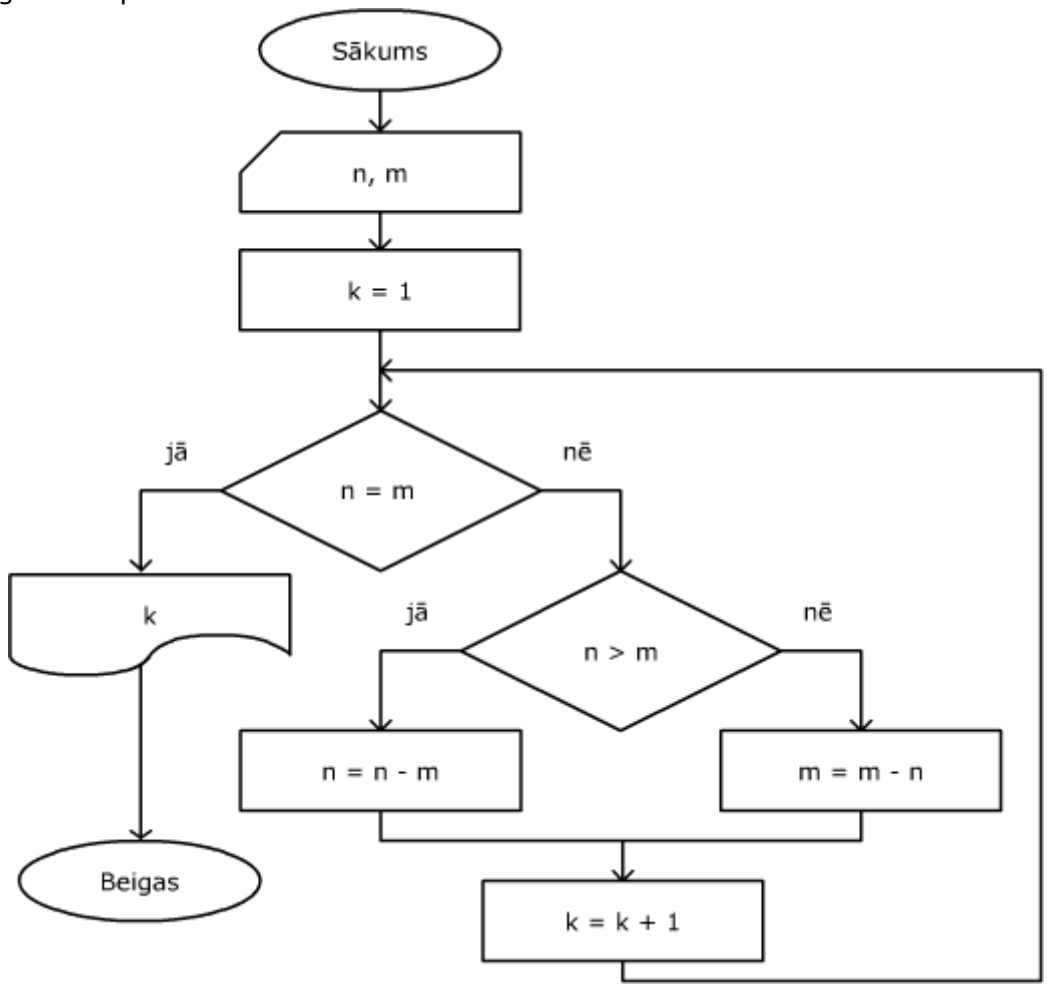
Liepājas Universitātes
Datorzinātņu olimpiādes
studentu programmēšanas konkursa uzdevumi

1. uzdevums [*uzd1*] (5 punkti)
Sastādi programmu pēc dotās blokshēmas.



2. uzdevums [*uzd2*] (5 punkti)
Uzzīmē blokshēmu vai sastādi algoritmu, pēc kura iespējams noteikt lielāko iespējamo elipšu skaitu, ko var uzzīmēt uz taisnstūra veida lapas, tā lai tās nekrustojas, ja *n* un *m* ir taisnstūra lapas malu garumi, bet *r1* un *r2* elipšu rādiusi.

3. uzdevums [*uzd3*] (5 punkti)
Aleksai ir kvadrātveida papīra lapa, kuras malu garumi ir *n*. Sākumā Aleksa lapu sagrieza četrās vienādās daļās, iegūstot četrus vienādus trijstūrus. Pēc tam, viņa katru trijstūri pārgrieza uz pusēm, iegūstot no katra trijstūra divus vienādus trijstūrus. Hm …, cik trijstūru iegūtu, ja vēl *n* reizes katru trijstūri pārgrieztu uz pusēm?
Sastādi programmu, kas aprēķina un izvada ekrānā Aleksas iegūto trijstūru skaitu un pēdējās pārgriešanas rezultātā iegūtā trijstūra laukumu, ja lietotājs ievada kvadrāta malas garumu *n* ($1 \le n \le 20$).

4. uzdevums [*uzd4*] (5 punkti)
Alise saņēma SMS ar tekstu "skeintsaP vet tilut sisenta ilutsev." un, izlasījusi to, neko nesaprata. Pēc brīža pie durvīm piezvanīja. Pastnieks bija atnesis viņai adresētu vēstuli, bet arī tās teksts bija nesaprotams. Meitene pārdomu pilna staigāja uz priekšu un atpakaļ pa istabu līdz ieraudzīja savu atspulgu spogulī un iesmējās, "Nu, protams, ...".
Sastādi programmu, kas palīdz Alisei izlasīt vēstules tekstu, kas ierakstīts teksta datnē *uzd4.in*. Rezultātu saglabā teksta datnē *uzd4.out*.

5. uzdevums [*uzd5*] (10 punkti)
Sastādi programmu, kas realizē doto algoritmu.

1.solis. Lietotājs ievada divus veselus skaitļus – *n* un *m*, intervālā no 1 līdz 10.
2.solis. Masīva A[*n*] vērtības aizpilda ar nejaušiem skaitļiem intervālā no 0 līdz 1.
3.solis. Masīva B[*m*] vērības aizpilda ar nejaušiem skaitļiem intervālā no 0 līdz 1.
4.solis. Masīva C[*k*] vērības aizpilda ar 0.
5.solis. Masīva C[*k*] elementam piešķir masīva A[*n*], B[*m*] un C[*k*] elementa summu.
6.solis. Ja masīva C[*k*] elementa vērtība ir lielāka nekā 1, masīva C[*k*] elementa vērtību samazina par 2 un masīva C[*k*-1] elementa vērtību aizstāj ar 1.
7.solis. *n* vērtību samazina par 1.
8.solis. *m* vērtību samazina par 1.
9.solis. *k* vērtību samazina par 1.
10.solis. Ja *k* vērtība ir lielāka nekā 0, pāriet pie 5.soļa izpildes, citādi pāriet pie 11.soļa izpildes.
11.solis. Ekrānā izvada masīva A[*n*] elementus.
12.solis. Ekrānā izvada pluss zīmi.
13.solis. Ekrānā izvada masīva B[*m*] elementus.
14.solis. Ekrānā izvada vienādības zīmi.
15.solis. Ekrānā izvada masīva C[*k*] elementus.

Ko dara (ko spēj aprēķināt) dotais algoritms?
Kādai ir jābūt *k* sākuma vērtībai attiecībā pret *n* un *m* sākuma vērtību?

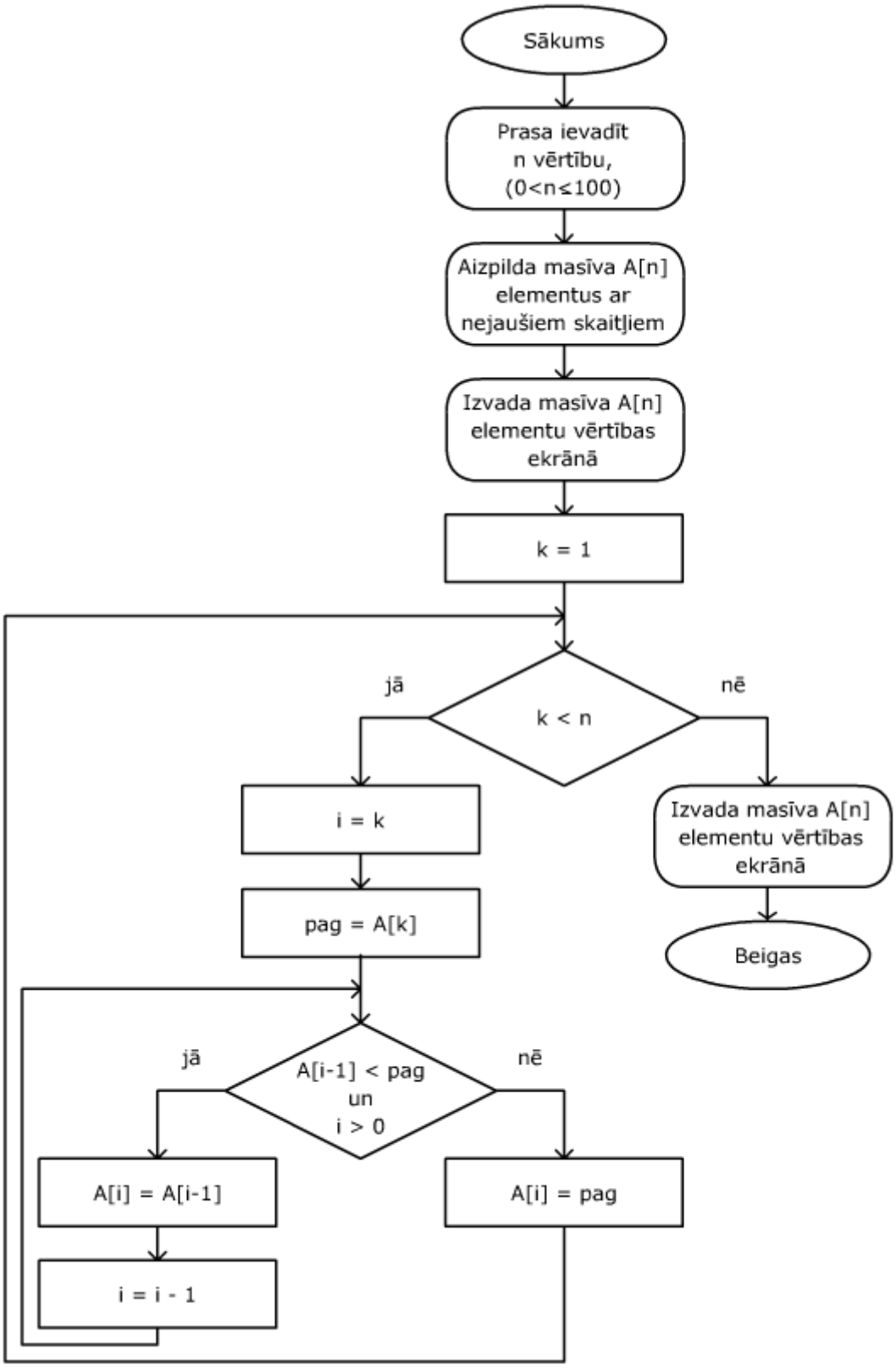
6. uzdevums [*uzd6*] (15 punkti)
Maizniekam ir ticējums, lai kūka sanāktu perfekti simetriska, tā no cepeškrāsns jāizņem brīdī, kad pulkstenis rāda palindroma laiku, tas ir, pulksteņa rādījums, lasīts no kreisās uz labo pusi, ir vienāds ar pulksteņa rādījumu, kas iegūts, ja to lasa no labās uz kreiso pusi.
Sastādi programmu, kas nosaka un izvada ekrānā tuvāko palindroma laiku, ja zināms pulksteņa rādījums (ievada lietotājs formā HH:MM) brīdī, kad kūka tiek ielikta cepeškrāsnī. Piemēram,

- ja lietotājs ievada 00:00, programma izvada 01:10;
- ja lietotājs ievada 12:34, programma izvada 13:31;
- ja lietotājs ievada 14:15, programma izvada 14:41;
- ja lietotājs ievada 23:57, programma izvada 01:10.

7. uzdevums [*uzd7*] (10 punkti)
Sastādi programmu, kas izvada ekrānā *n* x *n* tabulas vērtības. Tabulas vērtības veidojas pēc piemērā redzamā principa. Lietotājs ievada *n* vērtību, pie tam, *n* vērtība var būt tikai skaitlis intervālā [1;20].
Piemēram, ja *n* = 7, tabulas vērtības ir sekojošas:

1	**2**	**3**	**4**	**5**	**6**	0
1	**2**	**3**	**4**	**5**	0	**1**
1	**2**	**3**	**4**	0	**2**	**1**
1	**2**	**3**	0	**3**	**2**	**1**
1	**2**	0	**4**	**3**	**2**	**1**
1	0	**5**	**4**	**3**	**2**	**1**
0	**6**	**5**	**4**	**3**	**2**	**1**

8. uzdevums [*uzd8*] (10 punkti)
Sastādi programmu pēc dotās blokshēmas.



9. uzdevums [*uzd9*] (10 punkti)
Kādā karaļvalstī dzīvo pūķis – cilvēkēdājs. Viņam ir *n* galvas un *m* astes. Princis ir nolēmis "pielikt punktu" pūķa izdarībām un nogalināt to. Taču to izdarīt nemaz nav tik vienkārši, jo,

- ja pūķim ar vienu zobena cirtienu nocērt vienu galvu, tās vietā ataug jauna;
- ja nocērt vienu asti, tās vietā ataug divas astes;
- ja nocērt divas astes, tās vietā ataug 1 galva;
- ja nocērt divas galvas, to vietā neataug nekas.

Sastādi programmu, kas nosaka, vai princim izdevās uzvarēt pūķi, ja teksta datnes *uzd9.in* pirmajā rindā doti trīs naturāli skaitļi, kas atdalīti ar atstarpēm: *n* – pūķa galvu skaits ($0<n\leq100$), *m* – pūķa astu skaits ($0<m\leq100$), *k* – prinča izdarīto cirtienu skaits ($0<k\leq100$), bet otrajā rindā dotas *k* komandas, kas atdalītas ar atstarpēm, kur G1 nozīmē, ka princim izdodas nocirst vienu pūķa galvu, G2 – 2 galvas, A1 – vienu asti un A2 – divas astes. Cīņas rezultātu izvadīt teksta datnē *uzd9.out* sekojošā veidā: ja pūķis uzvarēts, izvadīt tekstu "UZVARA"; ja cīņa zaudēta, izvadīt divus skaitļus, kas atdalīti ar atstarpi – pūķa galvu skaits un pūķa astu skaits cīņas beigās.

10. uzdevums [*uzd10*] (15 punkti)
Ralfs skolas kursā "Programmēšanas pamati" šobrīd apgūst tēmu "Saspiešanas algoritmi". Skolotājs viņam ir iedevis programmas piemēru, kas realizē datu saspiešanu pēc RLE algoritma, tas ir, ja sākotnējā simbolu virknē, kāds simbols atkārtojas vairākas reizes, tad saspiestajā simbolu virknē to raksta vienu reizi, un aiz simbola raksta skaitli, kas norāda, cik reizes simbols atkārtojas. Piemēram, simbolu virkni **aaaaabbbacccbddddd** saspiestā veidā pierakstīta **a5b3ac3bd5**.
Palīdzi Ralfam uzrakstīt programmu, kas no saspiestās simbolu virknes, kas ierakstīta teksta datnē *uzd10.in*, atjauno oriģinālo simbolu virkni un ieraksta to datnē *uzd10.out*.

Skolotāja dotās C++ programmas kods

```
#include <fstream>
using namespace std;
int main(){
  ifstream dati1;
  ofstream dati2;
  dati1.open("uzd10.in",ios::in);
  dati2.open("uzd10.out",ios::out);
  char s1,s2;
  int sk;
  dati1.get(s1);
  while(!dati1.eof()){
     sk=1;
     dati1.get(s2);
     while(s1==s2&&!dati1.eof()){
        sk++;
        dati1.get(s2);
     }
     if(sk==1)
        dati2<<s1;
     else
        dati2<<s1<<sk;
     s1=s2;
  }
  dati1.close();
  dati2.close();
}
```