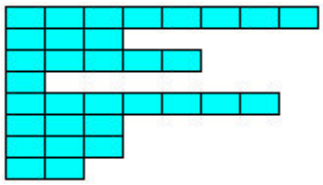
Strukture podataka i algoritmi 2
April, 2010

1. U realnim problemima je često pogodno koristiti matrice čiji redovi nisu jednakih dužina. Takve matrice se nazivaju *nazubljenim* ili *testerastim* matricama. Korišćenjem dinamičkih struktura omogućiti rad sa ovakvim matricama na sledeći način:
   - **a.** Formirati odgovarajuću strukturu podataka koja će predstavljati nazubljenu matricu celih brojeva. **(5 poena)**
   - **b.** Napisati funkciju *AddRow* koja dodaje novi red u matricu. **(5 poena)**
   - **c.** Napisati funkciju *AddElement* koja u zadati postojeći red dodaje novi element. **(5 poena)**
   - **d.** Napisati funkcije *SetElement* i *GetElement* koje dodeljuju, odnosno vraćaju vrednost elementa u *i*-tom redu i *j*-toj koloni matrice (pretpostaviti da element postoji). **(5 poena)**
   - **e.** Napisati funkciju *RemoveElement* koja iz matrice briše element u *i*-tom redu i *j*-toj koloni matrice (pretpostaviti da element postoji). **(5 poena)**
   - **f.** Napisati glavni program koji korišćenjem napisanih funkcija formira donju trougaonu matricu sa *n* redova i puni je redom brojevima od 1 do $n(n+1)/2$. **(5 poena)**

   Napomena: Nije dozvoljeno korišćenje nizova.



U nastavku je dato jedno od rešenje studenata.

Rešenje 1
(Dalibor Lakićević 49/08)

```
#include <stdio.h>
#include <stdlib.h>
/*
  Zbog dinamicke duzine moramo koristiti liste
  Pravimo strukturu za red i strukturu za celu matricu koja predstavlja listu redova.
*/
typedef struct ListaElemenata
  {
    int vrednost;
    struct ListaElemenata *sledeci;
  }Element;

typedef struct ListaRedova
  {
    Element *Koren;
    struct ListaRedova *sledeci;
  }Red;
/*
  Funkcija za dodavanje redova koristi algoritam za dodavanje novog elementa na kraj
liste
*/
void AddRow( Red **Matrica )
  {
    Red *novi,*tmp;
```

```
    novi = ( Red* ) malloc( sizeof( Red ) );
    novi->Koren = NULL;
    novi->sledeci = NULL;
    if( !( *Matrica ) )
      {
         *Matrica = novi;
         return;
      }
    else
      {
         tmp = *Matrica;
         while( tmp->sledeci )
           tmp = tmp->sledeci;
         tmp->sledeci = novi;
      }
  }
/*
  Nalazimo red koji koji se trazi
  Onda primenjujemo algoritam za dodavanje novog elementa na kraj liste
*/
void AddElement( Red *Matrica, int i )
  {
    int brojac = 0;
    Element *novi, *tmp;
    novi = ( Element* ) malloc( sizeof( Element ) );
    novi->sledeci = NULL;
    while ( i > brojac++ )
        Matrica = Matrica->sledeci;
    if( Matrica->Koren )
      {
        tmp = Matrica->Koren;
        while( tmp->sledeci )
          tmp = tmp->sledeci;
        tmp->sledeci = novi;
      }
     else
      Matrica->Koren = novi;
  }
/*
   Pomocna funkcija za nalazenje odredjenog elementa u matrici
*/
Element *NadjiElement( Red *Matrica, int i, int j )
  {
    int brojac;
    Element *tmp = NULL;
    brojac = 0;
    while( i > brojac++ )
        Matrica = Matrica->sledeci;
    tmp = Matrica->Koren;
    brojac = 0;
    while( j > brojac++ )
      tmp=tmp->sledeci;
    return tmp;
  }
/*
  Koristimo prethodnu funkciju( NadjiElement ) da nadjemo element
  Onda setujemo njegovu vrednost
*/
void SetElement( Red *Matrica, int i, int j, int setvrednost )
  {
    Element *tmp;
    tmp = NadjiElement( Matrica, i, j );
```

```
    tmp->vrednost = setvrednost;
  }
/*
  Koristimo prethodnu funkciju( NadjiElement ) da nadjemo element
  Onda vratimo njegovu vrednost
*/
int GetElement( Red *Matrica, int i, int j )
  {
    Element *tmp;
    tmp = NadjiElement( Matrica, i , j );
    return tmp->vrednost;
  }
/*
   Prolazimo kroz sve redove
   Onda koristimo algoritam za brisanje elementa iz liste na korenu onog reda koji se
trazi
*/
void RemoveElement( Red *Matrica, int i, int j )
  {
    int brojac;
    Element *prethodni, *trenutni;;
    brojac = 0;
    while( i > brojac++ )
      Matrica = Matrica->sledeci;
    prethodni = NULL;
    trenutni = Matrica->Koren;
    brojac = 0;
    while( j > brojac++ )
      {
        prethodni = trenutni;
        trenutni = trenutni->sledeci;
      }
    if( prethodni == NULL )
      {
        prethodni = trenutni;
        Matrica->Koren = trenutni->sledeci;
        free( prethodni );
        return;
      }
    else
      {
        prethodni->sledeci = trenutni->sledeci;
        free( trenutni );
        return;
      }
  }
main()
 {
   Red *Matrica = NULL;
   int i, j, n, brojac = 0;
   printf("Unesite broj n : ");
   scanf("%d",&n);
   for( i = 1; i <= n ; i++ )
     {
       AddRow( &Matrica );
       for( j = 0; j < i ; j++ )
         {
            AddElement( Matrica, i - 1 );
            SetElement( Matrica, i - 1, j, ++brojac );
         }
     }
 }
```