțiune;
while(expresie);
```

*Efectul* acestei instrucțiuni este următorul: Se execută instrucțiune după care se evaluează expresia. Dacă expresia este evaluată ca adevărat, se execută se revine la execuția instrucțiunii. Ciclarea continuă până când valoarea expresiei va fi fals (0), caz în care se trece la următoarea instrucțiune de după instrucțiunea do while.

# Metode de sortare

## Sortare ordinară

Cel mai simplu algoritm de sortare (dar și cel mai ineficient) se bazează pe următorul *principiu*: un șir este sortat dacă prin parcurgerea lui de la început până la sfârșit, fiecare element este mai mic decât succesorul. Dacă această condiție nu este îndeplinită, inversăm cele 2 elemente. Sortarea se încheie în momentul în care parcurgerea șirului se face fără a fi necesară nici o inversare (fiecare element este mai mic decât succesorul său).



```

#include<stdio.h>
#include<conio.h>

void sortare_ordinara(int a[], int n)
{
    int i,j,k,elem;
    do
    {
        for(i=0,k=0;i<n-1;i++)
            //la fiecare început de parcurgere a șirului, k este inițializat
            //cu 0
            if(a[i]>a[i+1])
                //dacă elementul curent este mai mare decât succesorul este
                //necesară inversarea
                //lor; atenție: dacă punem mai mare sau egal și avem 2
                //elemente egale, vom
                //ajunge la ciclu infinit
                {
                    elem=a[i];
                    a[i]=a[i+1];
                    a[i+1]=elem;
                    k=1;
                    //k diferit de 0 indică faptul că s-a făcut o inversare
                }
    } while(k);
    //sortarea se încheie în momentul în care k a rămas 0 (deci
    //nu s-a făcut nici o
    //inversare de elemente
}

void main(void)
{
    int n,a[100],i;
    printf("Introd dim sirului:");
    scanf("%d",&n);
    printf("Introd elem sirului:\n");
    for(i=0;i<n;i++)
    {
        printf("a[%d]=",i);
    }
}

```

```
scanf("%d",a+i);  
}  
sortare_ordinara(a,n);  
printf("Sirul sortat este: ");  
for(i=0;i<n;i++) printf("%d ",a[i]);  
putchar('\n');  
getch();  
}
```

# Metoda Divide et Impera

## Suma elementelor unui șir

Să se calculeze suma elementelor unui șir folosind Divide et Impera.

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

int a[20];

int divide(int ls, int ld)
{
    int mijloc,d1,d2;
    if(ls!=ld)
    {
        mijloc=(ls+ld)/2;
        d1=divide(ls,mijloc);
        d2=divide(mijloc+1,ld);
        return(d1+d2);
    }
    else
        return(a[ls]);
}

void main(void)
{
    int I,n;
    printf("Introd nr de elem ale sirului:");
    scanf("%d",&n);
    for(I=0;I<n;I++)
    {
        printf("a[%d]=");
```



```
scanf("%d",a+I);
}
printf("Suma elem este: %d\n",divide(0,n-1));
getch();
}
```

## Problema “Turnurilor din Hanoi”

Legenda acestei probleme spune că Brahma a fixat pe Pământ trei tije de diamante și pe una din ele a pus în ordine crescătoare 64 de discuri de aur de dimensiuni diferite, cu discul cel mai mare jos. Singura operațiune permisă călugărilor era mutarea a câte unui singur disc de pe o tijă pe alta, astfel încât niciodată să nu se pună un disc mai mare peste unul mai mic. Legenda spune că atunci când călugării vor muta toate cele 64 de discuri respectând regulile de maa sus, atunci va veni sfârșitul lumii. Presupunând că în fiecare secundă se mută un disc, lucrând fără întrerupere, cele 64 de discuri nu pot fi mutate nici în 500 de miliarde de ani de la începutul acțiunii!

Vom considera trei tije verticale notate Stânga, Mijloc, Dreapta. Pe tija Stânga se găsesc așezate  $n$  discuri de diametre diferite, în ordinea descrescătoare a diametrelor, privind de jos în sus. Inițial, tijele Mijloc și Dreapta sunt goale. Să se afișeze toate mutările prin care discurile de pe tija Stânga se mută pe tija Mijloc, în aceeași ordine, respectând următoarele reguli:

- la fiecare mișcare se mută un singur disc
- un disc mai mare nu poate fi plasat peste un disc cu diametrul mai mic
- un disc mai mare nu poate fi plasat peste un disc cu diametrul mai mic

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