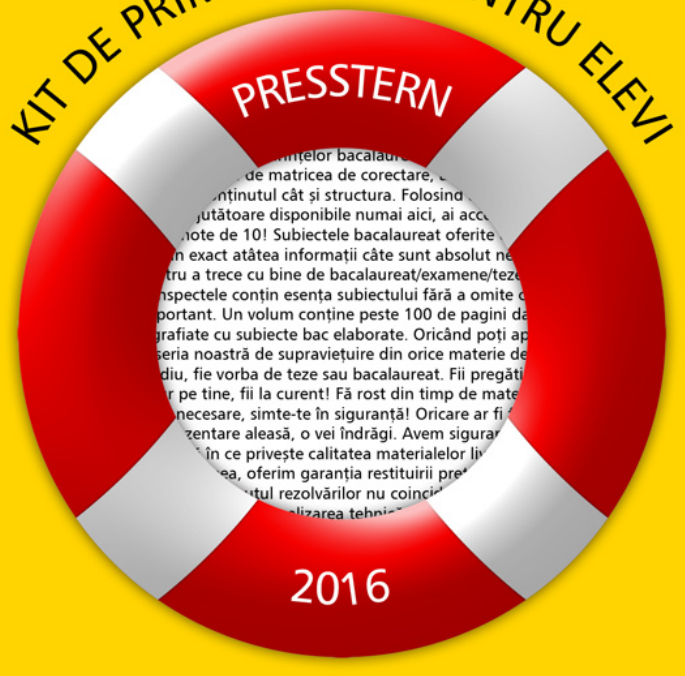
INFORMATICĂ

LIMBAJUL C

KIT DE PRIM AJUTOR PENTRU ELEVI



MATERIAL ELABORAT CORESPUNZÂND
CERINȚELOR DE BACALAUREAT 2016

Cuprins

Gândirea algoritmică

Problema căutării

Există şi cazul care, pentru o aceeaşi problem putem prezenta două soluţii, în care una este mai rapidă ca alta. De pildă, fie un şir de numere natural oarecare, de exemplu 4,2,10,1,8,15,7. Vream să testăm dacă un număr dat (să zivem numărul 8) se află în această secvenţă sau nu.

O astfel de căutare, prin parcurgerea de a stânga la dreapta a întregii secvenţe de numere, până se găseşte numărul dorit sau se epuiează toate elementele din secvenţă, se numeşte căutarea secvenţială. Dacă avem un şir S de n elemente (notat S[1..n]), atunci căutarea secvenţială a lui x în S se descrie prin:

```
Căutare_secventială(x,S[1..n]) înseamnă
Început
  Fie elemental_curent = primul_element;
  Atăt timp cât (elemental_curent <> x) si (pozitia elementului
current <= pozitia ultimului element (n)) execută
    Început
      Dacă elemental_curent = x atunci mesaj("găsit")
      Altfel treci la următorul element
    Sfârşit
Sfârşit.
```

În continuare să presupunem că numerele erau déjà ordonate crescător 1, 2, 4, 7, 8, 10, 15. În acest caz particular, căutarea secvenţială a unui număr cum este 8 nu este prea eficientă, deoarece 8 se află în a doua jumătate a secvenţei, 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 şi trebuie să oprim căutarea.

Căutarea în jumătatea aleasă se face tot la fel, deci se va înjumătăţi şi această zonă...

Procedeul anterior se numeşte căutare binară, deoarece, de fiecare dată, o secvenţă de numere este divizată în două jumătăţi. Putem descrie căutarea binară a unui număr x într-o secvenţă S[1..n] astfel.

```
Căutarea_binară(x, S[1..n]) înseamnă
Început
  Stabileste elemental_curent ca fiind elemental din mijlocul
secventei;
  Dacă n=1 atunci
    Dacă x = elemental_curent atunci mesaj("găsit")
    Altfel mesaj("negăsit")
  Altfel
    Început
      Împarte secventa în cele două jumătăti (S[1..mijloc] si
S[mijloc+1..n]);
      Dacă elemental_curent = x atunci mesaj("găsit")
      Altfel
        Dacă elemental_curent < x atunci
Căutare_binară(x,S[mijloc+1..n])
        Altfel căutarea_binară(x.S[1..mijloc])
    Sfârşit
Sfârşit.
```

Problema intrării într-un cabinet medical

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

```
Intrare_la_medic înseamnă
 Început
  Bate la_uşă;
  Dacă răspunsul = "poftim!" atunci
   Întră_în_cabinet;
  Altfel
   Început
     Atât timp cât cabinetul_este_ocupat_de_alt_pacient
execută
       Citeste_un_ziar;
    Sfârşit
Sfârşit.
```

O intrucţiune compusă este formată dintr-o secvenţă de intrucţiuni, încadrate de cuvinte început şi sfârşit, care pot conţine, la rândul lor, blocuri de alternativă şi repetiţie.

Programarea este tehnica realizării de algoritmi descrişi prin proceduri şi programe. Ea devine o artă, atunci când se folosesc cele trei elemente de structurate şi se numeşte 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_uşă;
 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
  Sfârşit
Sfârşit.
```

Instrucţiuni C

Instrucţiunile C implementează structurile de bază ale programării structurate:

- *structura secvenţială* (instrucţiunea compusă)
- structura alternativă (instrucţiunea if)
- *structura repetitivă* condiţionată anterior (instrucţiunea while) şi condiţionată posterior (instrucţiunea do while)

Toate instrucţiunile limbajului C se termină cu ‘;’ excepţie făcând instrucţiunile care se termină cu } (instrucţiunea compusă şi instrucţiunea switch) după care nu se pune ‘;’.

Instrucţiunea vidă

Instrucţiunea vidă se reduce la caracterul ; şi se foloseşte în acele construcţii în care este necesară prezenţa unei instrucţiuni fără a fi nevoie să se realizeze anumite operaţii.

Instrucţiunea de atribuire

Instrucţiunea de atribuire se obţine punând ; după o expresie de atribuire. Deci o expresie devine o instrucţiune dacă este urmată de ; (punct şi virgulă).

Instrucţiunea compusă

Instrucţiunea compusă este o succesiune de declaraţii urmate de instrucţiuni, succesiune care se include între acolade. Dacă există declaraţii atunci ele definesc variabile care sunt valabile numai în cadrul instrucţiunii compuse respective.

Instrucţiunea if

În limbajul C, instrucţiunea condiţională de bază este instrucţiunea if. *Formatul general* al instrucţiunii if este:

```
if (expresie) instrucţiune1;
else instrucţiune2;
```

Efectul acestei instrucţiuni este următorul: se evaluează expresia din paranteze. Dacă valoarea este adevărat (diferită de 0) atunci se execută instrucţiune1 altfel se execută instrucţiune2, apoi în ambele situaţii se trece la instrucţiunea următoare instrucţiunii if.

Deoarece instrucţiunea if testează valoarea numerică a unei expresii, următoarele construcţii sunt echivalente:

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

if(expresie) ...
```

Exemplu:

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

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

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

În acest caz, datorită instrucţiunii compuse, ramura else care tipăreşte valoarea 2 aparţine instrucţiunii if(x).

Orice instrucţiune de atribuire închisă între parantezele expresiei din cadrul instrucţiunii 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 greşit deoarece în loc să se folosească operatorul == se foloseşte operatorul de atribuire iar în acest caz valoarea de adevăr a expresiei de atribuire este 0 (fals), deci întotdeauna se va afişa mesajul “variabila i are o valoare diferită de 0” indiferent de valoarea iniţială a lui i.

Instrucţiunea while

Instrucţiunea while implementează structura repetitivă condiţionată anterior şi are următorul *format general*:

```
while(expresie) instrucţiune;
```

Efectul acestei instrucţiuni este următorul: Se evaluează valoarea expresiei din paranteză. Dacă valoarea expresiei este diferită de 0, atunci se execută instrucţiunea şi se revine la

evaluarea expresiei. Dacă valoarea expresiei este fals (0) atunci se trece la instrucţiunea următoare instrucţiunii 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 instrucţiuni compuse
  printf("cmmdc este  %d\n",b);
}
```

Instrucţiunea do while

Instrucţiunea do while implementează structura repetitivă condiţionată posterior. *Formatul general* al instrucţiunii este:

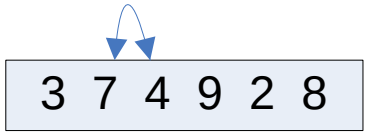
```
do
  instrucţiune;
while(expresie);
```

Efectul acestei instrucţiuni este următorul: Se execută instrucţiune după care se evaluează expresia. Dacă expresia este evaluată ca adevărat, se execută se revine la execuţia instrucţiunii. Ciclarea continuă până când valoarea expresiei va fi fals (0), caz în care se trece la următoarea instrucţiune de după instrucţiunea do while.

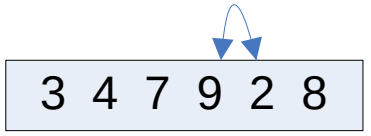
Metode de sortare

Sortare ordinară

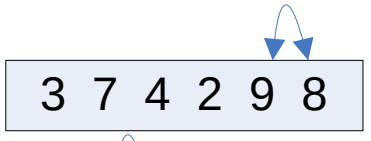
Cel mai simplu algoritm de sortare (dar şi cel mai ineficient) se bazează pe următorul *principiu*: un şir este sortat dacă prin parcurgerea lui de la început până la sfârşit, fiecare element este mai mic decât succesorul. Dacă această condiţie nu este îndeplinită, inversăm cele 2 elemente. Sortarea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



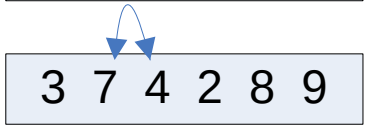
i=1; inversam a[1]=7 cu a[2]=4; k=1



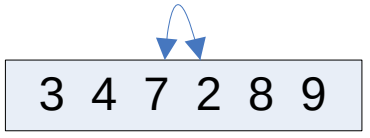
i=3; inversam a[3]=9 cu a[4]=2; k=1



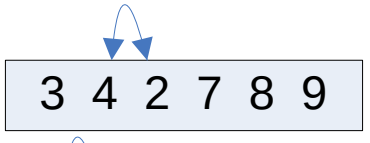
i=4; inversam a[4]=9 cu a[5]=8; k=1



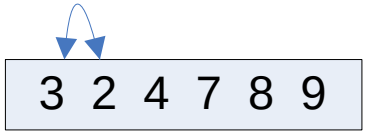
k<>0 deci pornim un nou ciclu;
i=1 inversam a[1]=7 cu a[2]=4; k=1



i=2; inversam a[2]=7 cu a[3]=2; k=1



k<>0 deci pornim un nou ciclu;
i=1 inversam a[1]=4 cu a[2]=2; k=1



k<>0 deci pornim un nou ciclu;
i=0 inversam a[0]=3 cu a[1]=2; k=1

k ramane 0 (nu facem nici o inversare)
deci sirul este sortat

```
#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 şirului,  k este iniţializat
cu 0
  if(a[i]>a[i+1])
  //dacă elementul curent este mai mare decât succesorul este
necesară inversarea
  //lor; atenţie: dacă punem mai mare sau egal şi 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 şir

Să se calculeze suma elementelor unui şir 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 şi pe una din ele a pus în ordine crescătoare 64 de discuri de aur de dimensiuni diferite, cu discul cel mai mare jos. Singura operaţiune 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 sfârşitul 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 acţiunii!

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

- la fiecare mişcare 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>
```