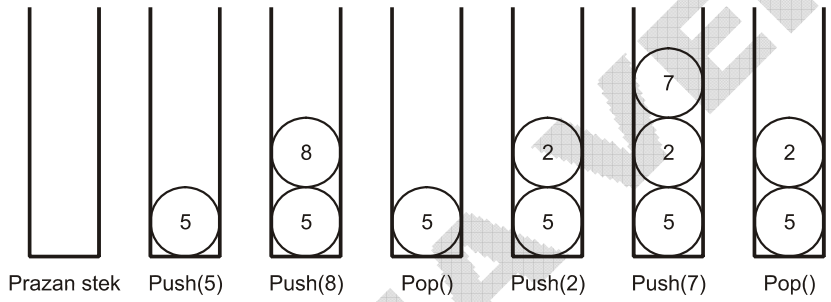
Stekovi i redovi

Veliki broj realnih problema zahteva korišćenje struktura podataka kao što su **stekovi** i **redovi**. Tipičan primer korišćenja stekova je stek koji programski jezik koristi da bi implementirao pozivanje funkcija i povratak iz njih. Sa druge strane, jedna od najčešćih primena redova je red za upravljanje redosledom procesa koje je potrebno izvršiti. Obe ove strukture podataka su u formi liste, tako da je za njihovu implementaciju moguće iskoristiti nizove ili povezane liste.

Stekovi

Stekovi (engleski *stack*) su liste elemenata u kojima je moguće dodavanje i oduzimanje elemenata samo na jednom kraju, koji se naziva **vrh steka**. To praktično znači da se elementi sa steka uklanjaju obrnutim redosledom u odnosu na redosled kojim su dodavani na stek. Iz tog razloga se ova struktura podataka često naziva i LIFO, što je skraćenica od engleskih reči **L**ast **I**n **F**irst **O**ut (Poslednji unutra prvi napolje). Operacija dodavanja elemanta na stek se najčešće naziva **Push** (gurnuti), dok se operacija skidanja elementa sa steka naziva **Pop** (skinuti).

Stek možemo zamisliti kao cev zatvorenu na jednom kraju u koju se na drugom kraju dodaju kuglice koje predstavljaju elemente (Slika ###). Prilikom vađenja kuglica iz cevi uvek moramo prvo izvaditi onu kuglicu koja je poslednja ubačena u cev i tako redom. Kuglica koja je prva ubačena u cev biće poslednja izvađena. Zahvaljujući ovoj osobini, stekovi se često u srpskom jeziku nazivaju i **stogovi**.



Slika ### Šematski prikaz dodavanja i skidanja elemenata sa steka

S obzirom da su stekovi u osnovi liste, oni se mogu implementirati korišćenjem nizova ili povezanih lista.

Implementacija steka pomoću niza

Kada se za implementaciju steka koristi niz, operacije dodavanja i skidanja elemenata sa steka se realizuju korišćenjem osnovnih operacija nad nizom. Ograničenje implementacije pomoću niza je nemogućnost proširenja i skraćenja niza u zavisnosti od broja elemenata na steku.

Za implementaciju steka koristi se niz konstante veličine, koja mora biti dovoljna da se u nju smesti maksimalan broj elemenata koji se u jednom trenutku može naći na steku. Pored toga, u svakom trenutku je neophodno znati broj elemenata na steku ili indeks elementa na vrhu steka. Indeks -1 označava da je stek prazan. Prilikom dodavanja elementa na stek, povećava se indeks vrha steka i na tu poziciju u nizu upisuje novi element. Suprotno, prilikom skidanja elementa sa steka umanjuje se indeks vrha steka.

Primer

```
#include <stdio.h>
#define MAX 100 /* Maksimalna velicina steka */
#include <stdlib.h>

void push(int stack[], int *top, int value)
{
   if( *top < MAX )
   {
      *top = *top + 1;
```

```
      stack[*top] = value;
   }
   else
   {
      printf("Stek je pun i ne moze da primi novu vrednost.\n");
      exit(0);
   }
}

void pop(int stack[], int *top, int *value)
{
   if( *top >= 0 )
   {
      *value = stack[*top];
      *top = *top - 1;
   }
   else
   {
      printf("Stek je prazan i ne moze se skinuti vrednost sa njega.\n");
      exit(0);
   }
}

void main()
{
   int stack[MAX];
   int top = -1;
   int n,value;

   do
   {
      do
      {
         printf("Unesite element koji zelite da dodate na stek:\n");
         scanf("%d",&value);

         push(stack,&top,value);

         printf("Unesite 1 za dodavanje novog elementa na stek:\n");
         scanf("%d",&n);
      } while(n == 1);

      printf("Unesite 1 za skidanje elementa sa steka:\n");
      scanf("%d",&n);

      while( n == 1)
      {
         pop(stack,&top,&value);
         printf("Skinuta vrednost je %d\n",value);

         printf("Unesite 1 za skidanje elementa sa steka:\n");
         scanf("%d",&n);
      }

      printf("Unesite 1 za dodavanje novog elementa na stek:\n");
      scanf("%d",&n);

   } while(n == 1);
}
```


Prethodni primer sadrzi funkcije *push* i *pop* za dodavanje i skidanje elementa sa steka. U glavnom programu omogućeno je dodavanje i skidanje elemenata sa steka proizvoljan broj puta odabirom odgovarajućih komandi.

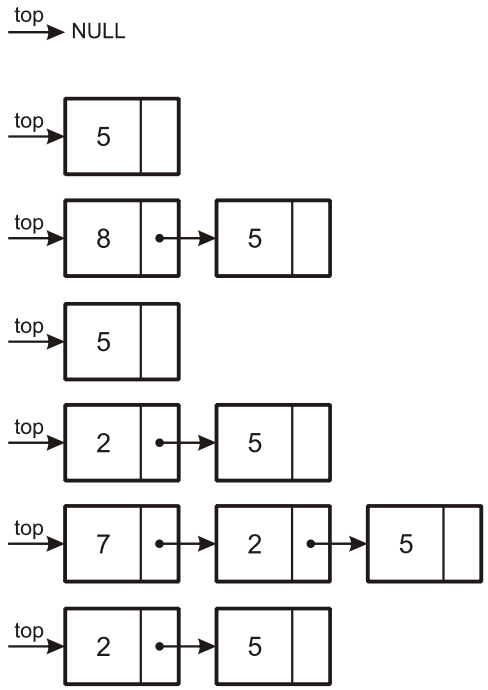
Implementacija steka pomoću liste

Stek se može veoma efikasno implementirati korišćenjem povezanih lista tako što bi se novi element dodavao uvek na početak liste. Takođe, u slučaju skidanja elementa sa steka skidao bi se uvek prvi element u listi, odnosno onaj koji je poslednji dodat.

Na početku, lista je prazna, pa je i pokazivač na početak liste (*top*) jednak NULL. Funkcija *push* uzima pokazivač na postojeću listu kao prvi parametar i vrednost koju treba dodati kao drugi parametar, kreira

novi element i dodaje ga na početak liste. Funkcija *pop* uzima pokazivač na početak liste kao prvi parametar i pokazivač na promenljivu u koju će smestiti vrednost skinutog elementa kao drugi parametar. Nakon toga, funkcija vraća vrednost prvog elementa u listi i pomera pokazivač *top* na sledeći element u listi. Na kraju se uništava element koji je bio na početku liste.

Na Slici ### prikazan je izgled steka predstavljenog povezanom listom prilikom dodavanja i skidanja elemenata sledećim redosledom: Push(5), Push(8), Pop(), Push(2), Push(7), Pop().



Slika ### Šematski prikaz dodavanja i skidanja elemenata sa steka predstavljenog povezanom listom

Primer

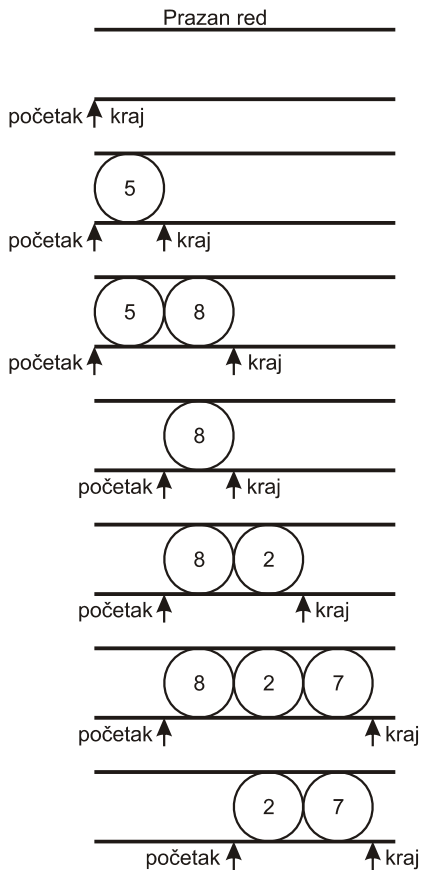
```
# include <stdio.h>
# include <stdlib.h>

struct node
{
   int data;
   struct node *link;
};

struct node* push(struct node *p, int value)
{
   struct node *temp;

   /* kreiranje novog cvora koriscenjem prosledjene vrednosti */
   temp=(struct node *)malloc(sizeof(struct node));
   if(temp==NULL)
   {
      printf("Greska pri alociranju memorije.\n");
      exit(0);
   }

   temp->data = value;
   temp->link = p;
   p = temp;

   return(p);
}

struct node* pop(struct node *p, int *value)
{
```

```
    struct node *temp;

    if(p==NULL)
    {
        printf("Greska. Stek je prazan.\n");
        exit(0);
    }

    *value = p->data;
    temp = p;
    p = p->link;

    free(temp);

    return(p);
}

void main()
{
    struct node *top = NULL;
    int n,value;

    do
    {
        do
        {
            printf("Unesite element koji zelite da dodate na stek:\n");
            scanf("%d",&value);

            top = push(top,value);

            printf("Unesite 1 za dodavanje novog elementa na stek:\n");
            scanf("%d",&n);
        } while(n == 1);

        printf("Unesite 1 za skidanje elementa sa steka:\n");
        scanf("%d",&n);

        while( n == 1)
        {
            top = pop(top,&value);
            printf("Skinuta vrednost je %d\n",value);

            printf("Unesite 1 za skidanje elementa sa steka:\n");
            scanf("%d",&n);
        }

        printf("Unesite 1 za dodavanje novog elementa na stek:\n");
        scanf("%d",&n);

    } while(n == 1);
}
```


Redovi

Redovi (engleski *queue*) su liste elemenata u kojima se elementi dodaju na jednom kraju liste, koji se naziva **kraj reda**, a oduzimaju se na drugom kraju liste, koji se naziva **početak reda**. To praktično znači da se elementi sa reda uklanjaju istim redosledom kao što su i dodavani na red. Iz tog razloga se ova struktura podataka često naziva i FIFO, što je skraćenica od engleskih reči **F**irst **I**n **F**irst **O**ut (Prvi unutra prvi napolje). Operacija dodavanja elemanta u red se najčešće naziva **Insert** (ubaciti), dok se operacija brisanja elementa iz reda naziva **Delete** (obrisati).

Red možemo zamisliti kao cev otvorenu na oba kraja u koju se na jednom kraju ubacuju kuglice koje predstavljaju elemente (Slika ###). Prilikom vađenja kuglica iz cevi uvek se prvo vadi kuglica koja je prva ubačena u cev i tako redom. Kuglica koja je poslednja ubačena u cev biće poslednja izvađena. Upravo ovoj osobini redovi duguju svoj naziv, jer podsećaju na čekanje u redu gde se prvo uslužuju oni koji su prvi stigli.



Slika ### Šematski prikaz ubacivanja i brisanja elemenata iz reda

S obzirom da su stekovi u osnovi liste, oni se mogu implementirati korišćenjem nizova ili povezanih lista.

Implementacija reda pomoću niza

Kada se za implementaciju reda koristi niz, operacije dodavanja i brisanja elemenata iz reda se realizuju korišćenjem osnovnih operacija nad nizom. Ograničenje implementacije pomoću niza je nemogućnost proširenja i skraćenja niza u zavisnosti od broja elemenata u redu.

Za implementaciju reda koristi se niz konstante veličine, koja mora biti dovoljna da se u nju smesti ukupan broj elemenata koji se dodaju u red, bez obzira na to koliko je njih obrisano iz reda. Pored toga, u svakom trenutku je neophodno znati indekse prvog i poslednjeg elementa u redu. Indeksi -1 označavaju da je red prazan. Prilikom dodavanja elementa u red, povećava se indeks poslednjeg elementa i na tu poziciju u redu upisuje se novi element. U slučaju brisanja elementa iz reda, povećava se indeks prvog elementa u redu.

Moguća je i drugačija implementacija koja bi zahtevala samo onoliku veličinu niza koliko je potrebno da se smeste elementi koji su u jednom trenutku u redu, međutim takva realizacija zahteva stalno pomeranje preostalih elemenata prilikom brisanja prvog elementa iz reda.

Primer

```
#include <stdio.h>
#define MAX 100 /* Maksimalna velicina reda */
#include <stdlib.h>

void insert(int queue[], int *rear, int value)
{
   if(*rear < MAX-1)
   {
      *rear= *rear +1;
      queue[*rear] = value;
   }
   else
```

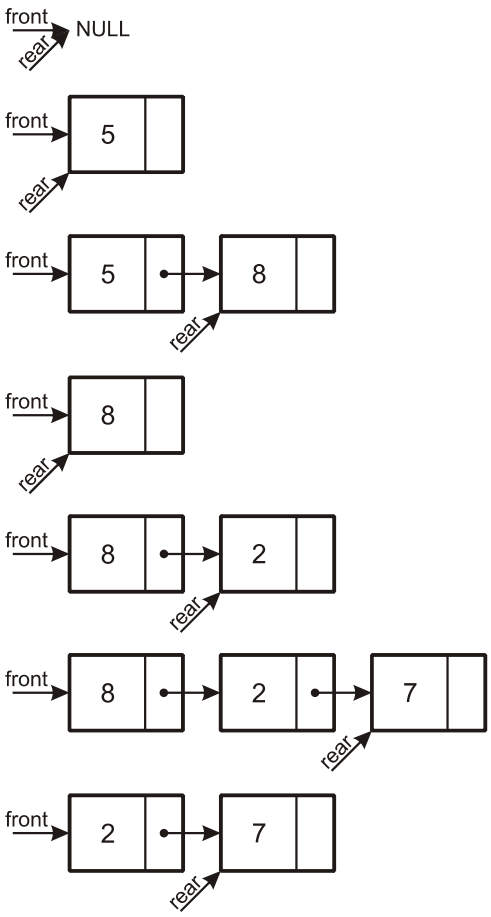
```
    {
        printf("Red je pun. Ne moze se dodati vrednost.\n");
        exit(0);
    }
}

void delete(int queue[], int *front, int rear, int * value)
{
    if(*front == rear)
    {
        printf("Red je prazan. Ne moze se obrisati vrednost.\n");
        exit(0);
    }

    *front = *front + 1;
    *value = queue[*front];
}

void main()
{
    int queue[MAX];
    int front,rear;
    int n,value;
    front=rear=(-1);

    do
    {
        do
        {
            printf("Unesite element koji zelite da dodate u red:\n");
            scanf("%d",&value);

            insert(queue,&rear,value);

            printf("Unesite 1 za dodavanje novog elementa u red:\n");
            scanf("%d",&n);
        } while(n == 1);

        printf("Unesite 1 za brisanje elementa iz reda:\n");
        scanf("%d",&n);

        while( n == 1)
        {
            delete(queue,&front,rear,&value);
            printf("Obrisana vrednost je %d\n",value);

            printf("Unesite 1 za brisanje elementa iz reda:\n");
            scanf("%d",&n);
        }

        printf("Unesite 1 za dodavanje novog elementa u red:\n");
        scanf("%d",&n);

    } while(n == 1);
}
```


Implementacija reda pomoću liste

Red se može veoma efikasno implementirati korišćenjem povezanih lista tako što bi se novi element dodavao uvek na kraj liste. Takođe, u slučaju brisanja elementa iz reda skidao bi se uvek prvi element u listi, odnosno onaj koji je prvi dodat.

Na početku, lista je prazna, pa su i pokazivači na početak i kraj liste (*front* i *rear*) jednaki NULL. Funkcija *insert* kreira novi element i dodaje ga na kraj liste. Funkcija *delete* vraća vrednost prvog elementa u listi i pomera pokazivač *front* na sledeći element u listi. Na kraju se uništava element koji je bio na početku liste.

Na Slici ### prikazan je izgled reda predstavljenog povezanom listom prilikom dodavanja i brisanja elemenata sledećim redosledom: Insert(5), Insert(8), Delete(), Insert(2), Insert(7), Delete().



Slika ### Šematski prikaz dodavanja i brisanja elemenata iz reda predstavljenog povezanom listom

Primer

```
# include <stdio.h>
# include <stdlib.h>

struct node
{
   int data;
   struct node *link;
};

void insert(struct node **front, struct node **rear, int value)
{
   struct node *temp;

   /* kreiranje novog cvora koriscenjem prosledjene vrednosti */
   temp=(struct node *)malloc(sizeof(struct node));
   if(temp==NULL)
   {
      printf("Greska pri alociranju memorije.\n");
      exit(0);
   }

   temp->data = value;
   temp->link=NULL;

   if(*rear == NULL)
   {
      *rear = temp;
      *front = *rear;
   }
   else
   {
      (*rear)->link = temp;
```

```
      *rear = temp;
   }
}

void delete(struct node **front, struct node **rear, int *value)
{
   struct node *temp;

   if((*front == *rear) && (*rear == NULL))
   {
      printf("Red je prazan. Ne moze se obrisati vrednost.\n");
      exit(0);
   }

   *value = (*front)->data;
   temp = *front;
   *front = (*front)->link;

   if(*rear == temp)
      *rear = (*rear)->link;

   free(temp);
}

void main()
{
   struct node *front=NULL,*rear = NULL;
   int n,value;

   do
   {
      do
      {
         printf("Unesite element koji zelite da dodate u red:\n");
         scanf("%d",&value);

         insert(&front,&rear,value);

         printf("Unesite 1 za dodavanje novog elementa u red:\n");
         scanf("%d",&n);
      } while(n == 1);

      printf("Unesite 1 za brisanje elementa iz reda:\n");
      scanf("%d",&n);

      while( n == 1)
      {
         delete(&front,&rear,&value);
         printf("Obrisana vrednost je %d\n",value);

         printf("Unesite 1 za brisanje elementa iz reda:\n");
         scanf("%d",&n);
      }

      printf("Unesite 1 za dodavanje novog elementa u red:\n");
      scanf("%d",&n);

   } while(n == 1);
}
```