INFORMATICĂ

LIMBAJUL C

MATERIAL ELABORAT CORESPUNZÂND CERINȚELOR DE BACALAUREAT 2016

FOLOSIREA FIȚUICILOR ESTE O FRAUDĂ.
NU RECOMANDĂM UTILIZAREA LOR ÎN TIMPUL EXAMENELOR!

Cuprins

Gândirea algoritmică 1–3
Problema căutării.....1–2
Problema intrării într-un cabinet medical 2–3

Structura unui program şi a unei funcţii C 4–7
Reprezentarea numărului în binar.....5–6
Transmiterea parametrilor în cadrul unei funcţii 6–7

Construcţiile de bază ale limbajului C.....7–17
Caracterele.....7
Identificatorii.....8
Cuvinte cheie.....8
Tipuri de date 8–9
Tipuri aritmetice.....9
Tipul întreg.....10
Tipul flotant.....10
Tipul caracter.....11
Constante şi variabile 11–13
Comentarii.....13–14
Operatori 14
Operatori aritmetici 14–15
Operatorul de conversie explicită (cast) 15
Operatorul dimensiune (sizeof) 16
Operatorul condiţional.....16–17

Structuri de date.....18–31
Lista liniară simplu înlănţuită.....18–24
Structura de tip stivă (LIFO Last In First Out) 24–28
Structura de tip coadă.....29–31

Instrucţiuni C.....31–42
Instrucţiunea vidă 31
Instrucţiunea de atribuire.....32
Instrucţiunea compusă.....32
Instrucţiunea if.....32–33
Instrucţiunea while 34
Cel mai mare divizor comun.....34

Instrucţiunea do while 34–35
Instrucţiunea for 35–36
Programul C care calculează n!.....36
Instrucţiunea return.....36–38
O funcţie care returnează maximul a 2 numere.....37
Numărul de elemente ale unei şir.....37
Comparaţia celor 2 şiruri 38
Concateneare.....38
Instrucţiunea break 38–39
Funcţia exit.....40
Instrucţiunea continue 40–41
Numerarea cifrelor existente într-un şir 40–41
Instrucţiunea goto.....41–42
Instrucţiunea switch.....42

Pointeri C 43–52
Operatori specifici pointerilor 44–45
Pointeri şi tablouri 45–47
Pointeri la funcţii.....47–48
Tipuri structurate de date.....49–52
Structuri 49
Câmpuri de biţi 50
Uniuni 51
Enumerări 51–52

Funcţii de bibliotecă 53–58
Funcţia printf.....53–54
Funcţia scanf 54–56
Funcţii de conversie.....56–58
Funcţii generatoare de numere aleatoare.....57
Funcţii pentru operaţii cu caractere şi şiruri de caractere 57–58

Operaţii cu fişiere 58–60

Calcul Matriceal 61–68
Produsul a două matrici 61–62
Inversa unei matrici 62–65
Metoda lui Gauss.....66–68

Metode de sortare.....69–86

Sortare ordinară 69–71
Sortare prin selecţie (Selection Sort) 71–73
Sortarea prin inserţie directă (Direct Insertion Sort).....73–75
Sortare prin inserţie binară (Binary Insertion Sort) 76–78
Sortare prin inserţie directă folosind o santinelă.....78–79
Sortarea prin metoda bulelor (Bubble Sort).....80–81
Sortare rapidă (Quick Sort).....82–83
Sortare prin interclasare (Merge Sort) 84–86

Recursivitate 86–88
Calculul valorii n !.....86–87
Algoritmul lui Euclid recursiv.....87
Să se genereze primele n numere din şirul lui Fibonacci.....88

Metoda backtracking 89–103
Permutările 89–91
Problema aranjamentelor.....91
Problema combinărilor 91–92
Problema reginelor 93–94
Problema labirintului.....94–97
Problema calului.....98–100
Problema mingii 100–103

Metoda Divide et Impera 103–108
Suma elementelor unui şir 103–104
Problema “Turnurilor din Hanoi”.....104–106
Elementul maxim într-un şir.....106–107
Problema căutării binare.....107–108

Grafuri neorientate 108–124
Implementarea parcurgerii în lăţime pentru un graf neo. 108–110
Implementarea parcurgerii în adâncime pentru un graf neo. 110–112
Drumuri într-un graf.....112–113
Grafuri ponderate 114–116
Graful hamiltonian 116–118
Grafuri euleriene.....118–120
Implementarea unui graf utilizând matricea de adiacenţă ...121–123
Implementarea unui graf utilizând pointeri.....123–124
Drumul optim într-un graf 124

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 întră_în_cabinet;
  Altfel Început
    Atât timp cât cabinetul_este_ocupat_de_alt_pacient execută
      Citeste_un_ziar;
    Întră_în_cabinet
  Sfârşit
Sfârşit.
```

```
      push(val);
    else
      if(!val)
      {
        printf("Am extras elementul %d\n",pop());
        printf("Traversarea cozii: ");
        for(k=ind_d-1;k>=0;k--) printf("%d ",queue[k]);
        putchar('\n');
      }
  } while(val!=-1);
}
```

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.

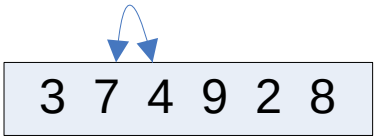
Metode de sortare

Sortare ordinară

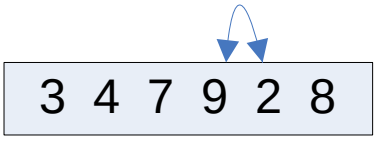
Cel mai simplu algoritm de sortare (dar şi cel mai ineficient) se bazează pe următorul *principiu*: un şir este sortat dacă prin parcurgerea lui de la început până la sfârşit, fiecare element este mai mic decât succesorul. Dacă această condiţie nu este îndeplinită, inversăm cele 2 elemente. Sortarea 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).

```
#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
```

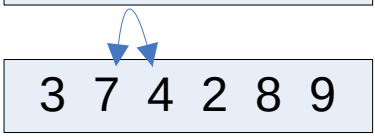


i=1; inversam a[1]=7 cu a[2]=4; k=1

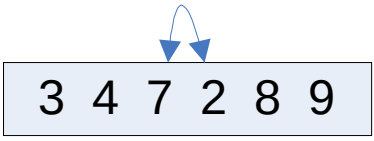


i=3; inversam a[3]=9 cu a[4]=2; k=1

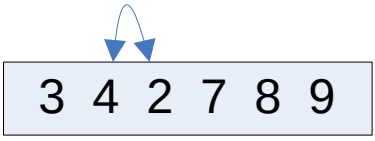
i=4; inversam a[4]=9 cu a[5]=8; k=1



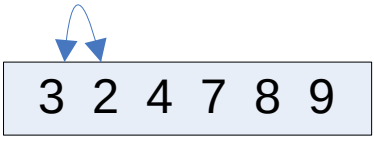
k<>0 deci pornim un nou ciclu;
i=1 inversam a[1]=7 cu a[2]=4; k=1



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

```
  {
    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();
}
```

Sortare prin selecţie (Selection Sort)

Principiul acestei metode este următorul: şirul este împărţit în 2 părţi: o “Parte stânga” sortată şi o “Parte dreapta” nesortată. Iniţial “Partea stângă” este vidă iar “Partea dreapta” conţine tot şirul. Se caută minimul din “Partea dreapta” şi se adaugă la sfârşitul “Părţii stânga”. Algoritmul se încheie în momentul în care “Partea dreaptă” mai conţine un singur element. Acest ultim element este maximul din şir deci, în final, “Partea stânga” va conţine toate elementele şirului sortate, iar “Partea dreapta” devine vidă.

```
    for(j=0;j<n;j++)
    {
      //drum va conţine drumul format către ieşire
      drum[i][j]=0;
      //tablou va conţine cotele
      printf("tablou[%d][%d]=",i,j);
      scanf("%d",&tablou[i][j]);
    }
  printf("Introd linia de pornire:");
  scanf("%d",&xi);
  printf("Introd coloana de pornire:");
  scanf("%d",&yi);
  //marcăm în drum punctul de pornire
  drum[xi][yi]=1;
  //apelăm funcţia recursivă pornind din punctul iniţial
  minge(xi,yi);
}
```

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