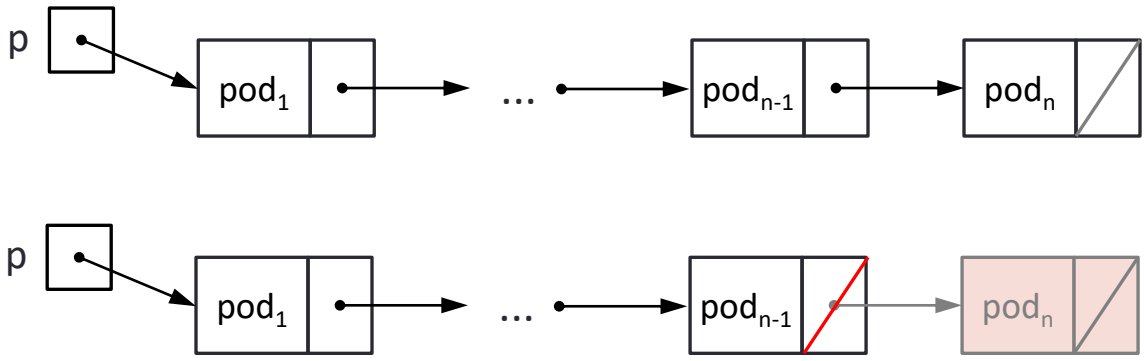
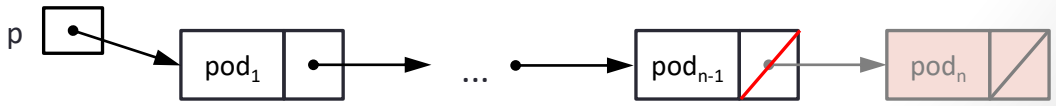
Brisanje liste

- Brisanjem glave liste gubi se veza ka ostatku liste, a svi elemnti liste i dalje zauzimaju memorijski prostor, a nije moguće pristupiti im
- Pri brisanju liste neophodno je obrisati najpre poslednji element, pa element pre njega i tako unazad do prvog elementa liste



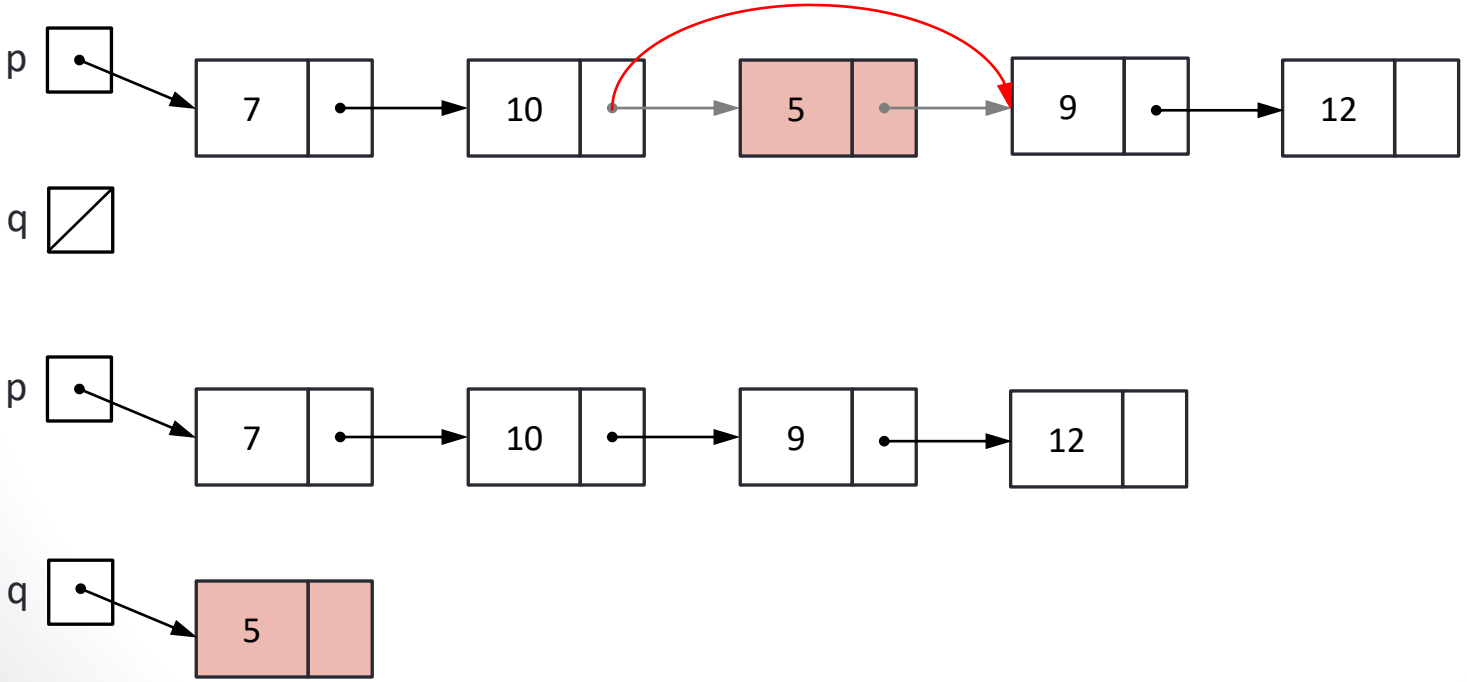
Brisanje liste

```
void obrisi_l(struct element *p)
{
    if(p == NULL)
        return;
    if(p->sledeci != NULL)
        obrisi_l(p->sledeci);
    free(p);
}
```



Sortiranje liste

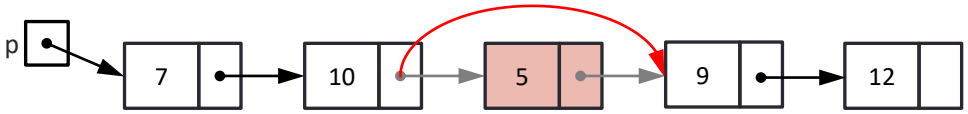
- Od date liste kreiramo novu soritranu listu tako što
  - Pronalazimo najmanji element date liste
  - Uklanjamo ga iz liste
  - Stavljamo na kraj soritirane liste koja je inicijalno bila prazna



Sortiranje liste

```
/* funkcija za sortiranje liste u rastucem redosledu */
struct element *sortiraj(struct element *p)
{
    struct element *pom1,*pom2,*min,*prethodni,*q;

    q = NULL;
    while(p != NULL)
    {
        prethodni = NULL;
        min = pom1 = p;
        pom2 = p->sledeci;

        /* pronalazenje minimuma */
        while ( pom2 != NULL )
        {
            if( pom2->podatak < min->podatak )
            {
                min = pom2;
                prethodni = pom1;
            }
            pom1 = pom2;
            pom2 = pom2->sledeci;
        }
        ...
```

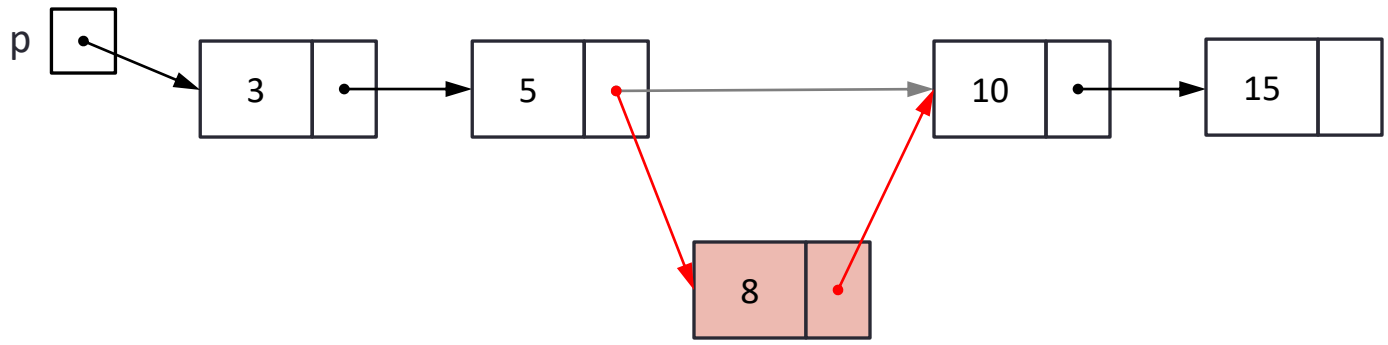
Sortiranje liste




```
        ...

        /* uklanjanje minimuma iz liste */
        if (prethodni == NULL) p = min->sledeci;
        else
            prethodni->sledeci = min->sledeci;
        min->sledeci = NULL;

        /* dodavanje pronadjenog elementa na kraj sortirane liste */
        if( q == NULL)
            q = min;
        else
        {
            pom1 = q;
            while( pom1->sledeci != NULL)
                pom1 = pom1->sledeci;
            pom1->sledeci = min;
        }
    }
    return (q);
}
```

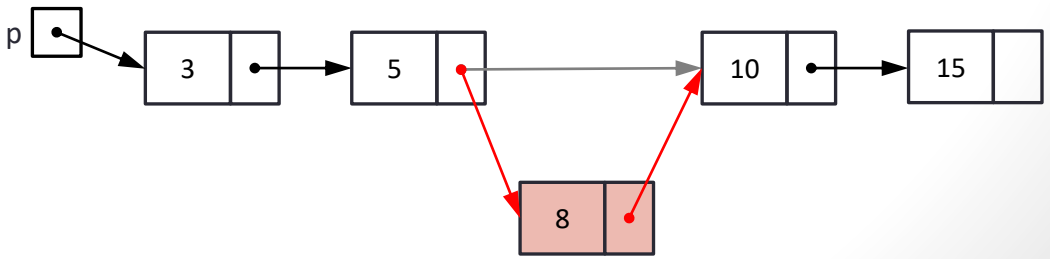
Umetanje elementa u sortiranu listu

- Da bismo u već sortiranu listu dodali novi element potrebo je pronaći element koji prethodi elementu koji je prvi veći od elementa koji se dodaje



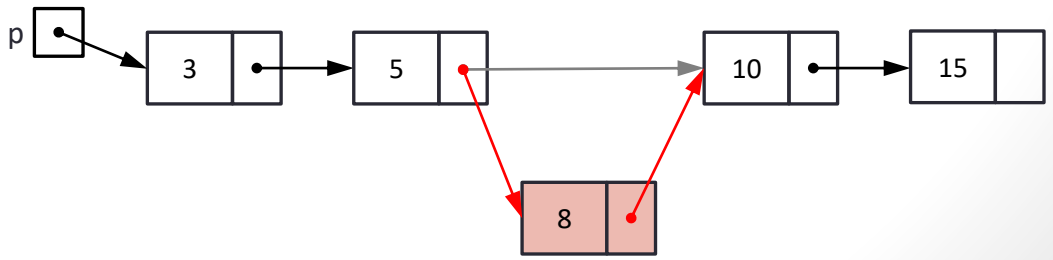
Umetanje elementa u sortiranu listu

```
/* funkcija za umetanje novog elementa u sortiranu listu*/
struct element *sort_dodaj(struct element *p, int n)
{
    struct element *trenutni, *prethodni, *novi;

    trenutni = p;
    prethodni = NULL;
    while( trenutni != NULL && trenutni->podatak < n )
    {
        prethodni = trenutni;
        trenutni = trenutni->sledeci;
    }

    novi = (struct element *) malloc(sizeof(struct element));
    if( novi == NULL)
    {
        printf("Greska pri alociranju memorije.\n");
        exit(0);
    }
    ...
```



Umetanje elementa u sortiranu listu

```
    ...

    if ( prethodni == NULL )
    {
        novi->podatak = n;
        novi->sledeci = p;
        p = novi;
    }
    else
    {
        novi->podatak = n;
        novi->sledeci = prethodni->sledeci;
        prethodni->sledeci = novi;
    }
    return(p);
}
```



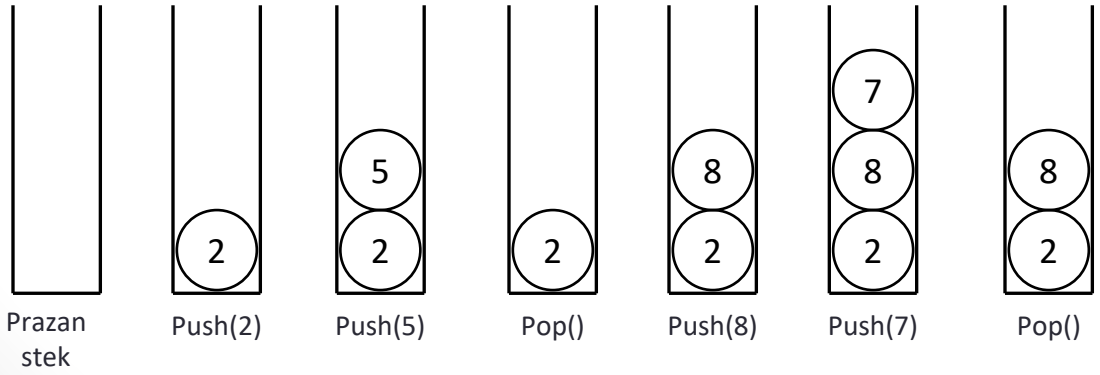
STEKOVI I REDOVI

Stekovi i redovi

- Veliki broj problema se rešava korišćenjem struktura kao što su stekovi i redovi
  - Stek koji koristi programski jezik da bi implementirao pozivanje funkcija i vraćanje iz njih
  - Red za upravljanje redosledom procesa koje treba izvršiti
- Obe strukture su linarne i mogu se realizovati korišćenjem nizova ili povezanih listi

Stek (*stack*)

- Stekovi su liste elemenata u kojima je moguće dodavanje i oduzimanje elemenata samo na jednom kraju, koji se naziva vrh steka.
- Elementi sa steka uklanjaju obrnutim redosledom u odnosu na redosled kojim su dodavani na stek.
- Ova struktura podataka često naziva i LIFO (*Last In First Out*)
- Operacija
  - dodavanja elemanta na stek se najčešće naziva Push (gurnuti),
  - skidanja elementa sa steka naziva Pop (skinuti).



Implementacija steka pomoću niza

- 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.
- 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.
  - Prilikom skidanja elementa sa steka umanjuje se indeks vrha steka.

Implementacija steka pomoću niza

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

#define MAX 100 /* Maksimalna velicina steka */

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);
    }
}
```

Implementacija steka pomoću niza

```
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);
    }
}
```

Implementacija steka pomoću niza

```
int 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);

        ...
```

Implementacija steka pomoću niza

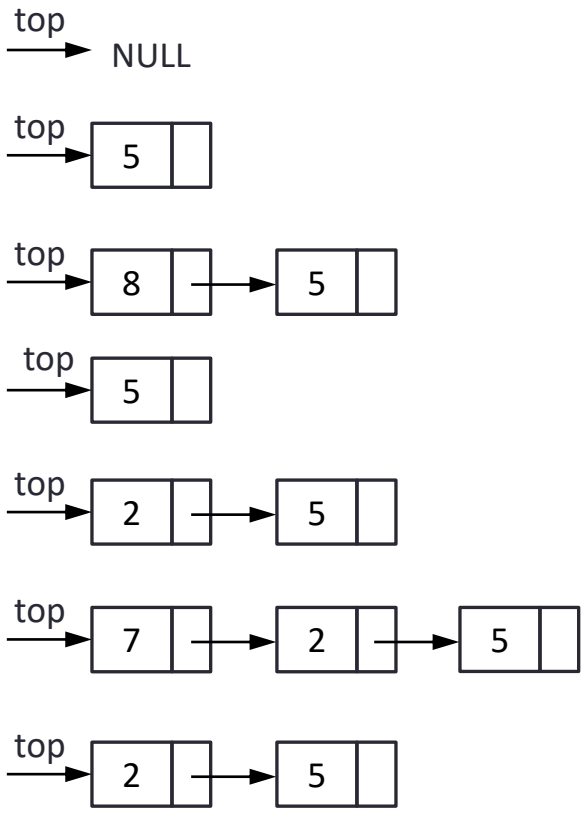
```
        ...
        printf("Unesite 1 za skidanje elementa sa steka:\n");
        scanf("%d",&n);
        while( n == 1)
        {
            pop(stack,&top,&value);
            printf("Skinuta vrednost je %d\n",value);
            printf("Unesite 1 za skidanje elementa sa steka:\n");
            scanf("%d",&n);
        }
        printf("Unesite 1 za dodavanje novog elementa na stek:\n");
        scanf("%d",&n);
    } while(n == 1);
}
```

Implementacija steka pomoću liste

- Stek se može veoma efikasno implementirati korišćenjem povezanih lista tako što bi se novi element dodavao uvek na početak liste.
- 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.
    - 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.

Implementacija steka pomoću liste

- Menjanje izgleda steka nakon niza operacija:



Implementacija steka pomoću liste

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

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

Implementacija steka pomoću liste

```
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);
}
```

Implementacija steka pomoću liste

```
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);
}
```

Implementacija steka pomoću liste

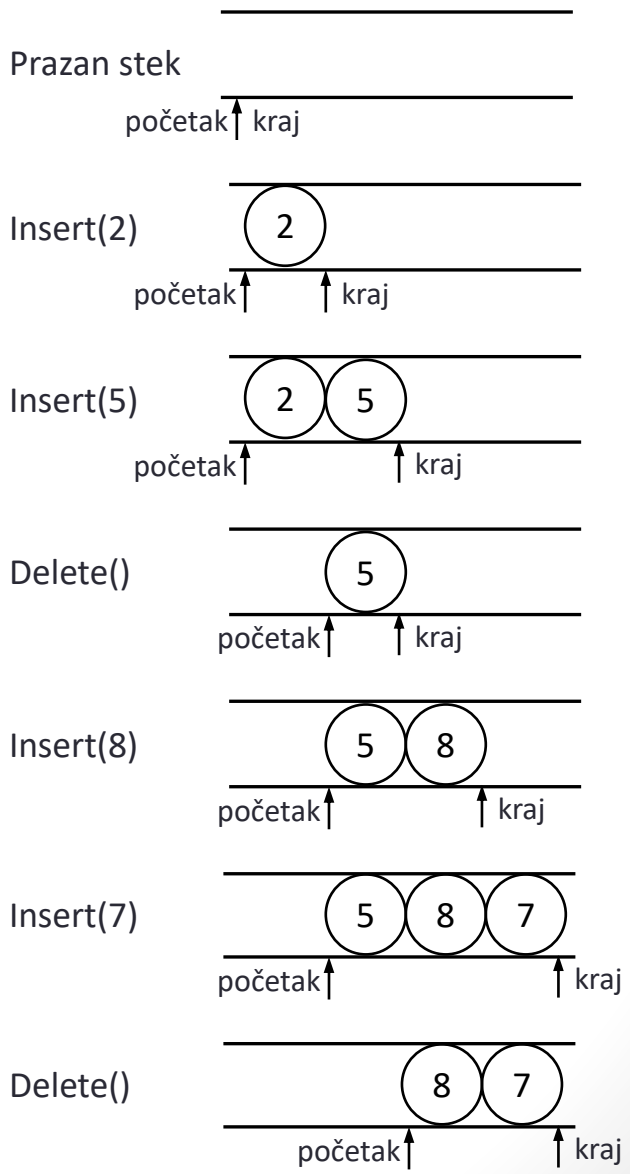
```
void main()
{
    struct node *top = NULL;
    int n,value;

    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);

        ...
```

Implementacija steka pomoću liste

```
        ...
        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);
}
```

Red (*queue*)

- Lista elemenata u kojoj se elementi
  - dodaju na jednom kraju liste - kraj reda
  - oduzimaju se na drugom kraju liste - početak reda.
- Elementi sa reda se uklanjaju istim redosledom kao što su i dodavani na red.
- Ova struktura podataka često naziva i FIFO (*First In First Out*)
- Operacija
  - dodavanja elemanta u red se najčešće naziva Insert (ubaciti)
  - brisanja elementa iz reda naziva Delete (obrisati).



Implementacija reda pomoću niza

- 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.
- 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.

Implementacija reda pomoću niza

- 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
  - Takva realizacija zahteva stalno pomeranje preostalih elemenata prilikom brisanja prvog elementa iz reda.

Implementacija reda pomoću niza

```
#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);
    }
}
```

Implementacija reda pomoću niza

```
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];
}
```

Implementacija reda pomoću niza

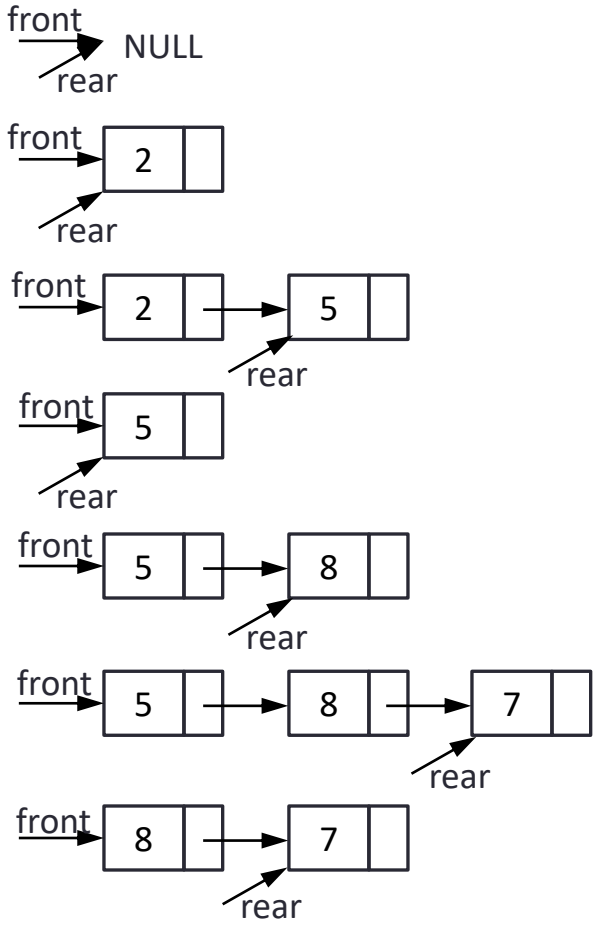
```
void main()
{
     int queue[MAX];
     int front,rear;
     int n,value;

     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.
- 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.

Implementacija reda pomoću liste

- Menjanje izgleda reda nakon niza operacija:



Implementacija reda pomoću liste

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

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

Implementacija reda pomoću liste

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

    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;
    }
}
```

Implementacija reda pomoću liste

```
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);
}
```

Implementacija reda pomoću liste

```
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);
}
```