

 **Algebra**
visoka škola za
primijenjeno računarstvo




STRUKTURE PODATAKA I ALGORITMI

Predavanje 03



LISTA, STOG I RED

Strana • 2

 **Algebra**
visoka škola za
primijenjeno računarstvo

Apstraktni tip podataka

▪ Tip podataka

- Uređen par (S, O) , gdje je S skup elemenata (vrijednosti), a O skup operacija nad tim elementima (vrijednostima)
- Primjerice, tip podataka `int` ima vrijednosti koje može poprimiti i operacije $+$, $-$, $*$, $/$, $\%$, ...

▪ Apstraktni tip podataka (engl. ADT, *abstract data type*)

- Bilo koji tip koji ne specificira implementaciju se smatra **apstraktnim**
- Služi za opisivanje koncepata
- Može biti implementiran na razne načine

Strana • 3



Implementacija apstraktnog tipa podataka

▪ Implementacija apstraktnog tipa podataka

- Konkretna realizacija dotičnog apstraktnog tipa podataka u nekom programu
- Sastoji se od:
 - Definicije za strukturu podataka (kojom se prikazuju podaci iz apstraktnog tipa podataka)
 - Niza metoda (kojima se operacije iz apstraktnog tipa podataka ostvaruju pomoću odabranih algoritama)
- Za isti apstraktni tip podataka obično se može smisliti više različitih implementacija
 - Razlikuju se po tome što koriste različite strukture za prikaz podataka te različite algoritme za izvršavanje operacija

Strana • 4



Lista, stog i red

- Osnovni apstraktni tipovi podataka koje ćemo proučavati u ovom ishodu su:
 - Lista
 - Stog
 - Red
- Naučit ćemo njihov apstraktni opis te napraviti nekoliko implementacija
 - Sve implementacije će težiti čitljivosti, a ne performansama
 - Sve implementacije će imati nedostataka

LISTA

Uvod

- **Lista** je konačni niz (od nula ili više) podataka **istog** tipa
- Podaci koji čine listu nazivaju se njeni **elementi**
- Listu obično bilježimo ovako:

$$a_1, a_2, \dots, a_{i-1}, a_i, a_{i+1}, \dots, a_n$$

- **n** je **duljina** liste
 - Uvijek smatramo da je $n \geq 0$
 - Ako je $n = 0$, kažemo da je lista prazna

Lista

- Pazimo na redne brojeve da bismo izbjegli zabunu:
 - a_1 je **prvi** element liste
 - a_2 je **drugi** element liste
 - a_i je **i-ti** element liste
 - a_n je **n-ti** ili **zadnji** element liste
- Zbog linearnog svojstva liste možemo reći da a_i dolazi ispred a_{i+1} i iza a_{i-1}
- Moguće je da neki od elemenata liste imaju jednaku vrijednost
 - Identitet elementa određen je njegovom **pozicijom**, a ne njegovom **vrijednošću**

Lista kao apstraktni tip podataka (1/3)

- Da bi lista bila ATP, potrebno je definirati:
 - Strukturu podataka
 - Operacije
- Definicija strukture podataka:

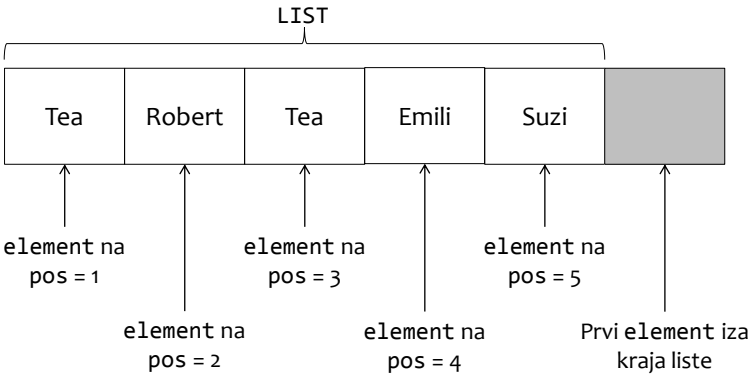
ELTYPE	Tip elemenata liste (može biti bilo koji tip podataka)
LIST	Lista (konačan niz elemenata tipa ELTYPE)
element	Element liste (podatak tipa ELTYPE)
pos	Pozicija elementa u listi

Strana • 9



Lista kao apstraktni tip podataka (2/3)

- Grafički prikaz primjera liste stringova od 5 elemenata:
 - ELTYPE je string



Strana • 10



Lista kao apstraktni tip podataka (3/3)

▪ Moguće operacije:

END()	Vraća poziciju prvog elementa iza kraja liste
FIRST()	Vraća poziciju prvog elementa u listi, a ako je lista prazna vraća END()
INSERT(element, pos)	Ubacuje element na pos pomičući preostale elemente iza pos za jedno mjesto dalje. Ako pos nije ispravan, javlja grešku.
READ(pos)	Vraća element liste na poziciji pos. Javlja grešku ako pos ne postoji ili ako je pos = END().
REMOVE(pos)	Briše element s pozicije pos pomičući preostale elemente iza pos za jedno mjesto unatrag. Javlja grešku ako pos ne postoji ili ako je pos = END().
FIND(element)	Vraća poziciju u listi koja ima vrijednost element. Ako ga nema, vraća poziciju END(). Ako ih postoji više, vraća poziciju prvoga pojavljivanja.
EMPTY()	Uklanja sve elemente iz liste i vraća END()
NEXT(pos)	Vraća prvu poziciju iza pos, uz pretpostavku da je pos ispravna pozicija i da nije jednaka END()
PREV(pos)	Vraća prvu poziciju ispred pos, uz pretpostavku da je pos ispravna pozicija i da nije jednaka FIRST().

Strana • 11



Primjer korištenja lista

```
lista osobe;
osobe.insert("Suzi", 1);
osobe.insert("Emili", 1);
osobe.insert("Robert", 1);
osobe.insert("Tea", 1);
osobe.insert("Tea", 3);

cout << "ISPIS LISTE: " << endl;
ELTYPE element;
for (POSITION pos = osobe.first(); pos < osobe.end(); pos =
                                osobe.next(pos)) {
    osobe.read(pos, element);
    cout << element << endl;
}

cout << endl << "TRAZENJE OSOBE: " << endl;
POSITION found = osobe.find("Emili");
if (found != osobe.end()) {
    cout << "Pronasao na mjestu " << found << endl;
}
}
```

Strana • 12



U nastavku

- U nastavku ćemo:
 - Implementirati listu pomoću polja
 - Lista stringova
 - Lista cijelih brojeva
 - Generička lista
 - Implementirati listu dinamičkom dodjelom memorije
 - Jednostruko povezana lista
 - Dvostruko povezana lista
- Programer koji koristi listu ne smije znati ništa o implementaciji
 - Možemo promijeniti implementaciju bez njegovog znanja

Strana • 13



Pogreške

- Ponekad moramo iz metode javiti pozivatelju da se dogodila pogreška
- Postoje dva načina za to napraviti:
 - Tradicionalni, vraćanjem `true` za uspjeh ili `false` za neuspjeh
 - Moderni, korištenjem `try/catch` bloka
- Na SPA ćemo koristiti tradicionalni način, a moderni način ćemo koristiti na OOP

Strana • 14



Programerske uloge

- Za ovu raspravu nam trebaju dva pojma:
 - Programer liste = onaj programer koji programira listu
 - Programer korisnik liste = onaj programer koji koristi listu
 - Mogu biti ista osoba ☺
- Što programer korisnik liste vidi?
 - Vidi samo ono što je javno, a to su apstraktni elementi liste!
- Vidi li programer korisnik liste implementaciju, tj. zna li da je lista implementirana poljem?
 - Ne!
 - Programer liste u bilo kojem trenutku može promijeniti implementaciju – to je snaga apstraktnih tipova podataka

Strana • 15

IMPLEMENTACIJA LISTE POMOĆU POLJA

Strana • 16

Uvod

▪ Ideja:

- Jedna klasa će predstavljati tip podataka za listu
- Elemente liste ćemo čuvati u polju koje će biti član klase
- Pozicija elementa u listi će biti jednaka indeksu elementa u polju + 1
 - Polje počinje s indeksom 0, a lista s elementom 1
- Brojač će sadržavati poziciju zadnjeg elementa u listi
 - Kako bismo znali koji dio polja je popunjen
 - Kapacitet liste je jednak veličini polja
- U polju ne smije biti "rupa" – svi elementi polja od početka do kraja liste moraju biti popunjeni

Strana • 17



Posljedice odabira implementacije

▪ Prednosti:

- Pristup i -tom elementu liste je brz
 - Pristup elementima polja je brz jer su poslagani jedan iza drugoga u memoriji

▪ Nedostaci:

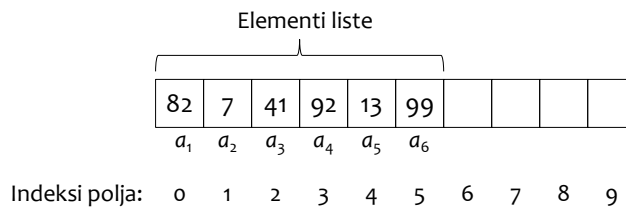
- Prilikom kreiranja liste moramo reći koliko najviše elemenata lista može sadržavati (kapacitet liste)
 - Ako lista sadržava manje elemenata, svejedno troši memoriju
 - Ako treba staviti više elemenata u listu, ne možemo bez promjene veličine polja (skupo)
- Operacije umetanja i brisanja su složene – elemente iza umetnog/obrisanog treba pomicati naprijed/nazad

Strana • 18



Grafički prikaz

- Slika prikazuje primjer liste od 6 elemenata (cijeli brojevi) koja je implementirana poljem od 10 elemenata (kapacitet)



- **Paziti:** Redni brojevi elemenata u listi nisu jednaki indeksima u polju
 - Prvi element liste ima broj 1, ali se nalazi na polju na indeksu 0!

Strana • 19



Implementacija liste stringova

- Moguća implementacija liste stringova poljem veličine 10

```
class lista {
private:
    static const uint CAPACITY = 10;
    string _elements[CAPACITY];
    uint _last; // Pozicija zadnjeg elementa liste (odmah
                // ispred mjesta "prvo iza").

public:
    lista();
    uint end();
    uint first();
    bool insert(string element, uint pos);
    bool read(uint pos, string& element);
    bool remove(uint pos);
    uint find(string element);
    uint empty();
    uint next(uint pos);
    uint prev(uint pos);
};
```

Strana • 20



Objašnjenje privatnih članova

- Privatni članovi služe za čuvanje podataka i podršku operacijama nad njima
 - Kažemo da čuvaju **stanje**
 - Polje `_elements` je veličine 10 i sadržavat će elemente liste
 - `_last` čuva **poziciju** na kojoj se nalazi zadnji element liste, tj. onaj element koji se nalazi odmah ispred "prvog iza"
 - Inicijalno mora biti 0 jer lista ne postoji ("prvi iza" je na poziciji 1)
 - Paziti:
 - Pozicija elemenata liste ide od 1
 - Za rad s poljem **treba koristiti indekse elemenata koji idu od 0**
 - **Uvijek biti svjestan radi li se o poziciji u listi ili o indeksu u polju**

Strana • 21



Konstruktor i typedef

- Konstruktor je metoda jednakog naziva kao i klasa
 - Poziva se prilikom izrade svakog objekta
 - Najčešće služi za inicijalizaciju privatnih varijabli
 - Kod nas će samo inicijalizirati `_last` na 0
- typedef služi za definiranje aliasa za postojeće tipove podataka
 - Sintaksa: `typedef postojeći_naziv novi_naziv;`
 - Primjer: `typedef unsigned int uint;`
 - Sad nam je svejedno hoćemo li pisati `uint` ili `unsigned int`

Strana • 22



DEMO

- Implementirajmo operacije
 - Rješenje: "ListaPoljem"

Strana • 23

Analiza rješenja

- Nedostaci rješenja:
 1. Korisnik liste "zna" da je pozicija po tipu `unsigned int`, a to ne bi trebao znati
 2. U listi možemo čuvati samo stringove

Strana • 24

Rješenje problema 1

▪ Moguće rješenje problema 1:

```
class lista {
private:
    static const uint CAPACITY = 10;
    string _elements[CAPACITY];
    POSITION _last; // Pozicija zadnjeg elementa liste (odmah
                    // ispred mjesta "prvo iza").

public:
    lista();
    POSITION end();
    POSITION first();
    bool insert(string element, POSITION pos);
    bool read(POSITION pos, string& element);
    bool remove(POSITION pos);
    POSITION find(string element);
    POSITION empty();
    POSITION next(POSITION pos);
    POSITION prev(POSITION pos);
};
```

Strana • 25

DEMO

- Riješimo problem 1: korisnik liste "zna" da je pozicija po tipu `unsigned int`, a to ne bi trebao znati
 - Rješenje "ListaPoljem_v2"

Strana • 26

Lista cijelih brojeva

- Kako bi izgledala implementacija liste cijelih brojeva?

```
class lista {
private:
    static const uint CAPACITY = 10;
    int _elements[CAPACITY];
    POSITION _last; // Pozicija zadnjeg elementa liste (odmah
                    // ispred mjesta "prvo iza").

public:
    lista();
    POSITION end();
    POSITION first();
    bool insert(int element, POSITION pos);
    bool read(POSITION pos, int& element);
    bool remove(POSITION pos);
    POSITION find(int element);
    POSITION empty();
    POSITION next(POSITION pos);
    POSITION prev(POSITION pos);
};
```

Strana • 27



Rješenje problema 2

- Moguće rješenje problema 2 (bolje bi bilo genericima):

```
class lista {
private:
    static const uint CAPACITY = 10;
    ELTYPE _elements[CAPACITY];
    POSITION _last; // Pozicija zadnjeg elementa liste (odmah
                    // ispred mjesta "prvo iza").

public:
    lista();
    POSITION end();
    POSITION first();
    bool insert(ELTYPE element, POSITION pos);
    bool read(POSITION pos, ELTYPE& element);
    bool remove(POSITION pos);
    POSITION find(ELTYPE element);
    POSITION empty();
    POSITION next(POSITION pos);
    POSITION prev(POSITION pos);
};
```

Strana • 28



DEMO

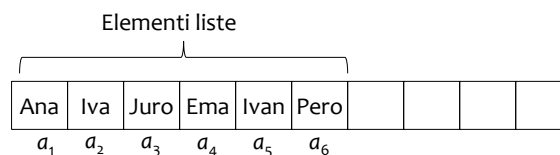
- Riješimo problem 2: U listi možemo čuvati samo stringove
 - Rješenje "ListaPoljem_v3"
 - Nije do kraja dobro rješenje – lista i dalje zahtijeva definiranje tipa podataka kojeg će čuvati prilikom kompajliranja
 - Na OOP ćemo naučiti koristiti generike

Strana • 29



Implementacija operacija

- U nastavku slijedi opis implementacija operacija koje smo implementirali za našu listu
- Kao primjer ćemo koristiti sljedeću listu stringova implementiranu poljem od 10 elemenata:



Strana • 30



Konstruktor, početak i kraj

- Implementacija konstruktora:

```
lista::lista() {
    _last = 0;
}
```

- Implementacija metode first():

```
POSITION lista::first() {
    return 1;
}
```

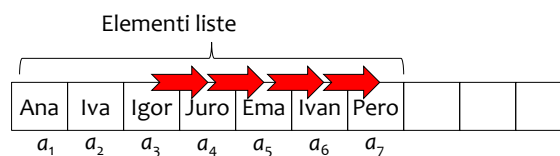
- Implementacija metode end():

```
POSITION lista::end() {
    return _last + 1;
}
```

Strana • 31

Dodavanje elemenata u listu (1/2)

- Želimo dodati Igora na 3 mjesto u listi
- Sve ostale elemente moramo pomaknuti za jedno mjesto prema kraju liste



- Problem: puno kopiranja!

Strana • 32

Dodavanje elemenata u listu (2/2)

- Implementacija metode `insert()`:

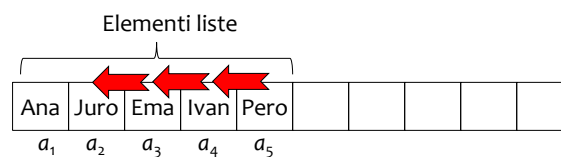
```
bool lista::insert(ELTYPE element, POSITION pos) {
    if (pos >= first() && pos <= end() && _last < CAPACITY){
        // Pomakni u desno.
        for (POSITION i = _last; i >= pos; i--) {
            _elements[i] = _elements[i - 1];
        }

        // Upiši novi i ažuriraj _last.
        _elements[pos - 1] = element;
        _last++;
        return true;
    }
    return false;
}
```

Strana • 33

Brisanje iz liste (1/2)

- Želimo obrisati Ivu iz liste
- Sve elemente iza Ive moramo pomaknuti za jedno mjesto prema početku liste
- Izgled liste nakon brisanja:



- Problem: puno kopiranja!

Strana • 34

Brisanje iz liste (2/2)

- Implementacija metode `remove()`:

```
bool lista::remove(POSITION pos) {
    if (pos >= first() && pos < end()) {
        // Pomakni u lijevo.
        for (POSITION i = pos - 1; i < _last - 1; i++){
            _elements[i] = _elements[i + 1];
        }

        // Ažuriraj _last.
        _last--;
        return true;
    }
    return false;
}
```

Strana • 35



Implementacije ostalih metoda (1/3)

- Implementacija metode `read()`:

```
bool lista::read(POSITION pos, ELTYPE& element) {
    if (pos >= first() && pos < end()) {
        element = _elements[pos - 1];
        return true;
    }
    return false;
}
```

- Implementacija metode `empty()`:

```
POSITION lista::empty() {
    _last = 0;
    return end();
}
```

Strana • 36



Implementacije ostalih metoda (2/3)

- Implementacija metode `next()`:

```
POSITION lista::next(POSITION pos) {
    return pos + 1;
}
```

- Implementacija metode `previous()`:

```
POSITION lista::prev(POSITION pos) {
    return pos - 1;
}
```

Strana • 37



Implementacije ostalih metoda (3/3)

- Implementacija metode `find()`:

```
POSITION lista::find(ELTYPE element) {
    for (POSITION i = first(); i < end(); i++) {
        if (_elements[i - 1] == element) {
            return i;
        }
    }
    return end();
}
```

Strana • 38



Složenost operacija

- Složenost većine operacija nad listom realiziranom poljem je jednostavna
 - Operacije FIND, REMOVE i INSERT imaju složenost $O(n)$
 - Ostale operacije imaju složenost $O(1)$

Nedostaci implementacije poljem

- Implementacija poljem je jednostavna, ali ima određene nedostatke:
 - Ograničeni smo s veličinom polja koju moramo unaprijed zadati (statičko polje)
 - Uvijek je u memoriji rezerviran prostor za maksimalnu duljinu liste, bez obzira bila lista puna, prazna ili poluprazna
 - Ubacivanja i brisanja traju dugo zbog potrebe za pomicanjem preostalih elemenata

Zadaci za vježbu

1. Proučiti implementacije s predavanja.