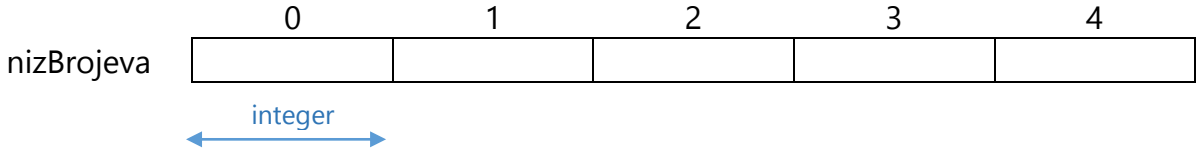
Nizovi

Nizovi

Nizovi predstavljaju složene tipove podataka. Niz predstavlja seriju elemenata istog tipa tačno određene dužine. Broj elemenata niza predstavlja dužinu niza. Svaki element ima određenu poziciju u nizu i moguće mu je pojedinačno pristupiti korišćenjem jedinstvenog rednog broja (indeksa).

Niz nam omogućava da koristimo na primer 5 vrednosti celobrojnog tipa, bez potrebe da deklarišemo 5 pojedinačnih celobrojnih promenljivih. Kako broj elemenata niza raste, ta prednost je izraženija.



Kao i kod običnih promenljivih, niz mora biti deklarisan pre nego što mu možemo pristupiti. Kada je deklarisanje nizovnih promenljivih i alokacija (zauzeće) memorije za niz u pitanju, u programskim jezicima postoje dva načina da se to uradi: *statička alokacija* i *dinamička alokacija*.

Statička alokacija podrazumeva da je u vreme pisanja programa poznata maksimalna dužina niza koja nam može biti potrebna, i u tom slučaju će program pri pokretanju rezervisati neophodan prostor za niz zahtevane veličine. Nasuprot tome, kod dinamičke alokacije maksimalan broj elemenata niza ne mora biti poznat u vreme pisanja programa, već je dovoljno znati ga u vreme izvršavanja programa. Kod ovog tipa alokacije, memorija za smeštanje niza se alocira tokom izvršavanja programa kada nam niz postane potreban.

Različiti programski jezici podržavaju jedan ili oba pristupa. Pascal podržava samo statičku alokaciju, C/C++ podržavaju i statičku i dinamičku alokaciju, dok C# podržava samo dinamičku alokaciju. Za potrebe ovog kursa generalno će biti dovoljno koristiti statičku alokaciju, jer će ograničenja veličine biti poznata u napred i zadata u tekstovima zadataka.

U nastavku ćemo za svaki od programskih jezika posebno objasniti kako se u njima alociraju nizovi i kako se indeksiraju elementi niza.

Nizovi u Pascalu

U Pascalu niz elemenata nad osnovnim tipom T, koji će se indeksirati elementom iz intervala Q se zadaje u formi:

```
Array [Q] of T
```

Na ovaj način je moguće deklarisati promenljivu direktno u `var` sekciji. Pored toga, takođe je moguće i prvo kreirati novi tip u `type` sekciji, pa ga potom iskoristiti za deklaraciju promenljive u `var` sekciji kao u sledećem primeru.

```
program deklaracija_niza(input, output);
type
      vector = Array [1..100] of Double;
var
      a, b: vector;
begin
      . . .
end.
```

U primeru iznad je uveden novi tip `vector` koji predstavlja niz od 100 elemenata brojeva u pokretnom zarezu.

Indeksiranje pojedinačnih elemenata u nizu se vrši navođenjem imena nizovne promenljive i indeksa iz dozvoljenog opsega:

```
promenljiva[indeks]
```

Kroz sledeći zadatak ćemo videti na primeru kako se nizovi alociraju i indeksiraju u Pascalu.

Problem 1

Potrebno je u programskom jeziku Pascal implementirati program koji izračunava i inicijalizuje prvih 100 elemenata niza $a_i = i^2$, za i od 1 do 100.

Zadatak ćemo rešiti tako što ćemo u petlji izračunavati vrednosti svakog od elemenata i dodeljivati tu vrednost elementima niza a.

```
program nizA (input, output);
var
      a : Array [1..100] of integer;
      i : integer;
begin
      for i:= 1 to 100 do
            a[i] := i * i;
end.
```

Ovde vidimo da je dozvoljeni opseg indeksa za niz a između 1 i 100. U Pascalu je praksa da se nizovi indeksiraju sa početkom od indeksa 1, međutim korisnika ništa ne sprečava da napiše program gde će se niz indeksirati od 0, ili čak nekog drugog broja, npr. -5.

Nizovi u C/C++

Promenljiva nizovnog tipa, čiji su elementi tipa T i dužina niza L se deklariše kao:

```
T niz[L];
```

Pored toga moguće je uvesti novi nizovni tip korišćenjem typedef naredbe, i potom iskoristiti novouvedeni tip:

```
typedef int customArray[100];
customArray arrayVar;
```

Indeksiranje elemenata niza se vrši tako što se navede ime nizovne promenljive a zatim validna vrednost indeksa:

```
promenljiva[indeks]
```

Treba uočiti da za razliku od Pascala gde se pri deklaraciji niza navodio validan opseg, u C-u i C++ se navodi samo dužina niza, a podrazumeva se da se niz indeksira indeksima u opsegu od 0 do (<dužina_niza>-1).

Pored toga C i C++ se razlikuju od Pascala i C# po tome što ne proveravaju da li je indeks korektan . Zbog toga treba posebno biti pažljiv pri indeksiranju elemenata niza, jer ako zadamo indeks koji je van opsega za dati niz (manji od 0, ili veći od N-1, gde je N dužina niza), program će pokušati da pristupi memorijskoj lokaciji ispred ili iza one na kojoj je niz alociran i pročitaće ili upisati podatak na tu lokaciju. Ta greška je vrlo komplikovana za otkrivanje, jer se može manifestovati na različite načine u zavisnosti od toga koji podatak je bio na lokaciji kojoj smo greškom pristupili.

Korišćenje nizova u C/C++ ćemo demonstrirati kroz sledeći zadatak.

Problem 2

Potrebno je u programskom jeziku C/C++ implementirati program koji izračunava i inicijalizuje prvih 100 elemenata niza $a_i = i^2$, za i od 1 do 100.

Zadatak ćemo opet rešiti tako što ćemo u petlji izračunavati vrednosti svakog od elemenata i dodeljivati tu vrednost elementima niza a. U nastavku je dato rešenje sa statičkom alokacijom.

```
int main()
{
      int a[100];
      for (int i=1; i <= 100; i++)
            a[i-1]=i*i;

      return 0;
}
```

U rešenju ovog zadatka možemo videti da smo zbog toga što se nizovi indeksiraju od 0, morali da prilagodimo indeks i da oduzmemo 1 od i. Alternativno rešenje bi bilo da alociramo niz od 101 elementa i da ne koristimo element sa indeksom 0, a mogli bismo direktno da koristimo i. Ali generalno kada koristimo nizove u C/C++ treba obratiti pažnju kako indeksiramo elemente niza i posebno treba obratiti pažnju na granične slučajeve, kada se indeksiraju prvi i poslednji element niza.

U dosadašnjim primerima smo koristili statičko alociranje nizova, a sada ćemo preći na dinamičko alociranje. Za dinamičko alociranje nizova koristi se komanda `new`, a nizovna promenljiva malo drugačije deklariše:

```
int *niz = new int[Len];
```

Na ovaj način smo alocirali niz celih brojeva dužine `Len`, gde `Len` može biti celobrojna promenljiva čiju smo vrednost mogli prethodno izračunati negde u programu, dakle ne mora biti konstanta. *Ono na šta treba posebno obratiti pažnju je da kada u jezicima C ili C++ na ovaj način alociramo niz, potrebno je kasnije u programu (kada završimo sa korišćenjem niza) da komandom delete oslobodimo memoriju koju smo alocirali za niz*. U suprotnom će ta memorija ostati trajno zauzeta dok naš program radi. Ovakva pojava se naziva curenje memorije i česta je prateća boljka programa koji dinamički alociraju memoriju. Iz tog razloga se preporučuje statička alokacija ukoliko ju je moguće koristiti.

Kada se niz dinamički alocira, on se koristi i indeksira na isti način kao i statički alociran niz, a na kraju se dealocira (oslobađa) korišćenjem `delete` komande:

```
delete[] niz;
```

Ukoliko bi se za rešavanje Problema 2 koristila dinamička alokacija rešenje zadatka bi izgledalo ovako:

```
int main()
{
      int a*;
      a = new int[100];

      for (int i=1; i <= 100; i++)
            a[i-1]=i*i;

      delete[] a;

      return 0;
}
```

Napomena: Komande `new` i `delete` su deo C++ i ne postoje u jeziku C gde se za dinamičku alokaciju koriste komande `malloc` i `free`, ali o njima ovde neće biti reči budući da su ih komande `new` i `delete` zamenile u praksi.

Nizovi u C#

Programski jezik C# podržava samo dinamičku alokaciju nizova, pa se niz dužine L, čiji su elementi tipa T u ovom jeziku alocira na sledeći način:

```
T niz[] = new T[L];
```

Kako C# ima komponentu koja vodi računa o dinamički alociranoj memoriji i oslobađa je kada ustanovi da se ona više ne koristi, to znači da u C# ne moramo da brinemo o problemu oslobađanja memorije, pa samim tim ni o curenju memorije.

Što se tiče indeksiranja elementa niza, ono se vrši navođenjem imena nizovne promenljive i indeksa elementa:

```
promenljiva[indeks]
```

Za dozvoljeni opseg indeksa za niz u C# važi isto što i za nizove u C/C++. Elementi niza se indeksiraju od 0 do (<dužina_niza>-1).

Na sledećem zadatku možemo videti kako se nizovi koriste u C#.

Problem 3

Potrebno je u programskom jeziku C# implementirati program koji izračunava i inicijalizuje prvih 100 elemenata niza $a_i = i^2$, za i od 1 do 100.

Zadatak ćemo opet rešiti tako što ćemo u petlji izračunavati vrednosti svakog od elemenata i dodeljivati tu vrednost elementima niza a.

```
using System;

public class ArrayDemo
{
   public static void Main()
   {
      int a[] = new int[100];
      for (int i=1; i <= 100; i++)
            a[i-1]= i*i;
   }
}
```

Slično kao i kod C/C++ i ovde moramo da obratimo pažnju na indekse niza, s obzirom da se nizovi indeksiraju od 0.

Zadaci nizovi

Kroz par zadataka vezanih za nizove ćemo u nastavku pokazati kako se nizovi mogu koristiti, kako bi čitaocima bolje približili koncept nizova.

Za rešavanje zadataka će biti korišćen pseudo jezik koji je sličan Pascalu.

Problem 4

Napisati program koji računa skalarni proizvod dva vektora veličine do 100 elemenata.

Ako su data dva vektora a i b dužine L, tada se njihov skalarni proizvod računa kao:

$$\sum_{i=1}^{L} a_i \cdot b_i$$

Skalarni proizvod ćemo računati u petlji sumirajući proizvod elemenata sa istim indeksom.

```
10    integer L, i
20    Array[1..100] of float a, b
30    float sproduct

40    read(L)
50    for i= 1 to L do
60          read (a[i])
70    for i= 1 to L do
80          read (b[i])

90    sproduct = 0
100   for i= 1 to 100 do
110         sproduct = sproduct + a[i]*b[i]

120   write (sproduct)
```

Problem 5

Zadat je niz celih brojeva A čija je dužina manja ili jednaka od 1000. Potrebno je naći indeks i vrednost člana niza koji je najbliži srednjoj vrednosti.

Zadatak se rešava tako što u prvom prolazu kroz niz izračunamo srednju vrednost niza, a zatim u drugom prolazu za svaki element niza proveravamo koliko je udaljen od aritmetičke sredine, i ako je manje udaljen od prethodno najbližeg onda pamtimo njegov indeks, u suprotnom samo prelazimo na sledeći element.

```
10    integer n, i, nearestI, sum
20    float avg, nearestDiff, diff
```

```
30    Array [1..1000] of integer a;

40    read (n)
50    for i= 1 to n do
60          read (a[i])

70    sum = 0
80    for i= 1 to n do
90          sum = sum + a[i]
100   avg = sum / (1.0 * n)
110   nearestI = 1
120   nearestDiff = abs(avg – a[1])
130   for i= 2 to n do
140         diff = abs(avg – a[i])
150         if diff < nearestDiff then
160               nearestDiff = diff
170               nearestI = i;
180         end if
190   end for

200   write (nearestI, a[nearestI])
```

Problem 6

Napisati program koji okreće naopačke niz celih brojeva maksimalne dužine 100, tj. na prvo mesto postavlja poslednji element niza i obrnuto, na drugo mesto predzadnji i obrnuto, itd.

Kako bi smo okrenuli niz naopačke dovoljno je da idemo do sredine niza i razmenimo vrednosti elemenata a[i] i a[L-i+1]. Ukoliko bismo na ovaj način prošli kroz ceo niz, tada bi niz ostao nepromenjen, jer bi pri prolasku kroz drugu polovinu niza vratili premeštene elemente na početna mesta. Takođe, ako je niz neparne dužine, srednji element nema potrebe da pomeramo, budući da on ostaje na svom mestu.

```
10    integer n, i, tmp
20    Array [1..100] of integer a

30    read (n)
40    for i= 1 to n do
50          read (a[i])

60    for i= 1 to (n/2) do
70          tmp = a[i]
80          a[i] = a[n-i+1]
90          a[n-i+1] = tmp;
100   end for
```

Problem 7

Napisati program koji izbacuje najmanji element niza iz niza celih brojeva. Dužina niza ne prelazi 5000 elemenata. Ukoliko se najmanji element ponavlja više puta potrebno je eliminisati sva njegova pojavljivanja.

Ovaj problem rešavamo tako što u prvom prolazu kroz niz izračunavamo minimum u nizu. U sledećem prolazu prepisujemo niz preko njega samog, ali tako da preskačemo i ne prepisujemo pojavljivanja minimuma. U toku drugog prolaza takođe računamo i novu dužinu niza.

```
10    integer n, newN, i, min
20    Array [1..5000] of integer a

30    read (n)
40    for i= 1 to n do
50          read (a[i])

60    min = a[1]
70    for i= 2 to n do
80          if a[i] < min then min = a[i]
90    end for

100   newN = 0
110   for i= 1 to n do
120         if a[i] != min then
130               newN = newN + 1
140               a[newN] = a[i]
150         end if
160   end for
180   n = newN
```

Problem 8

Dat je niz celih brojeva od kojih nikoja dva broja nisu jednaka. Potrebno je ispisati za zadati niz sve njegove podnizove. Podnizom se u ovom slučaju smatra podskup uzastopnih elemenata početnog niza. Smatrati da dužina početnog niza ne prelazi 1000 elemenata.

Zadatak se može rešiti tako što za svaki element u nizu ispišemo podnizove koji od njega počinju, a dužine su od 1 do N-i+1 elemenata. Na taj način ćemo pokriti sve podnizove tačno jedanput. To ćemo jednostavno uraditi tako što ćemo u jednoj petlji vrteti početne indekse za podniz, a u ugnježdenoj petlji krajnji indeks za podniz. Budući da su svi elementi u nizu jedinstveni, ne moramo da vodimo računa o tome da li se neka kombinacija ponavlja.

```
10    integer n, i, j, ii
20    Array [1..1000] of integer a

30    read (n)
40    for i= 1 to n do
50          read (a[i])

60    for i= 1 to n do
70          for j= i to n do
80                for ii= i to j do
90                      write (a[ii])
100               end for
110               writeline()
120         end for
130   end for
```

Višedimenzionalni nizovi

Pored jednodimenzionalnih nizova, koji imaju samo jedan indeks za indeksiranje elementa niza, programski jezici podržavaju i višedimenzionalne nizove. Među višedimenzionalnim nizovima, najčešći i najjednostavniji su dvodimenzionalni nizovi, koje često nazivamo matricama ili tabelama. U dvodimenzionalnom nizu prvi indeks određuje red u tabeli, a drugi kolonu u okviru tog reda. Na primer, programi koji rešavaju Sudoku ili ukrštene reči su primeri programa u kojima je pogodno koristiti dvodimenzionalne nizove. Takođe, radni list u programskom alatu Excel je dobar primer dvodimenzionalnog niza gde se ćelije indeksiraju pomoću slova koje određuje kolonu i broja koji određuje red.

Trodimenzionalni niz možemo zamisliti kao niz od više dvodimenzionalnih tabela, gde prvi indeks niza određuje koju od tabela posmatramo, drugi – koji red u tabeli posmatramo i treći indeks kolonu u okviru reda u kojoj se nalazi željeni element. Dobar primer za ovo je opet Excel koji u jednom dokumentu može da ima više radnih listova. U tom slučaju prvi indeks označava kom radnom listu pristupamo, zatim slovo određuje u kojoj koloni na tom listu se nalazi ćelija, a broj u kom redu. Takođe, kao primer za trodimenzionalni niz može da posluži i kalendar u nekoj aplikaciji koja beleži prosečne temperature za svaki dan. U tom slučaju prvi indeks predstavlja godina, drugi mesec u okviru godine, a treći dan u godini.

```
type yearlyTemps = Array [0..2020, 1..12, 1..31] of real
```

Moguće je definisati i nizove sa četiri i više dimenzija, ali se takvi nizovi ne sreću toliko često u praksi.

U nastavku ćemo slično kao i za jednodimenzionalne nizove, za svaki od programskih jezika posebno objasniti kako se u njima alociraju višedimenzionalni nizovi i kako se indeksiraju elementi tih nizova.

Višedimenzionalni nizovi u Pascalu

Višedimenzionalni niz sa elementima tipa T, sa N dimenzija koje su u intervalima Q_i se deklariše kao:

`Array [`Q_1`, `Q_2`, … , `Q_N`] of T`

Dok se element takvog niza indeksira pomoću imena nizovne promenljive i N indeksa:

`mdNiz[`i_1`][`i_2`]...[`i_N`]`

Na primeru sledećeg zadatka možemo videti kako se koriste multidimenzionalni nizovi.

Problem 9

Zadata je tabela sa prirodnim brojevima čije dimenzije ne prelaze 100 x 100. Potrebno je napisati program u programskom jeziku Pascal koji ispituje da li je zbir elemenata po redovima jednak u svim redovima.

Zadatak se može rešiti tako što nakon što se učitaju elementi tabele, saberemo sve elemente u prvom redu tabele. A zatim idemo red po red i sabiramo elemente po redovima i proveravamo i pamtimo da li je suma ista kao ona u prvom redu.

```
program SumRows(input, output);
var
      m, n, i, j, sumInit, sumCurr : integer;
      table : Array [1..100, 1..100] of integer;
      areEqual : boolean;
begin
      readln(n, m);
      for i:= 1 to n do
            for j:=1 to m do
                  read(table[i][j]);

      sumInit := 0;
      for j := 1 to m do
            sumInit := sumInit + table[1][j];

      areEqual := true;
      for i:= 2 to n do
      begin
            sumCurr := 0;
            for j:=1 to m do
                  sumCurr := sumCurr + table[i][j];
```

```
            areEqual := areEqual AND sumCurr = sumInit;
      end;

      if areEqual
            then writeln('Svi redovi imaju jednaku sumu')
            else writeln('Nisu svu jednaki');
end.
```

Višedimenzionalni nizovi u C/C++

Promenljiva nizovnog tipa, čiji su elementi tipa T sa N dimenzija L_1, L_2, ... L_N se definiše na sledeći način:

`T niz[L1] [L2] [LN];`

Niz se indeksira navođenjem nizovne promenljive i odgovarajućih indeksa:

`mdNiz[i1][i2]...[iN]`

Na ovaj način se statički alociraju višedimenzionalni nizovi u C/C++. Iako je moguće koristiti i dinamičku alokaciju, tom temom se nećemo baviti, ali preporučujemo čitaocima koje to interesuje da potraži u dokumentaciji ili na Internetu kako se to radi.

Na primeru sledećeg zadatka možemo videti kako se koriste multidimenzionalni nizovi.

Problem 10

Zadata je tabela sa prirodnim brojevima čije dimenzije ne prelaze 100 x 100. Potrebno je napisati program u programskom jeziku C/C++ koji ispituje da li je zbir elemenata po redovima jednak u svim redovima.

Zadatak se opet može rešiti tako što, nakon što se učitaju elementi tabele, saberemo sve elemente u prvom redu tabele. A zatim idemo red po red i sabiramo elemente po redovima i proveravamo i pamtimo da li je suma ista kao ona u prvom redu.

```
#include <iostream>
using namespace std;

int main()
{
      int n, m;
      int table[100][100];

      cin >> n >> m;
      for (int i=0; i < n; i++)
            for (int j=0; j < m; j++)
                  cin >> table[i][j];

      int sumInit;
```

```
	bool areEqual = true;

	sumInit = 0;
	for (int j=0; j < m; j++)
		sumInit += table[0][j];

	for (int i=1; i < n; i++)
	{
		int sumCurr = 0;
		for (int j=0; j < m; j++)
			sumCurr += table[i][j];

		areEqual = areEqual && (sumCurr == sumInit);
	}

	if (areEqual)
		cout << "Svi redovi imaju jednaku sumu" << endl;
	else
		cout << "Nisu svu jednaki" << endl;
	return 0;
}
```

Višedimenzionalni nizovi u C#

Isto kao kod jednodimenzionalnih nizova tako i kod višedimenzionalnih, C# podržava samo dinamičku alokaciju. Niz čiji su elementi tipa T, a sa N dimenzija se alocira na sledeći način:

`T mdNiz [ , , … , ] = new T[L1, L2, … LN];`

Niz se indeksira navođenjem nizovne promenljive i odgovarajućih indeksa:

`mdNiz[i1, i2, ... iN]`

Ovaj pristup alocira višedimenzionalni niz i ne treba ga mešati sa alokacijom „nazubljenog niza“, koji može da izgleda na primer ovako:

nazubljeniNiz	0	1	2	3	4
0					
1					
2					

Ovakav niz se može deklarisati na sledeći način:

```
T nazubljeniNiz [ ] [ ] = new T [ ] [ ]
{
  new T[2],
  new T[5],
  new T[3]
};
```

I u ovom slučaju ispravno je indeksirati element sa `nazubljeniNiz[1][3]`, ali ne i `nazubljeniNiz[0][3]`, budući da taj element ne postoji.

Na primeru sledećeg zadatka možemo videti kako se koriste multidimenzionalni nizovi.

Problem 11

Zadata je tabela sa prirodnim brojevima čije dimenzije ne prelaze 100 x 100. Potrebno je napisati program u programskom jeziku C# koji ispituje da li je zbir elemenata po redovima jednak u svim redovima.

Zadatak se opet može rešiti tako što nakon što se učitaju elementi tabele, saberemo sve elemente u prvom redu tabele. A zatim idemo red po red i sabiramo elemente po redovima i proveravamo i pamtimo da li je suma ista kao ona u prvom redu.

```
using System;

public class SumRows
{
      public static void Main()
      {
      int n, m;
      n = int.Parse(Console.ReadLine());
      m = int.Parse(Console.ReadLine());
      int tabela[ , ]= new int [n, m];

      for (int i=0; i < n; i++)
            for (int j=0; j < m; j++)
               table[i,j] = int.Parse(Console.ReadLine());

      int sumInit;
      bool areEqual = true;

      sumInit = 0;
      for (int j=0; j < m; j++)
            sumInit += table[0, j];

      for (int i=1; i < n; i++)
      {
            int sumCurr = 0;
            for (int j=0; j < m; j++)
```

```
                sumCurr += table[i, j];

            areEqual = areEqual && (sumCurr == sumInit);
        }

        if (areEqual)
            Console.WriteLine(
              "Svi redovi imaju jednaku sumu");
        else
            Console.WriteLine(
              "Nisu svu jednaki");
    }
}
```

Zadaci višedimenzionalni nizovi

U nastavku ćemo kroz zadatke videti na koje načine se višedimenzionalni nizovi mogu koristiti pri rešavanju različitih problema.

Za rešavanje zadataka će biti korišćen pseudo jezik koji je sličan Pascalu.

Problem 12

Data je dvodimenzionalna matrica koja sadrži cele brojeve. Potrebno je naći maksimume za svaki red i za svaku kolonu matrice, ako se zna da su dimenzije matrice manje od 1000 x 1000.

Za rešavanje ovog zadatka prolazićemo red po red u matrici, a zatim kolonu po kolonu i tretirati ih kao niz u kome ćemo tražiti maksimum.

```
10    array[1..1000][1..1000] of integer a
20    integer n, m, i, j, max
30    read (n, m)
40    for i= 1 to n do
50          for j= 1 to m do
60                read(a[i][j])

70    for i= 1 to n do
80          max = a[i][1]
90          for j= 2 to m do
100               if a[i][j] > max then max = a[i][j]
110         end for
120         write (“Maksimum reda ”, i, “je: ”, max)
130   end for

140   for j= 1 to m do
150         max = a[1][j]
160         for i= 2 to n do
```

```
170             if a[i][j] > max then max = a[i][j]
180         end for
190         write (“Maksimum kolone ”, j, “je: ”, max)
200   end for
```

Problem 13

Zadatu matricu A koja sadrži cele brojeve je potrebno ispisati spiralno počevši od elementa $A_{1,1}$. Dimenzije matrice su manje od 1000.

Ovaj problem rešavamo tako što idemo uvek u jednom smeru: desno, dole, levo ili gore kroz matricu i pamtimo granice unutar matrice do kojih idemo. Kada naiđemo na granicu menjamo smer „zaokrećući“ na desno i ažuriramo granice do kojih se ide u ostatku obilaženja. Smer u kome ćemo ići će nam određivati dve promenljive za pomak po x i po y osi koje će moći da imaju vrednosti -1, 0, ili 1 i njihovu vrednost ćemo dodavati na koordinate trenutne pozicije kako bismo dobili novu poziciju.

```
10    array [1..1000][1..1000] of integer a
20    integer n, m, i, j, x, y, xstep, ystep
30    integer xleft, xright, yup, ydown
40    read (n, m)
50    for i= 1 to n do
60          for j= 1 to m do
70                read(a[i][j])

80    x=1
90    y=1
100   xstep=1
110   ystep=0
120   xleft = 1
130   xright = m
140   yup = 1
150   ydown = n

160   for i= 1 to n*m do
170         write (a[y][x])

180         if x + xstep > xright then
190               xstep = 0; ystep = 1
200               y = y + 1
210               yup = yup + 1
220         else if y + ystep > ydown then
230               xstep = -1; ystep = 0
240               x = x - 1
250               xright = xright - 1
260         else if x + xstep < xleft then
270               xstep = 0; ystep = -1
```

```
280             y = y - 1;
290             ydown = ydown - 1
300       else if y + ystep < yup then
310             xstep = 1; ystep = 0
320             x = x + 1
330             xleft = xleft + 1
340       else
350             x = x + xstep
360             y = y + ystep
370       end if
380  end for
```

Problem 14

Potrebno je napisati program koji množi dve matrice. Dimenzije zadatih matrica ne prelaze 100.

Množenjem dve matrice dobija se nova matrica. Da bi se dve matrice M1 i M2 dimenzija n1 x m1 i n2 x m2 mogle pomnožiti, mora biti ispunjeno da je m1=n2, u suprotnom matrice se ne mogu pomnožiti. U slučajevima kada je navedeni uslov zadovoljen, tada se kao rezultat množenja dobija matrica R čije su dimenzije n1 x m2. Element R[i][j] rezultujuće matrice se dobija kao skalarni proizvod i-tog reda matrice M1 i j-te kolone M2, tj.

$$R[i][j] = \sum_{k=1}^{m1} M1[i][k] \cdot M2[k][j]$$

Prema tome, jedan element rezultujuće matrice ćemo računati u jednoj petlji kao skalarni proizvod, i to onda ponavljamo za svaki element rezultujuće matrice.

```
10   array [1..100][1..100] of float M1, M2, R
20   integer n1, m1, n2, m2, i, j, k
30   float sproduct

40   read (n1, m1)
50   for i= 1 to n1 do
60        for j= 1 to m1 do
70             read (M1[i][j])
80   read (n2, m2)
90   for i= 1 to n2 do
100       for j= 1 to m2 do
110            read (M2[i][j])

120  if m1 != n2 then
130    write (“Matrice se ne mogu pomnoziti, m1<>n2.”)
140  else
```

```
150     for i= 1 to n1 do
160       for j= 1 to m2 do
170         sproduct = 0
180         for k= 1 to m1 do
190               sproduct = sproduct + M1[i][k]*M2[k][j]
200         end for
210         R[i][j] = sproduct
220       end for
230     end for
240   end if
```