Nizovi

Niz je uređena kolekcija objekata, gdje je svakom objektu pridružen nenegativan cio broj – indeks elementa niza
Primjeri:

1. niz parnih prirodnih brojeva manjih od 15: 2,4,6,8,10,12,14
2. niz prostih brojeva manjih od 20: 2,3,5,7,11,13,17,19
3. niz cifara u decimalnom razvoju razlomka 1/7: 0,1,4,2,8,5,7,1,4,2,8,5,7,1,4…

Ako niz nazovemo x, tada se element čiji je indeks i označavamo sa xi. Npr. u gornjem primjeru je $x_1=2$ a $x_6=12$. Niz ima 7 elemenata.

Indeks	1	2	3	4	5	6	7
Element	2	4	6	8	10	12	14

Ponekad je moguće napisati vezu između indeksa niza i elementa. Npr. u gornjem primjeru je to moguće: $x_i=2^i$, i=1,2,3,4,5,6,7

Primjer: Neka je $x_i=2^{i+1}-3$, i=1,2,3,4,5.

Indeks	1	2	3	4	5
Elementi	1	5	13	29	61

$x_i=2^{i+1}-3$ je opšti član niza, i=1,2,3,4,5.
Uobičajeno je da indeksi niza počinju od 1. Međutim, mogu počinjati i od nula

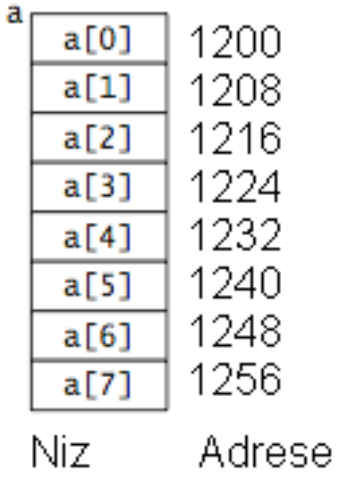
- $x_n=1+n^2$, n=0,1,2... predstavlja beskonačni niz 1, 2, 5, 10...
- Odredite x_{20}

Nizovi u programiranju

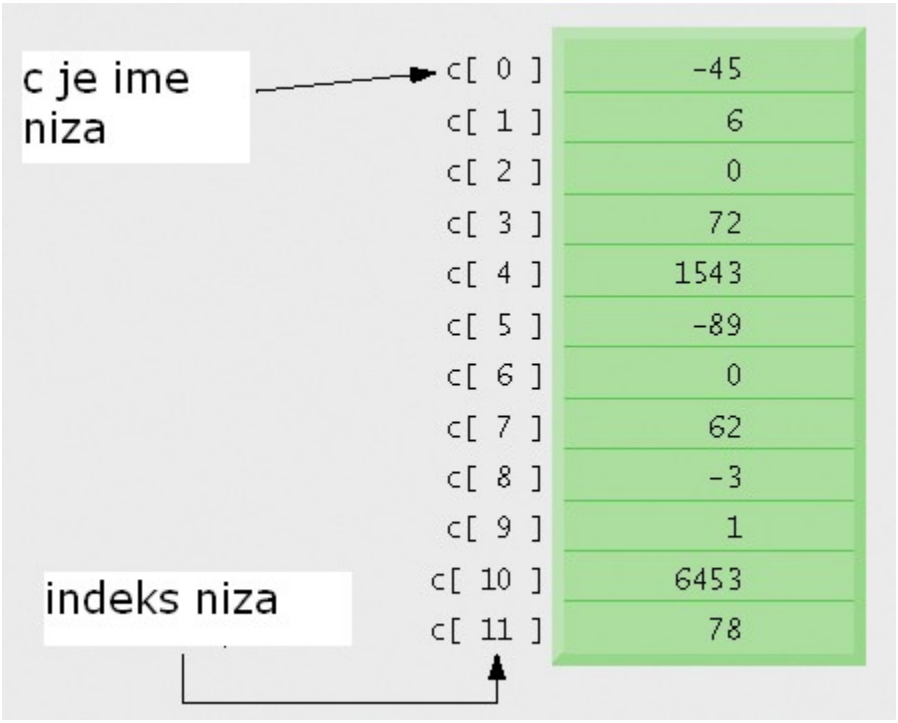
Nizovi sadrže samo elemente jednog tipa (npr. cijele brojeve, realne brojeve, karaktere, nizove karaktera, stringove, automobile, predmete, ocjene...). Elementi niza obično se označavaju srednjim (uglastim) zagradama. Npr. petnaesti element niza x (tj. x_{15}) označava se sa x[15]. Uzastopni elementi niza zauzimaju uzastopne memorijske lokacije.
Primjer: Niz cijelih brojeva u pascal-u, jedan cio broj zauzima 4 bajta

Indeks	1	2	3	4	5
Element	-22	12	123	0	-13
Adresa	3000	3004	3008	3012	3016

Primjer: niz realnih brojeva, jedan realan broj zauzima 8 bajtova

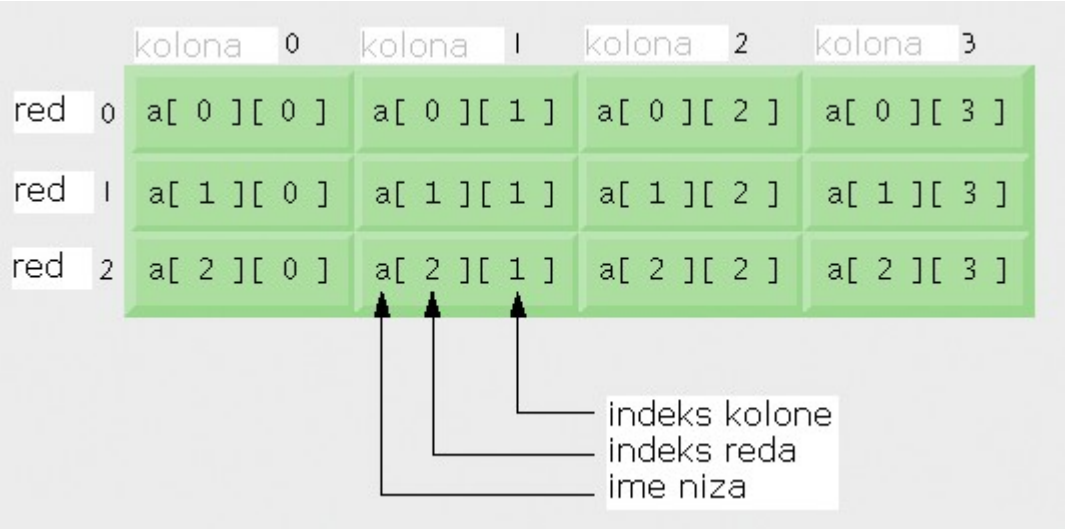


Primjer: niz u jezicima C, C++, Java. Obratite pažnju da indeksi počinju od 0.



Niz u jeziku Java je tzv. kontejnerski objekat koji može da čuva konačan fiksiran broj vrijednosti jednog tipa. Dužina niza (tj . broj elemenata niza) se određuje pri njegovom kreiranju i poslije toga je fiksirana. Elementi niza mogu biti proizvoljnog tipa.

Dvodimenzionalni nizovi (matrice)



Deklarisanje niza

- Java
  - `int c[] = new int[ 12 ];`
  - `String b[] = new String[ 100 ];`
- C, C++
  - `int c[12];`
- Pascal
  - `c:array[1..12] of integer;`

Deklarisanje matrice

- Java
  - `int c[][] = new int[3][4];`
- C, C++
  - `int c[3][4];`
- Pascal
  - `c:array[1..3,1..4] of integer;`
  - `d:array[-1..3,-2..4] of integer;`

Načini deklarisanja i alokacija memorije (java)

- Deklaracija i alokacija memorije odvojeno
```
int[] students;
students = new int[100];
```
- Jedna naredba
```
int[] students = new int[100];
```
- Upotreba postojećeg niza
```
int[] gradStudents = new int[100];
int[] students = gradStudents ;
```

Kada je memorija alocirana, podrazumijevane vrijednosti za elemente niza su **0** za brojeve, **false** za tip boolean i **null** za reference.

Primjer:

```
int[] students = new int[5];
System.out.println(students[3]);
```


```
String[] students = new String[2];
System.out.println(students[1]);
```

Dužina niza (java)
Koristi se **length** za određivanje dužine niza

Primjer:

```
String[] staff = {“Red”, “Sean”, “Patrick”, “Orr”};
int[] numbers = {4,2,1};

System.out.println(staff.length): // stampa se 4
System.out.println(numbers.length): // stampa se 3
```

Pristup elementima niza

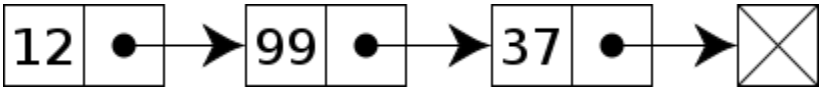
Elementu niza pristupa se navođenjem imena niza i odgovarajućeg indeksa
Primjer:

```
int[] a = {3, 5, 7, 9};

System.out.println(a[2]): // stampa se 7
System.out.println(a[a.length - 1]): // stampa se 9
System.out.println(a[15]): // java.lang.ArrayIndexOutOfBoundsException
```

Ulančane liste (linked lists)

Ulančana lista (**linked list)** je struktura podataka koja sadrži elemente ili čvorove (engl. nodes) a u svakom čvoru sadrži same podatke i referencu (vezu ili link ili pokazivač) na sljedeći element (čvor) u listi.

Primjer: olančana lista koja sadrži cijele brojeve



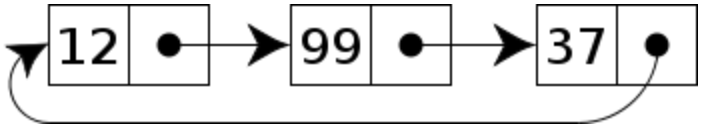
Osnovna prednost u odnosu na nizove je da redosljed podataka različit od onoga u memoriji ili na disku. Dozvoljeno je dodavanje ili uklanjanje čvorova na bilo koju poziciju u listi. Nedostatak je što ne dopuštaju slučajan pristup elementu. Zbog toga mnoge osnovne operacije kao što su pronalaženje posljednjeg čvora u listi ili nalaženje čvora sa zadatom vrijednošću zahtijevaju pretraživanje cijele liste.

Polje u čvoru liste koje sadrži adresu sljedećeg čvora najčešće se naziva ***next* link** ili ***next* pointer**. Polja u kojima se čuvaju informacije su najčešće **data**, **information**, **value**, **cargo** ili **payload. Glava liste (engl. head)** je njen prvi čvor dok je rep liste (engl. **tail**) lista koja se dobija uklonimo prvi čvor (ili, drugim riječima, pokazivač na ostatak liste).

Linearne i kružne (ciklične ili cirkularne) liste

Posljednji čvor liste najčešće sadrži tzv. **null** referencu (u C/C++: NULL, u Pascal-u: nil, u Java: null), koja je posebna vrijednost koja označava da “nema sljedećeg čvora”. Ako posljednji čvor pokazuje na prvi čvor liste, tada kažemo da je lista kružna (cirkularna); u suprotnom, lista je linearna ili otvorena.

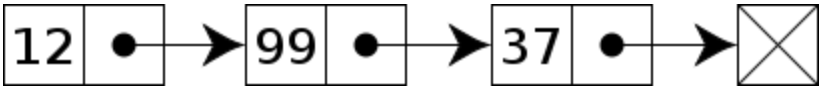
Primjer: olančana kružna lista koja sadrži cijele brojeve



Jednostruko olančane liste i višestruko olančane liste

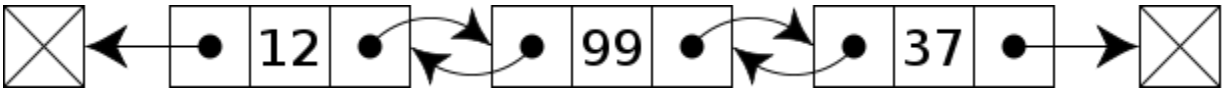
Jednostruko olančana lista (engl. singly-linked list) sadrži čvorove koje imaju data polje i samo jedan pokazivač na sljedeći čvor u listi.

Primjer: Jednostruko olančana lista koja sadrži cijele brojeve



Dvostruko olančana lista (engl. doubly-linked list**)** se sastoji od čvorova koji sadrže po dva pokazivača: jedan na sljedeći čvor liste i jedan na prethodni čvor liste, Ovi se pokazivači najčešće nazivaju **forward**(**s**) i **backwards**, ili **next** i **prev**(**ious**).

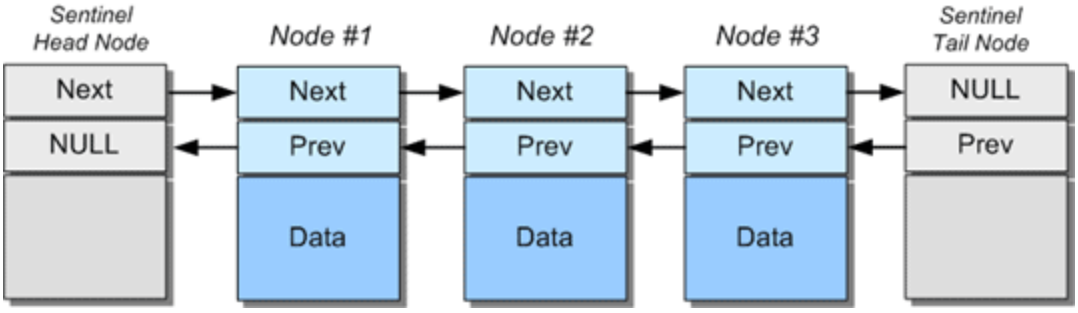
Primjer: Dvostruko olančana lista koja sadrži cijele brojeve



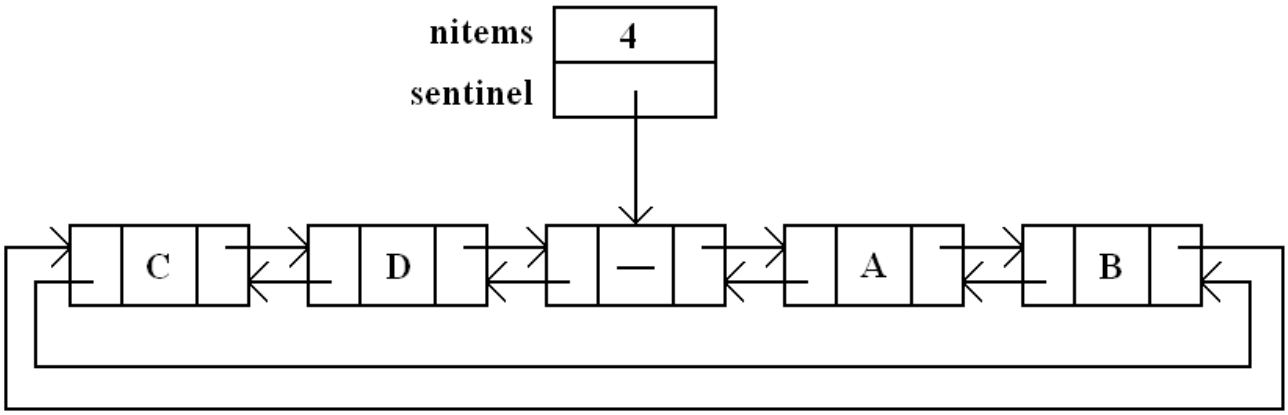
Sentinel čvor

U nekim implementacijama, dodaje se poseban čvor tzv. **sentinel** ili **dummy** koji se dodaje ispred prvog čvora liste i-ili iza posljednjeg čvora u listi. Ova konvencija omogućava lakše manipulisanje listom, jer svi pokazivači mogu biti dereferencirani.

Primjer: Dvostruko olančana lista sa senetinel čvorom i na početku (sentinel head node) i na kraju liste (sentinel tail node ili trailer).



Primjer: Dvostruko olančana kružna lista sa senetinel čvorom i poljem koje čuva broj elemenata u listi.



Prazna lista (empty list)

Prazna lista je ona koja ne sadrži nijedan zapis. Obično se kaže da ima nula čvorova. Ako se koristi sentinel, tada se kaže da je lista prazna ako sadrži samo sentinel čvor.

Olančane liste i dinamički nizovi (Linked lists vs. dynamic arrays)

	Lista	Niz	Dinamički niz
Indeksiranje	Θ(*n*)	Θ(1)	Θ(1)
Dodavanje/brisanje na početku	Θ(1)	N/A	Θ(*n*)
Dodavanje/brisanje na kraju	Θ(1)[1]	N/A	Θ(1)
Dodavanje/brisanje u sredini	traženje + Θ(1)[2]	N/A	Θ(*n*)
Wasted space (average)	Θ(*n*)	0	Θ(*n*)

Operacije sa olančanom listom

Prikazaćemo pseudokod za dodavanje i uklanjanje čvora iz liste

Linearne liste

Jednostruko olančane liste

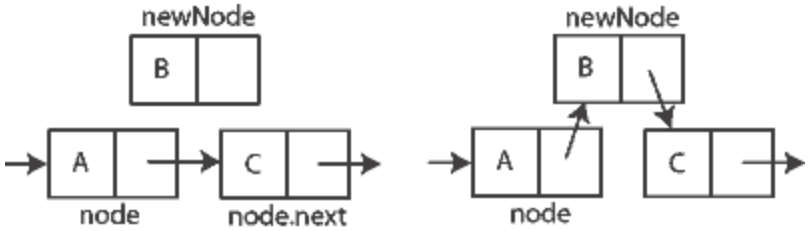
Čvor opisujemo strukturom koja ima 2 polja. *firstNode* uvijek pokazuje na prvi čvor liste ili je *null* ako je lista prazna.

```
record Node {
    data // The data being stored in the node
    next // A reference to the next node, null for last node
 }
record List {
     Node firstNode   // points to first node of list; null for empty list
}
```

Prolazak kroz listu (engl. traversal) – krećemo od prvog čvora i slijedimo pokazivač next dok ne dođemo do kraja

```
 node := list.firstNode
 while node not null {
     (do something with node.data)
     node = node.next
 }
```

Dodavanje novog čvora neposredno iza postojećeg čvora u listi.



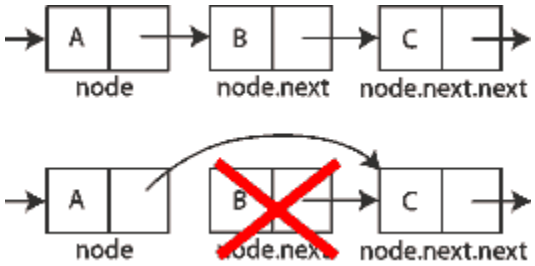

```
function insertAfter(Node node, Node newNode)
{// insert newNode after node
     newNode.next := node.next
     node.next    := newNode
}
```

Dodavanje (umetanje) čvora ispred postojećeg čvora ne može se uraditi direktnom jer je potrebno da znate prethodni čvor i zatim umetnete čvor iza njega.

Umetanje na početak liste zahtijeva promjenu pokazivača *firstNode*.

```
 function insertBeginning(List list, Node newNode) { // insert node before current
first node
     newNode.next   := list.firstNode
     list.firstNode := newNode
 }
```

Slično, funkcije za brisanje čvora iza zadatog čvora i brisanje prvog čvora. Da bi našli i uklonili čvor iz liste, moramo voditi evidenciju o prethodnom čvoru.




```
function removeAfter(node node) { // remove node past this one
     obsoleteNode := node.next
     node.next := node.next.next
     destroy obsoleteNode
 }

 function removeBeginning(List list) { // remove first node
     obsoleteNode := list.firstNode
     list.firstNode := list.firstNode.next   // point past deleted node
     destroy obsoleteNode
 }
```

Funkcija removeBeginning() postavlja *firstNode* na *null* ako uklanjamo jedini element iz liste.

Kako nema pokazivača na prethodni čvor, ne postoji efikasna implementacija operacija "insertBefore" i "removeBefore" operations are not possible.

Mnogi specijalni slučajevi mogu biti eliminisati dodavanjem sentinel čvora. Npr, uz sentinel nije potrebno posebno implementirati insertBeginning() i removeBeginning()

Kružne liste

Kod cikličnih listi, ne koristi se *null* za kraj liste. Polje *next* posljednjeg čvora pokazuje na prvi čvor. Elementi se mogu dodavati na kraj liste i uklanjati sa početka za konstantno vrijeme. Mogu biti jednostruko ili dvostruko olančane. Obilazak liste može početi sa bilo koje pozicije u listi.

Pretpostavimo da je *someNode* neki čvor u nepraznoj cikličnoj jednostruko olančanoj listi:

```
function iterate(someNode)
  if someNode  ≠ null
    node := someNode
    do
      do something with node.value
      node := node.next
    while node ≠ someNode
```

Test "**while** node ≠ someNode" mora biti na kraju petlje, jer u slučaju liste sa jednim elementom funkcija ne bi radila.

Sljedeća funkcija umeće čvor "newNode" u cikličnu listu iza datog čvora "node". Ako je "node" null, smatramo da je lista prazna.

```
function insertAfter(Node node, Node newNode)
    if node = null
      newNode.next := newNode
    else
      newNode.next := node.next
      node.next := newNode
```

Neka je "L" pokazovač na posljednji čvor ciklične liste (ili null ako je lista prazna). Dodavanje čvora "newNode" na kraj liste:

```
insertAfter(L, newNode)
L := newNode
```

Umetanje "newNode" na početak liste:

```
insertAfter(L, newNode)
if L = null
  L := newNode
```

Implementacija liste pomoću niza

Jezici koji ne podržavaju pokazivače mogu zamijeniti pokazivače indeksima niza. Listu implementiramo pomoću niza zapisa, gdje je jedno polje zapisa next koje sadrži indeks sljedećeg zapisa.

Primjer: Implementacija dvostruko olančane liste pomoću niza:

```
 record Entry {
    integer next // index of next entry in array
    integer prev // previous entry (if double-linked)
    string name
    real balance
 }
integer listHead
Entry Records[1000]
```

Veze između elemenata ostvaruju se postavljanjem odgovarajućeg indeksa u polje Next ili Prev. Npr:

Indeks	Next	Prev	Name	Balance
0	1	4	Jones, John	123.45
1	-1	0	Smith, Joseph	234.56
2 (listHead)	4	-1	Adams, Adam	0.00
3			Ignore, Ignatius	999.99
4	0	2	Another, Anita	876.54
5				
6				
7				

ListHead je postavljen na 2, što je lokacija prvog čvora. Polja 3, 5, 6 i 7 nisu dio liste, već se mogu koristiti za dodavanje novih elemenata.

Sljedeći kod obilazi listu i prikazuje imena i brojeve računa:

```
i := listHead
while i >= 0 { '// loop through the list
     print i, Records[i].name, Records[i].balance // print entry
     i := Records[i].next
}
```

Implementacija liste – jezik C

```
#include <stdio.h>    /* for printf */
#include <stdlib.h>   /* for malloc */

typedef struct node {
        int data;
        struct node *next; /* pointer to next element in list */
} LLIST;

LLIST *list_add(LLIST **p, int i);  /* Function definition to add an element */
void list_remove(LLIST **p);        /* Function definition to remove element */
```

```
LLIST **list_search(LLIST **n, int i); /* Function definition to search the list */
void list_print(LLIST *n);            /* Function definition to print the list */
/*************************************************************/
/*  Function: Add an element to our list                     */
/*                                                           */
/*   Parameters: **p is the node that we wish to insert at.  */
/*               if the node is the null insert it at the beginning    */
/*               Other wise put it in the next space                   */

LLIST *list_add(LLIST **p, int i)
{
        if (p == NULL)   /*checks to see if the pointer points somewhere in space*/
                return NULL;

        LLIST *n = (LLIST *)malloc(sizeof(LLIST));   /* creates a new node of the
correct data size */
        if (n == NULL)
                return NULL;

        n->next = *p; /* the previous element (*p) now becomes the "next" element
*/
        *p = n;       /* add new empty element to the front (head) of the list */
        n->data = i;

        return *p;
}

void list_remove(LLIST **p) /* remove head */
{
        if (p != NULL && *p != NULL)
        {
                LLIST *n = *p;
                *p = (*p)->next;
                free(n);
        }
}

LLIST **list_search(LLIST **n, int i)
{
        if (n == NULL)
                return NULL;

        while (*n != NULL)
        {
                if ((*n)->data == i)
                {
                        return n;
                }
                n = &(*n)->next;
        }
        return NULL;
}

void list_print(LLIST *n)
{
        if (n == NULL)
        {
                printf("list is empty\n");
        }
```

```
        while (n != NULL)
        {
                printf("print %p %p %d\n", n, n->next, n->data);
                n = n->next;
        }
}

int main(void)
{
        LLIST *n = NULL;

        list_add(&n, 0); /* list: 0 */
        list_add(&n, 1); /* list: 1 0 */
        list_add(&n, 2); /* list: 2 1 0 */
        list_add(&n, 3); /* list: 3 2 1 0 */
        list_add(&n, 4); /* list: 4 3 2 1 0 */
        list_print(n);
        list_remove(&n);                /* remove first (4) */
        list_remove(&n->next);          /* remove new second (2) */
        list_remove(list_search(&n, 1)); /* remove cell containing 1 (first) */
        list_remove(&n->next);          /* remove second to last node (0) */
        list_remove(&n);                /* remove last (3) */
        list_print(n);

        return 0;
}
```

Implementacija liste – jezik Pascal

```
program PovezaneListe(input, output);

{  Jednostruko povezana lista. Svaka procedura ili funkcija koja ima
   sufiks _N oznacava nerekurzivnu verziju. }

  type
    lista = ^lst;
    lst = record
            glava:integer;
            rep:lista;
          end;

  var
    i,j,n,x:integer;
    p,q, p1,q1:lista;
    file_ime, file_ime_new:string;

 (*****************************************************************)

  function read_list:lista;
     { Ucitavanje nekoliko brojeva sa standardnog ulaza, iz jednog reda
      i njihovo smjestanje u listu. Redosled brojeva u listi je isti kao na ulazu.
      Kraj ulaza je <Enter>. }

     var
        l:lista;

     begin {read_lista}
```

```
      if eoln then
         read_list:=nil
      else
         begin
           new(l);
           read(l^.glava);
           l^.rep:=read_list;
           read_list:=l
         end;
   end;  {read_lista}

(*********************************************************************)

 procedure dodaj(el:integer; var l:lista);

   { Dodavanje elementa el u listu, na prvu poziciju. }

   var
      h:lista;

   begin { dodaj }
         new(h);
         h^.glava:=el;
         h^.rep:=l;
         l:=h;
   end;  { dodaj }

(*********************************************************************)

 function read_list_N:lista;

   { Ucitavanje nekoliko brojeva sa standardnog ulaza, u jednom redu,
     i njihovo smjestanje u listu. Redosled brojeva u listi je obrnut od
    redosleda na ulazu. Kraj ulaza je <Enter>. }

   var
      l:lista;
      x:integer;

   begin { read_list_N }
     l:=nil;
     while not eoln  do
       begin
          read(x);
          dodaj(x,l);
       end;
     read_list_N:=l;
   end;  { read_list_N }
 (*********************************************************************)

 function read_list_file(ime_dat:string):lista;

   { Ucitavanje nekoliko brojeva iz tekstualne datoteke ime_dat,
     i njihovo smjestanje u listu. Redosled brojeva u listi je obrnut od
    redosleda na ulazu. Kraj ulaza je <Enter>. }

   var
      l:lista;
      x:integer;
```

```
      fd:text;

   begin { read_list_file }
     assign(fd,ime_dat);
     reset(fd);
     l:=nil;
     while not eof(fd)  do
       begin
          readln(fd,x);
          dodaj(x,l);
       end;
     read_list_file := l;
     close(fd);
   end;  { read_list_file }
(*********************************************************************)

 procedure delete(x:integer; var l:lista);

   { Brisanje svih elemenata jednakih broju x iz liste. }

   var
      h:lista;

   begin { dodaj }
     if l <> nil then
       if l^.glava = x then
         begin
           l:=l^.rep;
           delete(x,l)
         end
       else
         delete(x,l^.rep);
   end;  { dodaj }

*********************************************************************)

 procedure delete_N(x:integer; var l:lista);

   { Brisanje elementa jednakog broju x iz liste. }

   var
      h,f:lista;

   begin { dodaj }
     h:=l;
     while (h^.rep <> nil) and (h^.glava <> x) do
       begin
         f:=h;
         h:= h^.rep;
       end;
     if h^.glava = x then
         begin
           f^.rep := h^.rep;
           dispose(h);
         end;
   end;  { dodaj }

(*********************************************************************)
```

```
 procedure write_list(l:lista);
    { Stampanje liste na standardni izlaz. }
    begin {write_list}
      if l<>nil then
        begin
          write(l^.glava:4);
          write_list(l^.rep);
        end;
    end;  {write_list}

(*********************************************************************)

 procedure write_list_N(l:lista);
    { Stampanje liste na standardni izlaz - nerekurzivna varijanta . }

    begin {write_list_N}
      while l<>nil do
        begin
          write(l^.glava:4);
          l:=l^.rep;
        end;
    end;  {write_list_N}

(*********************************************************************)

 procedure write_list_file(l:lista; ime_dat:string);

    { Stampanje liste u tekstualnu datoteku cije je
      ime ime_dat - nerekurzivna varijanta. U jednom redu
      upisuje se jedan element datoteke. }

    var
      fd:text;

    begin {write_list_file}

      assign(fd,ime_dat);
      rewrite(fd);
      while l<>nil do
        begin
          writeln(fd,l^.glava);
          l:=l^.rep;
        end;
      close(fd);
    end;  {write_list_file}

(*********************************************************************)

 function Length(l:lista):integer;
  { Duzina liste. }
  begin
     if l= nil then
      length:=0
     else
       length:= 1 + length(l^.rep);
  end;

(*********************************************************************)
```

```
  function Length_N(l:lista):integer;
   { Duzina liste nerekurzivno. }
   var
     n:integer;
   begin
      {p:=l;}
      n:=0;
      while (l <> nil) do
        begin
          n:=n+1;
          l:=l^.rep;
        end;
      length_N :=n
   end;
 (*********************************************************************)

  function member(el:integer; l:lista):boolean;
     { Ispitivanje da li je element el u listi. }

     begin { member }
        while (l<>nil) and (l^.glava<>el) do
            l:=l^.rep;
        member:=(l<>nil);
     end;  { member }

 (*********************************************************************)

  procedure insert(el,y:integer; l:lista);
     { Dodavanje elementa el u listu, neposredno iza elementa y. }
     var
        h:lista;

     begin { insert }
        while (l<>nil) and (l^.glava<>y) do
          l:=l^.rep;
        if l<> nil then
           begin
              new(h);
              h^.glava:=el;
              h^.rep:=l^.rep;
              l^.rep:=h;
           end;
     end;  { insert }
```

Implementacija liste – jezik Java

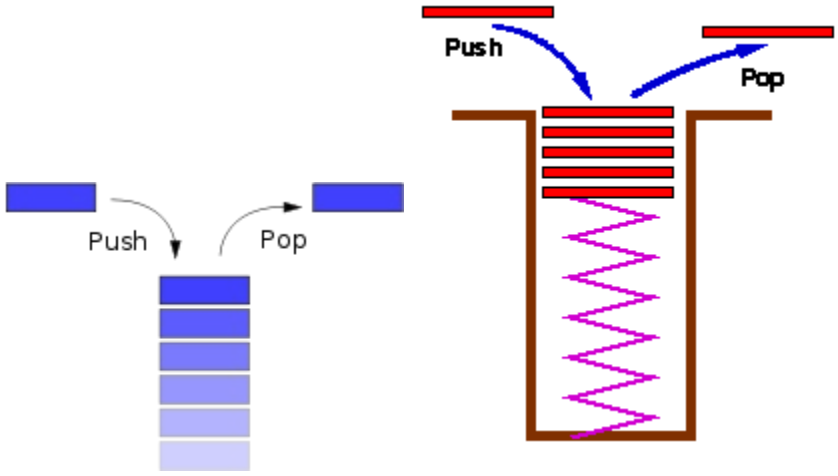
```
// Klasa Lista
/* Jednostruko olancana lista cijelih brojeva
 *
 * */

public class Lista {
      int val;
      Lista next;

      public Lista(int v)
```

```
	{
		val = v;
		next = null;
	}

	public Lista add(int v)
	{
		Lista a = new Lista(v);
		a.next = this;
		return a;
	}

}


//  Klasa TestLista
import java.util.Random;
import java.util.Scanner;

public class TestList {

	/**
	 * @param args
	 */

	public static void stampa(Lista l)
	{
		Lista curr = l;
		while (curr != null)
		{
			System.out.printf("%4d", curr.val);
			curr = curr.next;
		}
		System.out.printf("\n");
	}

	public static void stampaRec(Lista l)
	{
		if (l != null)
		{
			System.out.printf("%4d", l.val);
			stampaRec(l.next);
		}
		else
		{
			System.out.printf("\n");
		}
	}

	/* Dodavanje elementa na pocetak liste*/
	public static void dodaj(Lista l, int a)
	{
		Lista t = new Lista(a);
		t.next = l;
		l = t;
	}

	/* Rekurzivno brisanje prvog pojavljivanja broja a u listi l */
	public static Lista brisiRec(Lista l, int a)
```

```
{
    if (l != null)
      if (l.val == a)
        l = l.next;
      else
        l.next = brisiRec(l.next, a);

    return l;
  }

/* Brisanje prvog pojavljivanja broja a u listi l */
public static Lista brisi(Lista l, int a)
{
      if (l == null) return l;
      Lista curr = l, prev = null;
      while (curr!=null && curr.val != a)
      {
            prev = curr;
            curr = curr.next;
      }
      if (curr!=null)
      {
            if (prev!=null)
            {
                  prev.next = curr.next;
            }
            else
            {
                  curr = l.next;
                  l = curr;
            }
      }
      return l;
}

public static int brojElemenata(Lista l)
{
      Lista curr = l;
      int cnt = 0;
      while (curr != null)
      {
            cnt++;
            curr = curr.next;
      }
      return cnt;
}

public static int brojElemenataN(Lista l)
{
      int cnt = 0;
      while (l != null)
      {
            cnt++;
            l = l.next;
      }
      return cnt;
}

public static int brojElemenataRec(Lista l)
```

```
	{
		if (l == null) return 0;
		return 1+brojElemenataRec(l.next);
	}

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		Lista l = new Lista(5);
		Random r = new Random();
		int x;
		for (int i = 0; i<10; i++)
		{
			x = 1 + r.nextInt(20);
			l = l.add(x);
		}
		stampa(l);
		stampaRec(l);
		System.out.printf("%4d\n", brojElemenataRec(l));
		System.out.printf("%4d\n", brojElemenata(l));
		for (int i = 0; i<5; i++)
		{
			x = 1 + r.nextInt(20);
			dodaj(l,x);
		}
		stampa(l);
		stampaRec(l);
		System.out.printf("%4d\n", brojElemenataRec(l));
		System.out.printf("%4d\n", brojElemenata(l));
		System.out.printf("%4d\n", brojElemenataN(l));
		stampa(l);

		Scanner ulaz = new Scanner(System.in);
		System.out.printf("Unesite broj koji zelite da obrisete iz liste:");
		x = ulaz.nextInt();
		l = brisiRec (l, x);
		stampa(l);
		l = brisi(l, x);
		stampa(l);

	}

}
```

Stek (Stack)



Stek je last in, first out (LIFO) apstraktni tip podataka. Na steku se mogu čuvati bilo koji elementi. Postoje dvije osnovne operacije sa stekom: *push* i *pop*. Operacija push dodaje element na vrh steka (engl. top of the stack). Operacija pop uklanja element sa vrha steka i vraća ga pozivaču.

Elementi se uklanjaju sa steka u redosljedu obrnutom od onoga u kojem su dodavani.

Druge operacije sa stekom

Pored operacija "push" i "pop", postoje i operacije top, remove i isempty..

Apstraktna definicija

Ako N označava element (u ovom slučaju prirodan broj) i U označava uniju:

```
init: -> Stack
push: N x Stack -> Stack
top: Stack -> (N U ERROR)
remove: Stack -> Stack
isempty: Stack -> Boolean
```

Semantika

```
top(init()) = ERROR
top(push(i,s)) = i
remove(init()) = init()
remove(push(i, s)) = s
isempty(init()) = true
isempty(push(i, s)) = false
```

Implementacija

U većini jezika, stek se može implementirati pomoću niza ili olančane liste.

Implementacija - jezik C/C++

Pomoću niza

Kreira se niz u kome je prvi element onaj koji se stavlja na stek i posljednji element koji se uklanja sa steka. Stek se inplementira kao struktura sa dva polja:

```
typedef struct {
    int size;
    int items[STACKSIZE];
} STACK;
```

Funkcija `push()` se koristi da inicijalizuje stek i da sačuva vrijednost. Takođe, provjerava se da li ima prostora u nizu za novi element.

```
void push(STACK *ps, int x)
{
    if (ps->size == STACKSIZE) {
        fputs("Error: stack overflow\n", stderr);
        abort();
    } else
        ps->items[ps->size++] = x;
}
```

Funkcija `pop()` uklanja elementa sa steka i provjerava da li je stek prazan.

```
int pop(STACK *ps)
{
    if (ps->size == 0){
        fputs("Error: stack underflow\n", stderr);
        abort();
    } else
        return ps->items[--ps->size];
}
```

Pomoću olanačane liste

Ova implementacija steka je još prostija od one pomoču niza. Obična olančana lista u kojoj je moguće dodavanja na vrh liste i brisanje sa vrha liste završava posao:

```
typedef struct stack {
    int data;
    struct stack *next;
} STACK;
```

Ovakav čvor je identičan čvoru u tipičnoj olančanoj listi u jeziku C.

Funkcija `push()` ekvivalentna je dodavanju novog čvora na početak liste:

```
void push(STACK **head, int value)
```

```
    {
        STACK *node = malloc(sizeof(STACK));  /* create a new node */

        if (node == NULL){
            fputs("Error: no space available for node\n", stderr);
            abort();
        } else {        /* initialize node */
            node->data = value;
             /* insert new head if any */
            node->next = empty(*head)?NULL:*head;
            *head = node;
        }
    }
```

Funkcija pop( ) uklanja glavu liste:

```
    int pop(STACK **head)
    {
        if (empty(*head)) {                       /* stack is empty */
           fputs("Error: stack underflow\n", stderr);
           abort();
        } else {                                  /* pop a node */
            STACK *top = *head;
            int value = top->data;
            *head = top->next;
            free(top);
            return value;
        }
    }
```

Implementacija - jezik Pascal

Pomoću niza

```
Program StekNiz;

const
        EmptyTOS = 0;
        MinStackSize =  5;
        Duz = 100;
Type
        ElementType = integer;

        niz = array[1..DUZ] of ElementType;

        Stack = record
              Capacity:integer;
              TopOfStack:integer;
              Memory: niz ;
        end;

var
        st1:stack;
        el:ElementType;
```

```
        i:integer;

Function IsEmpty( S:Stack ):boolean;
begin
        IsEmpty := S.TopOfStack = EmptyTOS;
end;

Function IsFull( S:Stack ):boolean;
begin
        IsFull := S.TopOfStack = S.Capacity ;
end;

Procedure MakeEmpty( var S:Stack );
begin
        S.TopOfStack := EmptyTOS;
end;

Procedure CreateStack( MaxElements:integer; var S:Stack );
begin
        if( MaxElements < MinStackSize ) or (MaxElements > Duz) then
        writeln( 'Stek nije odgovarajuce velicine. Mora imati vise od ',
MinStackSize, ' i manje od ', Duz, ' elemenata.' )
    else
        begin
                S.Capacity := MaxElements;
                S.TopOfStack := 1;
        end;
        MakeEmpty( S );
end;

Procedure Push( X:ElementType; var S:Stack );
begin
        if( IsFull( S ) ) then
                writeln( 'Stek je pun!' )
        else
                begin
                        S.TopOfStack := S.TopOfStack + 1;
                        S.Memory[ S.TopOfStack ] := X;
                end;
end;

Function Pop( var S:Stack ):ElementType;
begin
        if( not IsEmpty( S ) ) then
                begin
                        Pop := S.Memory[ S.TopOfStack ];
                        S.TopOfStack := S.TopOfStack - 1;
                end
        else
                begin
                writeln( 'Stek je prazan!' );
                pop := 0;
        end;
end;

procedure DisposeStack(var S:Stack );
begin
     while (not IsEmpty(S)) do
      writeln(Pop(S));
```

```
end;
```

Pomoću olančane liste

```
program stek(input,output);
 type sledeci = ^Lista;
       Lista  = record
                  el:integer;
                  next:sledeci;
       end;
 var p,q :sledeci;
     x:integer;

procedure push(var lst:sledeci);
  var  p:sledeci;
Begin
       new(p);
       read(p^.el);
       p^.next := lst;
       lst:= p;
end; {Procedure push}

procedure pop (var lst:sledeci);
   var p:sledeci;
begin
        if lst = nil then
           writeln('Prazan')
        else
        begin
             p:=lst;
             lst:= lst^.next;
             writeln(p^.el);
             dispose(p);
        end;
end; {pop}
```

Implementacija - jezik C++ (STL)

```
#include <iostream>
#include <stack>

using namespace std;

int main ()
{
    stack <string> cards; /* stek stringova */
    cards.push("King of Hearts"); /* dodajemo kartu */
    cards.push("King of Clubs");  /* dodajemo kartu */
    cards.push("King of Diamonds");
    cards.push("King of Spades");
    cout << "U spilu imamo " << cards.size () << " karata " << endl;
    cout << "Karta na vrhu je " << cards.top() << endl;
    /* Stampace King of Spades, jer je on posljednji dodat */
    cards.pop();
    cout << "Karta na vrhu je " << cards.top() << endl;
    cout << cards.size();
    cin.get ();
    return EXIT_SUCCESS;
```

```
}
```

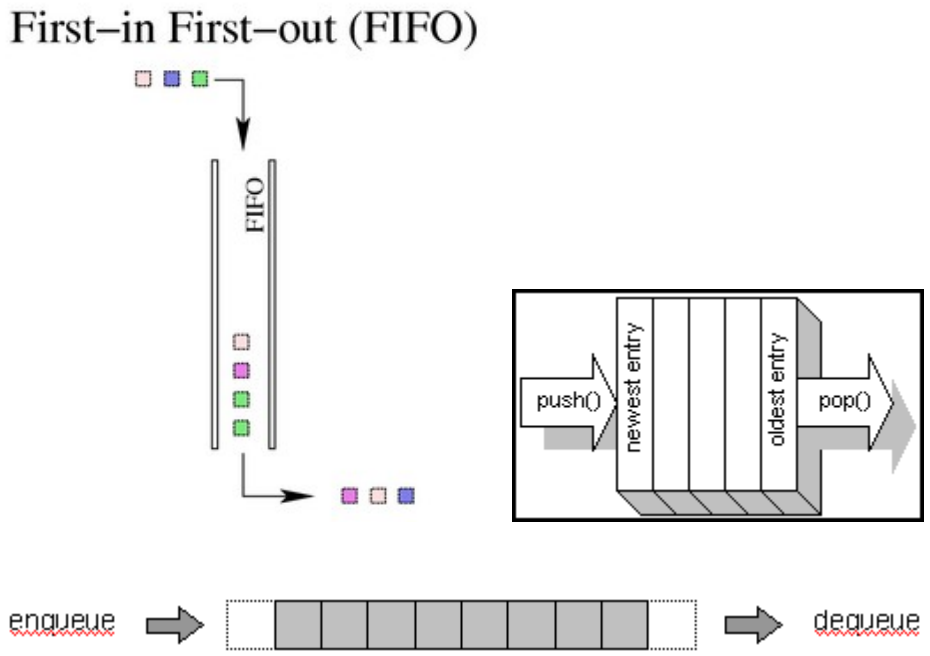
Implementacija - jezik Java

Koristimo ugrađenu klasu `Stack` iz paketa `java.util.*`. Objekti ove klase (kao npr. `st`) ne mogu da primaju primitivne tipove (kao što su `int`, `double`, `boolean`, `char`) već samo objekte. Zbog toga se, u našem primjeru, cio broj `x` pretvara u objekat klase `Integer` pri upisivanju u stek (kod operacije `push`) i prilikom izvlačenja iz steka (operacija `pop`).

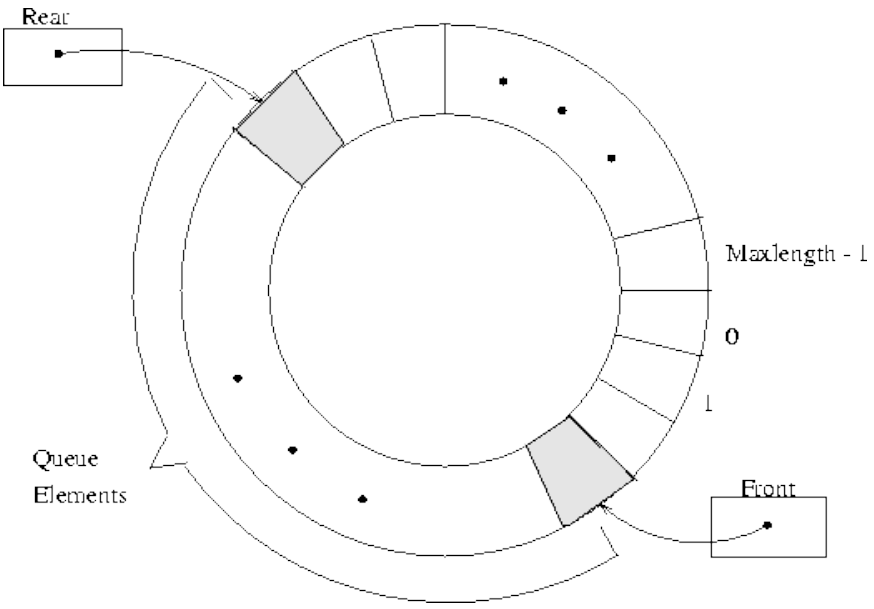
```
import java.util.*;
...
Stack st = new Stack();
Random r = new Random();

for (int i = 0; i<10; i++) // popunjavamo stek sa 10 slucajnih brojeva
{
      int x = r.nextInt(100);
      st.push(new Integer(x));
}

Integer a;
System.out.printf("\nSadrzaj steka\n");
while (!st.empty())
{
      a = (Integer)st.pop();
      System.out.printf("%4d", (int)a.intValue());
}
```

Red (Queue)

Red je first in, first out (FIFO) apstraktni tip podataka. U redu se mogu čuvati bilo koji elementi. Postoje dvije osnovne operacije sa redom: *`push`* (`ili` *`enqueue ili addQ`*) i *`pop`* (`ili` *`dequeue`* `ili` *`removeQ`*). Operacija *`push`* (*`enqueue`*) dodaje element na kraj reda (engl. *`rear`* ili *`last`* ili `back`). Operacija *`pop`* (*`dequeue`*) uklanja element sa vrha reda (engl. *`front`* ili *`first`*) i vraća ga pozivaču. Elementi se uklanjaju iz reda u istom redosljedu u kojem su dodavani.



Implementacija preko cirkularnog niza



Koristimo cjelobrojne front i rear.

- front je jednu poziciju suprotno kazaljki sata od prvog elementa
- rear daje poziciju posljednjeg elementa

Operacija dodavanja elementa:

- Pomjeri se rear jedno mjesto u smjeru kazaljke na satu.
- Zatim se element stavi u queue[rear].

Operacija uklanjanja elementa:

- Pomjeri se front jednu poziciju u smjeru kazaljke sata.
- Zatim se vrati queue[front].

Pomjeranje u smjeru kretanja kazaljke sata (i za front i za rear):

```
    rear++;
    if (rear = = capacity) rear = 0;
ili  rear = (rear + 1) % capacity;
```

Situacija front==rear se pojavljuje u dvije situacije:

- Red je prazan
- Red je pun (svi elementi niza su popunjeni).

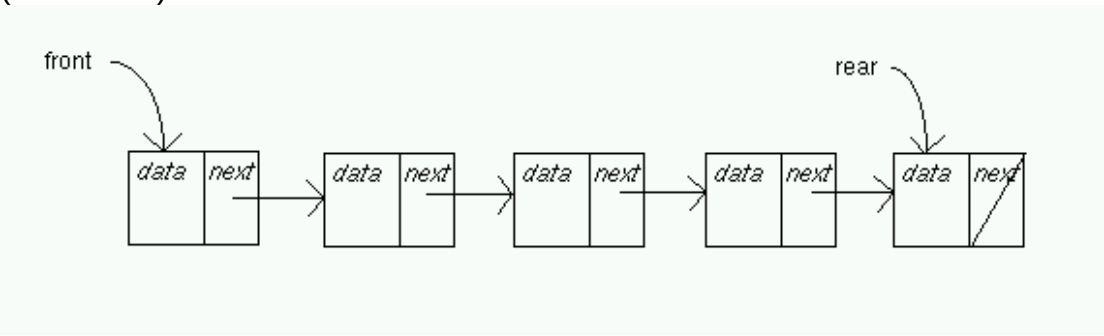
Da bi otklonili ovaj problem, možemo uraditi sljedeće:

- Ne dozvoliti da se red napuni.
  - Ako dodavanje elementa uzrokuje prepunjavanje, povećati veličinu niza (moguće u C/C++ i Java-i, nije moguće u Pascal-u).
- Definisati Bulovsku promjenljivu lastOperationIsAddQ.
  - Svako dodavanje u red postavlja ovu promjenljivu na true.
  - Svako brisanje iz reda postavlja ovu promjenljivu false.
  - Red je prazan ako je (front == rear) && !lastOperationIsAddQ
  - Red je pun ako je (front == rear) && lastOperationIsAddQ
- Definisati cjelobrojnu promjenljivu size.
  - Svako AddQ odrađuje i size++.
  - Svako DeleteQ odrađuje i size--.
  - Red je prazan ako je (size == 0)
  - Red je pun ako je (size == arrayLength)

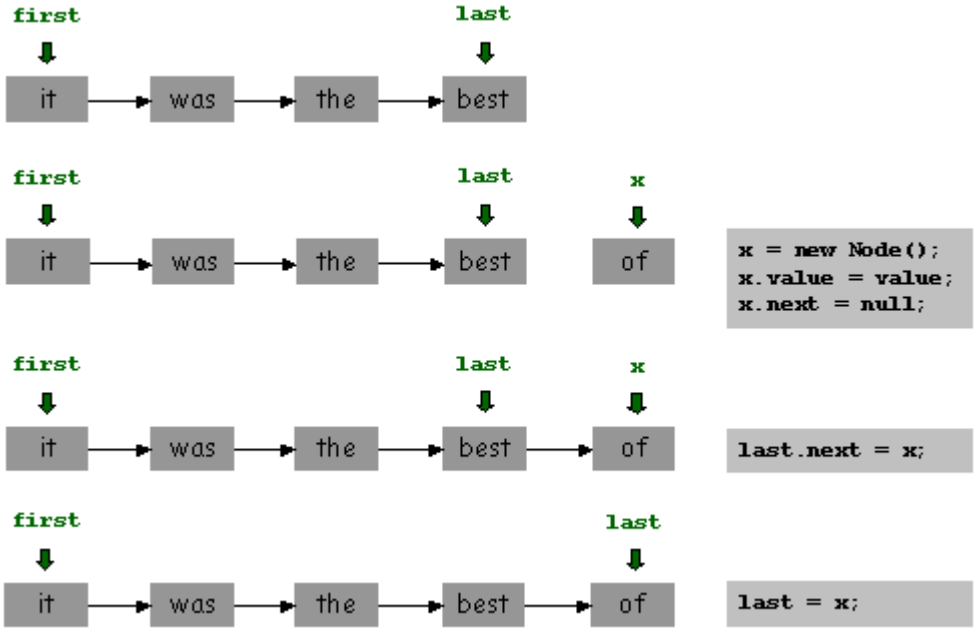
Za detalje kako rade operacije dodavanja i uklanjanja elementa u red pogledati prezentaciju Queue.ppt.

Implementacija preko olančane liste

Red se opisuje jednostruko olančanom listom ili dvostruko olančanom listom. U oba slučaja imamo pokazivač na prvi element liste (first ili front) i pokazivač na posljednji element liste (rear ili last).



Primjer dodavanja elementa prikazan je na sljedećoj slici:



Implementacija – jezik C++ (STL)

```
/* STL queue primjer */

#include <iostream>
#include <queue>

using namespace std;

/* simple queue example */
void functionA()
{
  queue <int> q;                  //q je red cijelih brojeva

  q.push(2);                      //dodajemo 2, 5, 3, 2 u red
  q.push(5);
  q.push(3);
  q.push(1);
  cout<<"q ima " << q.size() << " elemenata.\n";

  while (!q.empty()) {
    cout << q.front() << endl;    //stampa prvi element u redu
    q.pop();                      //brise prvi element iz reda
  }
}

int main()
{
  cout << "calling functionA...\n";
  functionA();

  return 0;
}
```

Implementacija – jezik Java

Postoji interfejs Queue, ali kao i za klasu Stack, u redu moramo čuvati objekte a ne primitivne tipove. Osnovne operacije sa redom su prikazane u tabeli:

Queue Interface Structure

Operacija	Throws exception	Returns special value
Insert	`add(e)`	`offer(e)`
Remove	`remove()`	`poll()`
Examine	`element()`	`peek()`

Primjer:

```
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Queue;

public class QueueExample {

    public static void main(String[] args) {

        Queue<String> qe=new LinkedList<String>(); // red stringova
        // dodavanje 5 stringova
        qe.add("b");
        qe.add("a");
        qe.add("c");
        qe.add("e");
        qe.add("d");

        Iterator it=qe.iterator();

        System.out.println("Initial Size of Queue :"+qe.size());

        while(it.hasNext())
        {
            String iteratorValue=(String)it.next();
            System.out.println("Queue Next Value :"+iteratorValue);
        }

        // get value and does not remove element from queue
        System.out.println("Queue peek :"+qe.peek());

        // get first value and remove that object from queue
        System.out.println("Queue poll :"+qe.poll());

        System.out.println("Final Size of Queue :"+qe.size());
    }
}
```

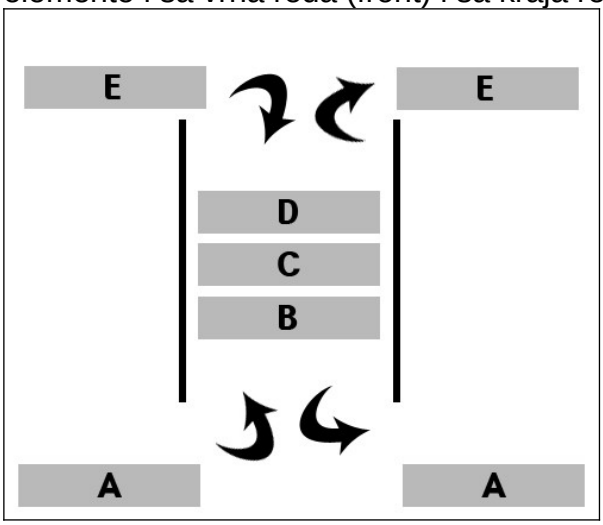
Izlaz:

Initial Size of Queue :5
Queue Next Value :b
Queue Next Value :a

Queue Next Value :c
Queue Next Value :e
Queue Next Value :d
Queue peek :b
Queue poll :b
Final Size of Queue :4

Dvostrani red (dequeue, deque, deck)

Samo ime strukture nam govori o čemu se radi: red kod koga je moguće dodavati i brisati elemente i sa vrha reda (front) i sa kraja reda (rear ili back).



Načini implementacije su isti kao i kod reda. Stek i red možemo posmatrati kao specijalne slučajeve ove strukture.
U jezicima C++ i Java postoje implementacija ove strukture. Spisak operacija dat je u tabeli:

Operacija	C++	Java
dodavanje elementa na kraj	`push_back`	`offerLast`
dodavanje elementa na početak	`push_front`	`offerFirst`
brisanje posljednjeg elementa	`pop_back`	`pollLast`
brisanje prvog elementa	`pop_front`	`pollFirst`
vrijednost posljednjeg elementa	`back`	`peekLast`
vrijednost prvog elementa	`front`	`peekFirst`

Implementacija – jezik C++ (STL)

U jeziku C++, STL ima podršku za dvostrani red – potrebno je odraditi `#include <deque>`.

```
#include <iostream>
#include <stdexcept>   // zbog excepions
#include <deque>

using namespace std;

int main() {
        //deque sa 5 elemenata, svi imaju vrijednost 8.1
        deque<double> dq(5, 8.1);

        for (int i=0; i<=5; ++i) {
                cout << "Element "<< i << " sa [] : "<< dq[i]<< endl;
```

```
        }

        try {
            cout << "Element " << i << " sa at(i) :" << dq.at(i) << endl;
        } catch (out_of_range&) {
            cout << "**out of range za element " << i << " with at() **" << endl;
        }

        return 0;
}
```

Implementacija – jezik Java

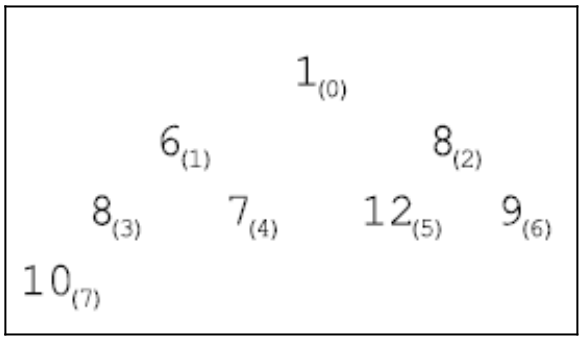
U programskom jeziku Java, `Deque` (obratite pažnju na skraćeni naziv) je intefejs koji se može implementirati preko niza (`ArrayDeque`) ili preko olančane liste (`LinkedList`).

Primjer Java koda koji koristi ovu strukturu:

```
import java.util.Deque
...
Deque dequeA = new LinkedList();
Deque dequeB = new ArrayDeque();
...
```

Hrpa (heap)

Hrpa ili gomila (engl. heap) je struktura podataka koja vraća minimalni (ili maksimalni) element za O(1), a dodaje novi element i uklanja rekući minimum (masksimum) za O(logn). Druge operacije za hrpu obično nisu definisane. Ova se struktura često koristi za implementaciju prioritetnog reda.



Pogledajmo kako se hrpa može realizovati pomoću niza. Elementi sa indeksima 2*i+1 i 2*i+2 su potomci elementa sa indeksom i, indeksi počinju od 0. Potomci dva različita elementa se ne sijeku i svaki element je nečiji potomak (osim elementa sa indeksom 0). Osnovno svojstvo hrpe je da je svaki element manji ili jednak od svojih potomaka. Na primjer, niz 1,6,8,7,12,8,10 je hrpa.

Kreirajmo strukturu za hrpu:

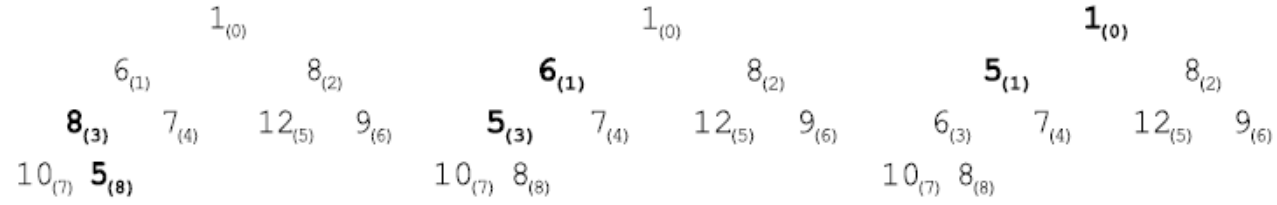
```
typedef struct
{
      int hs;
      int val[MAXN];
} heap;
```

Opišimo tri funkcije za rad sa hrpom. Prvo je funkcija koja vraća najmanji elemenet. Ona je jednostavna jer se najmanji element nalazi na pozicji 0.

```
int get_min(heap *h)
{
return h->val[0];
}
```

Prije poziva ove funkcije, obavezno provjeriti da hrpa nije prazna.

Druga funkcija je za dodavanje elementa u hrpu. Dodamo ga na kraj niza, a zatim ga zamjenjujemo sa njegovim precima sve dok ne postane manji od pretka ili dok indeks ne bude 0. Svojstvo hrpe se ne narušava, a složenost operacije je O(logn). Primjer dodavanja u hrpu prikazan je na slici:




```
void add_heap(heap *h, int x)
{
      int y, pos=h->hs, npos;
```

```
	h->val[h->hs++] = x;
	npos=(pos-1)/2;
	while (pos && h->val[pos] < h->val[npos]) {
		y=h->val[pos];
		h->val[pos]=h->val[npos];
		h->val[npos] = y;
		pos=npos;
		npos=(pos-1)/2;
	}
}
```

Treća funkcija je za udaljavanje minimalnog elementa iz hrpe. Na mjesto elementa sa indeksom 0 zapišemo posljednji element niza, smanjimo veličinu niza za 1 i zatim poguramo element iz korijena na njegovo pravo mjesto. Ako je element veći od manjeg od svojih potomaka, onda oni zamijene mjesta; produžavamo postupak dok nije zadovoljen uslov ili dok ne izađemo van granica hrpe.

```
void del_heap(heap *h)
{
	int minp, pos=0, y;
	h->val[0]=h->val[--h->hs];
	while (pos*2+1 < h->hs) {
		y=pos*2+1;
		minp=h->val[y]<h->val[y+1]?y:y+1;
		if (h->val[pos]>h->val[minp]) {
			y=h->val[pos];
			h->val[pos]=h->val[minp];
			h->val[minp]=y;
			pos=minp;
		}
		else break;
	}
}
```

Na prvi pogled, jedino mjesto gdje je teorijski moguća greška jeste kada imamo samo jednog potomka, tj. pos*2+2 == h->hs. Takva varijanta je moguća ako je broj elemenata u hrpi paran. U tom slučaju, mi određujemo manji od prvog potomka i prvo elementa van hrpe, što izgleda kao greška. Međutim, znamo da su element koji guramo i prvi element van granice niza jednaki, pa neće biti greške.

Implementacija – jezik Java

Implementacija hrpe pomoću niza, koja u potpunosti odgovara gore opisanim funkcijama.

```
public class Heap {

	int hs; // velicina hrpe
	int[] val; // niz za cuvanje elemenata

	public Heap()
	{
		hs = 0;
		val = new int[10];
	}
```

```
	public Heap(int n)
	{
		if (n<0) n = 10;
		hs = 0;
		val = new int[n];
	}

	public int get_min()
	{
		return val[0];
	}

	public void add_heap( int x)
	{
		int y, pos= hs, npos;
		val[hs++] = x;
		npos=(pos-1)/2;
		while (pos>0 && val[pos] < val[npos]) {
			y= val[pos];
			val[pos]=val[npos];
			val[npos] = y;
			pos=npos;
			npos=(pos-1)/2;
		}
	}

	public void del_heap()
	{
		int minp, pos=0, y;
		val[0]= val[--hs];
		while (pos*2+1 < hs) {
			y=pos*2+1;
			minp= val[y]<val[y+1]?y:y+1;
			if (val[pos]>val[minp]) {
				y=val[pos];
				val[pos]=val[minp];
				val[minp]=y;
				pos=minp;
			}
			else break;
		}
	}

	public void print()
	{
		for(int i=0; i<hs; i++)
		{
			System.out.printf("%4d", val[i]);
		}
		System.out.printf("\n");
	}

} // end class Heap


public class TestHeap {

	public static void main(String[] args) {
```

```
            Heap h = new Heap(30);

            h.add_heap(1);
            h.add_heap(12);
            h.add_heap(8);
            h.add_heap(6);
            h.add_heap(7);
            h.add_heap(8);
            h.print();
            System.out.printf("Minimum je %4d\n", h.get_min());
            h.del_heap();
            h.print();
        }

}
```

Prioritetni red (priority queue)

Prioritetni red je apstraktni tip podataka gdje je svakom elementu pridružen prioritet. Ovaj tip podataka mora implementirati bar sljedeće dvije operacije:

- insert_with_priority: dodajemo element sa pridruženim prioritetom u red
- pull_highest_priority_element: brišemo element sa najvećim prioritetom iz reda i vraćamo ga pozivaču (takođe se može zvati "pop_element(Off)", "get_maximum_element" ili "get_front(most)_element"). Neke implementacije smatraju da elementi čiji je prioritet manji broj imaju veći prioritet, pa se ova operacija često naziva i "get-min".

Prioritetni red može se implemenirati na više načina: pomoću niza, liste, hrpe (heap) itd.

Implementacija – jezik C++ (STL)

STL ima podršku za prioritetni red. Dovoljno je napočetku koda dodati `#include <queue>`. Važno je da se elementi koje dodajemo u prioritetni red mogu upoređivati, tj. da je za njih definisan operator manje (<). To se posebno odnosi ako u red dodajemo objekte neke klase – tada se za tu klasu mora definisati komparator ili preopteretiti operator <. Ako u red dodajemo elemente tipa za koji je definisan operator <, onda nije potrebno ništa raditi, ako je to poredak koji nam odgovara (kao u prvom primjeru niže). U drugom primjeru, naredba `priority_queue <int, vector<int>, greater<int> > pq` nam kaže da je `pq` prioritetni red cijelih brojeva koji se implementira pomoću strukture `vector`, a kao operator poređenja se ne koristi standardni operator manje (<) već operator veće (>, zbog `greater<int>`).

```
/* STL priority queue primjeri */

#include <iostream>
#include <queue>

using namespace std;

/* primjer upotrebe prioritetnog reda */
void functionB()
{
  priority_queue <int> pq;      //pq je prioritetni red cijelih brojeva

  pq.push(2);                   //dodajmo 2, 5, 3, 1 u red
  pq.push(5);
  pq.push(3);
  pq.push(1);
  cout<<"pq sadrzi " << pq.size() << " elemenata.\n";

  while (!pq.empty()) {
    cout << pq.top() << endl;   // stampamo element sa najvecim prioritetom
    pq.pop();                   // brisemo element sa najvecim prioritetom
  }
}

/* primjer prioritetnog reda gdje veci prioritet ima manji broj */
void functionC()
```

```
{
  priority_queue <int, vector<int>, greater<int> > pq;

/* pq je proritetni red cijelih brojeva koji se implementira preko vektora korsiti
operator > za odredjivanje prioriteta (tj. ako je a>b, tada a ima manji prioritet
od b)
*/

  pq.push(2);                       //dodamo 2, 5, 3, 2 u red
  pq.push(5);
  pq.push(3);
  pq.push(1);
  cout<<"pq sadrzi " << pq.size() << " elemenata.\n";

  while (!pq.empty()) {
    cout << pq.top() << endl;      // stampamo element sa najvecim prioritetom
    pq.pop();                      // brisemo element sa najvecim prioritetom
  }
}

/* definicija klase Height */
class Height
{
  public:
    Height() {};                                        //default konstruktor
    Height(int x, int y) { feet = x; inches = y; }    //konstructor
    bool operator<(const Height&) const;              //overloaded < operator

    int get_feet() const { return feet; }             //accessor methods
    int get_inches() const { return inches; }

  private:
    int feet, inches;                                 //data fields
};

/* overload operator < da bi znali kako da uporedimo 2 objekta klase Height */
bool Height::operator<(const Height& right) const
{
      return feet*12 + inches < right.feet*12 + right.inches;
}

/* primjer prioritetnog reda koji koristi klasu Height */
void functionD()
{
  priority_queue <Height> pq;      //pq is a priority queue of Height objects

  Height x;

  x = Height(10,20);      //dodajemo put 10'20", 11'0", 8'25" i 9'4" u red
  pq.push(x);
  x = Height(11,0);
  pq.push(x);
  x = Height(8,25);
  pq.push(x);
  x = Height(9,4);
  pq.push(x);

  cout<<"pq contains " << pq.size() << " elements.\n";
```

```
  while (!pq.empty()) {
    cout << pq.top().get_feet()
         << "' " << pq.top().get_inches() << "\"" << endl;
    pq.pop();
  }
}

int main()
{
  cout << "calling functionB...\n";
  functionB();
  cout << "calling functionC...\n";
  functionC();
  cout << "calling functionD...\n";
  functionD();

  return 0;
}
```

Implementacija – jezik Java

Klasa `PriorityQueue` je implementacija prioritetnog reda. Kao i klase `Stack` i `Queue`, i ova klasa dopušta da dodajemo samo objekte a ne primitivne tipove. Na objektima klase T koje dodajemo u red mora biti definisana operacija poređenja, pomoću funkcije `compareTo` (pogledati klasu `Test` u primjeru 2) tj. klasa T mora implementirati interfejs (interface) `Comparable<T>`. Funkcija `compareTo` treba da vrati negativan broj, nulu ili pozitivan broj (obično su to -1, 0, 1) u zavisnosti da li je prvi objekat manji, jednak ili veći od drugog. U prvom primjeru, u red dodajemo stringove, a na klasi `String` je već definisan poredak.

Primjer 1:

```
import java.util.*;

public class PriorityQueueDemo {

      PriorityQueue<String> stringQueue;


      public static void main(String[] args){

      stringQueue = new PriorityQueue<String>();

      stringQueue.add("ab");
      stringQueue.add("abcd");
      stringQueue.add("abc");
      stringQueue.add("a");

      while(stringQueue.size() > 0)
             System.out.println(stringQueue.remove());

      }

}
```

Primjer 2:

```
class Test implements Comparable<Test> {
   private int priority; // prioriret
   public int x,y; // vrijednosti

   // konstruktori
   public Test(int priority) {
      this.priority = priority;
      x = 0;
      y = 0;
   }

   public Test(int priority, int xx, int yy) {
            this.priority = priority;
            x = xx;
            y = yy;
   }

   public int compareTo(Test o) {
      if (this.priority < o.priority)
         return -1;
      else if (this.priority > o.priority)
         return 1;
      return 0;
   }

   public int getPriority() {
      return this.priority;
   }
}

import java.util.*;
public class PriorityQueueExample {

   public static void main(String args[]) {

      Queue<Test> queue = new PriorityQueue<Test>(); // kreiramo novi red
      Test t;
      // Dodamo tri objekta klase Test u red
      queue.offer(new Test(3));
      queue.offer(new Test(1));
      queue.offer(new Test(2));

      // dodamo 10 slucajnih objekata klase Test
      Random r = new Random();
      for(int i=0; i<10; i++)
      {
        queue.offer(new Test(1+r.nextInt(20),1+r.nextInt(20),-10+r.nextInt(30)));
      }

      // ispraznimo red, elementi ce biti poredjani po neopadajucim prioritetima
      while (queue.size()!= 0) {
            t = queue.poll();
            System.out.printf("%d %d %d\n", t.getPriority(), t.x, t.y);
      }
   }
}
```