Eesti koolinoorte informaatika lahtine võistlus

3. detsember 1994. a.

II vooru lahendused kommentaaridega

Tulemused näitavad, et monelgi juhul on ülesanded osutunud pisut raskeks. Oma töö esimeselt leheküljelt leiate Teie lahenduste eest antud punktid, vasakul on "stiilipunktid" iga ülesande eest eraldi ja paremal neljanda liidetavana "stiilipunktide" keskmine. Tööle on lisatud parandajate kommentaare. Moned neist on programmis leitud vigade kohta. Monel juhul on aga lahenduse idee puudulik ja siis on toodud näiteid selliste algandmete kohta, millega Teie programm ilmselt hakkama ei saa. Loomulikult ei saa ka kommentaare lugedes alati selgeks, mis programmil tegelikult viga on ja kuidas teda tuleks korrigeerida. Sellepärast soovitame Teil oma programmi ise arvutil testida ja parandada. Kui näete, et Teie esialgse idee pohjal korralikku lahendust programmeerida ei onnestu, soovitame koostada uue programmi allpool antud soovituste järgi. Nüüd meie üldised selgitused ülesannete kohta:

ÜLESANNE 1.

Programm peab omama kolme "oskust":

1) mitmetäheliste identifikaatorite eraldamine stringist,

2) kordusteta väljastamine,

3) selliste identifikaatorite samastamine, mis erinevad ainult suure/väikese tähe poolest.

Koigepealt voiks direktiivi vasakult paremale läbi vaadata, salvestades leitud identifikaatorid stringide massiivi (esialgu kordustega).

Edasi saab selle massiivi üle vaadata, väljastades ainult need elemendid, mis massiivis eespool ei esine (suure/väikese tähe täpsusega). Muidugi voib viimase probleemi ka nii lahendada, et enne töötluse algust asendatakse näiteks koik suurtähed väikestega.

ÜLESANNE 2.

Selle ülesande lahendamiseks voib pakkuda kaks erinevat lihtsat algoritmi.

Esimene algoritm vaatleb koigepealt algpositsioonist 8-nda reani ulatuvat laienevat "koonust". Kui sihtpositsioon asub seal, siis on käikude arvuks reanumbrite vahe. Kui ei, siis tuleb 8 ja algpositsiooni reanumbri vahele liita 1, kui sihtpositsioon asub mone "koonuses" asuva 8-nda rea ruuduga ühel diagonaalil. Vastasel juhul tuleb sellele vahele liita 2 (on selge, et kabe saab suvaliselt ruudult suvalisele ruudule ülimalt kahe käiguga).

Teine algoritm on üldisem ja sobib ka paljude muude ülesannete lahendamiseks, kus otsitavaks on minimaalne sammude arv. Laua kujutamiseks votame 8x8 massiivi, milles igasse ruutu tahame paigutada algpositsioonist sinna joudmiseks vajaliku käikude arvu. Algul omistame muudele elementidele väärtuse -1 ja algpositsiooni kujutavale elemendile 0. Edasi

organiseerime tsükli, mille i-ndal läbimisel omistame väärtuse i koigile nendele ruutudele, mis asuvad väärtusega i-1 tähistatud ruutudest ühe kivikäigu kaugusel. Nii märgime ära koik ruudud, kuhu on voimalik jouda i sammuga. Seda tsüklit tuleb täita 8-algreanr korda. Järgmiste kahe i väärtusega tuleb läbida tsükkel, kus omistatakse väärtus i nendele ruutudele, kus seni on -1 ja kuhu on monest ruudust väärtusega i-1 voimalik jouda kabekäiguga. Loomulikult tuleb hoolitseda selle eest, et väärtusi ei omistataks üle laua ääre. Kui soovime, voime tsükli täitmise ajal kontrollida, kas omistasime juba väärtuse sihtpositsioonile, ja seejärel töö katkestada.

ÜLESANNE 3.

Sellele ülesandele anti palju ekslikke lahendusi. Pohimotteliselt töötas enamik koostatud programmidest järgmiselt: esimesest fraasist valiti esimene täht, igast järgmisest esimene selline täht, mida ees-pool veel polnud kasutatud. Kui mone fraasi jaoks enam tähte ei leidunud, siis eespool tehtud valikuid enam muutma ei hakatud ja anti negatiivne vastus. Praktiliselt koik programmid oleksid andnud negatiivse vastuse juba sisendandmete 'ab a' korral, sest esimesest fraasist valiti täht a. Nii naiivne üks olümpiaadiülesande lahendus ikka ka olla ei tohi. Tosine programmeerimine seisneb ju ikkagi selles, et programm peab ka igasuguste ebameeldivate algandmete korral oige vastuse andma.

Ülesande lahendamiseks tuleb tegelikult (halvemal juhul) läbi vaadata koik voimalikud valikud, s. t. juhul, kui tagapool tähti valida ei onnestu, tuleb eespool valikut muu-ta ja uuesti proovida. Loomulik programmeerimisvahend sellise koigi variantide läbivaatamisalgoritmi jaoks on rekursioon. Moeldav on aga kasutada ka tsüklit, kus igas fraasis proovitakse igat (eespool kasutamata) tähte. Väga paljudest sarnaste tähtedega fraasidest koosneva menüü korral voib programm töötada kaua.

Toome siin ühe PASCALis rekursiooni abil kirjutatud algoritmi:

```
Program VALIK;

Type String81=String[81];

Var RIDA, MENYY, VALITUD:String81; OK:Boolean;

Procedure PAIGUTA (RIDA, VALITUD:String81;

                   Var MENYY:String81; Var OK:Boolean);

   {Koostab MENYY stringi RIDA jaoks, kusjuures ei

    kasuta neid tähti, mis esinevad stringis VALITUD.

    OK näitab, kas onnestus}

Var I:Integer;

Begin

   While (RIDA&lt;&gt;'') and (RIDA[1]=' ') Do

          Delete(RIDA,1,1);

   If RIDA='' Then Begin MENYY:=''; OK:=True; Exit; End;
```

```
   For I:=1 To Pos(' ',RIDA)-1 Do
      If (RIDA[I]&lt;&gt;'_') And (Pos(RIDA[I],
                    VALITUD + Copy(RIDA,1,I-1))=0) Then
      Begin
         PAIGUTA(Copy(RIDA, Pos(' ', RIDA) + 1, 80),
                      VALITUD + RIDA[I], MENYY, OK);
         {proovime leida paigutuse stringi
          RIDA tagumise otsa jaoks}
         If OK Then
         Begin
            MENYY := Copy(RIDA,1,Pos(' ',RIDA))+MENYY;
            MENYY[I]:=Upcase(MENYY[I]); Exit;
         End;
      End;
End;
{------------ Pohiprogramm --------------}
Begin
Readln(RIDA);
While RIDA[Length(RIDA)]=' ' Do
        Delete(RIDA, Length(RIDA),1);
   RIDA:=RIDA+' '; PAIGUTA(RIDA,'',MENYY,OK);
   If OK Then Begin
      Writeln('Tulemus:');
      Writeln(MENYY);
   End
   Else Writeln('Ei saa');
End.
```