

Objektno orijentisano programiranje

Generičke klase

Generičke klase

- U programskom jeziku C++ postoji mehanizam pomoću kog možemo napisati šablon (eng. *template*) kojim opisujemo opšti slučaj, bez upotrebe konkretnih tipova.
- Klasa koja je napisana pomoću šablona naziva se **generička klasa**.
- Kada se šablonu navedu konkretni tipovi dobijamo konkretne klase.

Primer 1

Napisati generičku klasu **Niz** koja modeluje niz dužine **D** i tipa **T**.

(videti tekst zadatka 1 u folderu “Tekstovi”)

Rešenje: 1.1. Generička klasa **Niz** - templatejt

- Kao što je rečeno u tekstu zadatka, generička klasa **Niz** treba biti modelovana tako da opisuje opšti slučaj niza koji sadrži maksimalno **D** elemenata tipa **T**.
- Iznad naziva generičke klase potrebno je definisati njen šablon. To se postiže korišćenjem ključne reči **template** iza koje se definišu elementi šablona.

```
#include <iostream>
using namespace std;

template <class T, int D>
class Niz {};
```

Rešenje: 1.2. Generička klasa **Niz** - polja

- Na osnovu teksta zadatka, generička klasa **Niz** treba da sadrži niz **el** koji sadrži maksimalno **D** elemenata tipa **T**, kao i polje **brEl** koje predstavlja trenutni broj elemenata u nizu.

```
#include <iostream>
using namespace std;

template <class T, int D>
class Niz {
    private:
        T el[D];
        int brEl;
};
```

Napomena: Tip **T** može biti
prost tip (int, double, char, ...)
ili klasi tip (DinString, ...).

Rešenje: 1.3. Generička klasa **Niz** - konstruktor/destruktor

- Potrebno je kreirati konstruktor bez parametara i destruktor.
- Preporuka je da se odmah nakon kreiranja konstruktora i ostalih metoda u fajlu *niz.hpp* izvrši njihovo testiranje u fajlu *main.cpp* (kreiranjem objekta ili pozivom metode u *main* funkciji).
- Na taj način smanjujemo mogućnost od gomilanja grešaka.

Rešenje: 1.3. Generička klasa **Niz** - konstruktor/destruktor

- *niz.hpp*:

```
public:  
    Niz () { brEl=0; }  
    ~Niz () {}
```

- *main.cpp* - kreiranje objekta, konkretizacije generičke klase niz:

```
int main()  
{  
    cout << "Niz celih brojeva duzine 10:" << endl;  
    Niz<int,10> iNiz1, iNiz2;  
  
    return 0;  
}
```

Rešenje: 1.4. Generička klasa **Niz** - metode

- Potrebno je napisati metodu **getBrEl** koja vraća trenutni broj elemenata u nizu, metodu **insertNiz** za dodavanje elementa u niz kao i metodu **printNiz** za ispis elemenata niza.

```
int getBrEl() const { return brEl; }
```

```
void printNiz() const;
```

```
bool insertNiz(const T&);
```


Rešenje: 1.4. Generička klasa **Niz** - metode

- Iznad definicije svake od metoda u okviru kojih se manipuliše elementima šablona, potrebno je definisati šablon generičke klase.
- To se postiže korišćenjem ključne reči **template** iz koje se definišu elementi šablona.

```
template <class T, int D>  
bool Niz<T,D>::insertNiz(const T& t) {
```

- Napomena: Gore napisano važi ukoliko se deklaracija i definicija metode razdvajaju. Ukoliko to nije slučaj, odnosno ukoliko se metoda definiše unutar tela generičke klase, nije potrebno definisati šablon.

Rešenje: 1.4. Generička klasa **Niz** - metode

- Metoda za dodavanje elementa u niz - **insertNiz**
- *niz.hpp*:

```
template <class T, int D>
bool Niz<T,D>::insertNiz(const T& t) {
    if(brEl < D) {
        el[brEl] = t;
        brEl++;
        return true;
    }
    else
        return false;
}
```

Rešenje: 1.4. Generička klasa **Niz** - metode

- Poziv metode za dodavanje elementa u niz - **insertNiz**
- *main.cpp*:

```
cout << "Niz celih brojeva duzine 10:" << endl;  
Niz<int, 10> iNiz1, iNiz2;
```

```
iNiz1.insertNiz(1);  
iNiz1.insertNiz(2);  
iNiz1.insertNiz(3);  
iNiz1.insertNiz(4);  
iNiz1.insertNiz(5);  
iNiz1.insertNiz(6);
```

Rešenje: 1.4. Generička klasa **Niz** - metode

- Metoda za ispis elementa niza - **printNiz**
- *niz.hpp*:

```
template <class T, int D>
void Niz<T,D>::printNiz() const {
    cout << "( ";
    for(int i = 0; i < brEl - 1; i++)
        cout << el[i] << ", ";
    cout << el[brEl - 1] << " )" << endl;
}
```

Rešenje: 1.4. Generička klasa **Niz** - metode

- Poziv metode za ispis elementa niza - **printNiz**
- *main.cpp*:

```
cout << "Niz celih brojeva duzine 10:" << endl;  
Niz<int,10> iNiz1, iNiz2;
```

```
iNiz1.insertNiz(1);  
iNiz1.insertNiz(2);  
iNiz1.insertNiz(3);  
iNiz1.insertNiz(4);  
iNiz1.insertNiz(5);  
iNiz1.insertNiz(6);  
iNiz1.printNiz();
```

Podsećanje - preklapanje operatora

- Preklapanje operatora je mehanizam koji omogućava da se za većinu standardnih operatora definiše njihovo ponašanje za slučaj da su operandi klasnih tipova.
- Operatori se mogu preklopiti na dva načina: operatorskom metodom ili operatorskom funkcijom.

Napomena: Više o ograničenjima koja postoje prilikom definisanja novih ponašanja operatora naučićete na predavanjima.

Rešenje: 1.4. Generička klasa **Niz** - operatorske metode

- Potrebno je preklopiti operator `=` za dodelu vrednosti i operatore `[]` za indeksiranje elemenata. Preklapanje je potrebno implementirati operatorskim metodama.

```
Niz<T,D>& operator=(const Niz<T,D>&);
```

```
T operator[](int i) const { return el[i]; }
```

```
T& operator[](int i) { return el[i]; }
```

Rešenje: 1.4. Generička klasa **Niz** - operatorske metode

- Preklopljen operator = za dodelu vrednosti
- *niz.hpp*:

```
template <class T, int D>
Niz<T,D>& Niz<T,D>::operator=(const Niz<T,D>& rn) {
    for(brEl = 0; brEl < rn.brEl; brEl++)
        el[brEl] = rn[brEl];
    return *this;
}
```


Rešenje: 1.4. Generička klasa **Niz** - operatorske metode

- Poziv preklopljenog operatora = za dodelu vrednosti
- *main.cpp*:

```
iNiz2 = iNiz1;
```

Rešenje: 1.5. Generička klasa **Niz** - operatorske funkcije

- Potrebno je preklopiti operator `==` za proveru jednakosti. Koristiti operatorsku funkciju.
- *niz.hpp* izvan tela klase **Niz**:

```
template <class T, int D>
bool operator==(const Niz<T,D>& rn1, const Niz<T,D>& rn2) {
    if(rn1.getBrEl() != rn2.getBrEl())
        return false;
    for(int i = 0; i < rn1.getBrEl(); i++)
        if(rn1[i] != rn2[i])
            return false;
    return true;
}
```

Rešenje: 1.5. Generička klasa **Niz** - operatorske funkcije

- Poziv preklopljenog operatora `==` za proveru jednakosti
- *main.cpp*:

```
iNiz2 = iNiz1;  
cout << endl;  
if(iNiz1 == iNiz2)  
    cout << "iNiz1 i iNiz2 su jednaki" << endl;  
else  
    cout << "iNiz1 i iNiz2 nisu jednaki" << endl;
```

Rešenje: 1.5. Generička klasa **Niz** - operatorske funkcije

- Potrebno je preklopiti operator **!=** za proveru nejednakosti. Koristiti operatorsku funkciju.
- *niz.hpp* izvan tela klase **Niz**:

```
template <class T, int D>
bool operator!=(const Niz<T,D>& rn1, const Niz<T,D>& rn2) {
    if(rn1.getBrEl() != rn2.getBrEl())
        return true;
    for(int i = 0; i < rn1.getBrEl(); i++)
        if(rn1[i] != rn2[i])
            return true;
    return false;
}
```

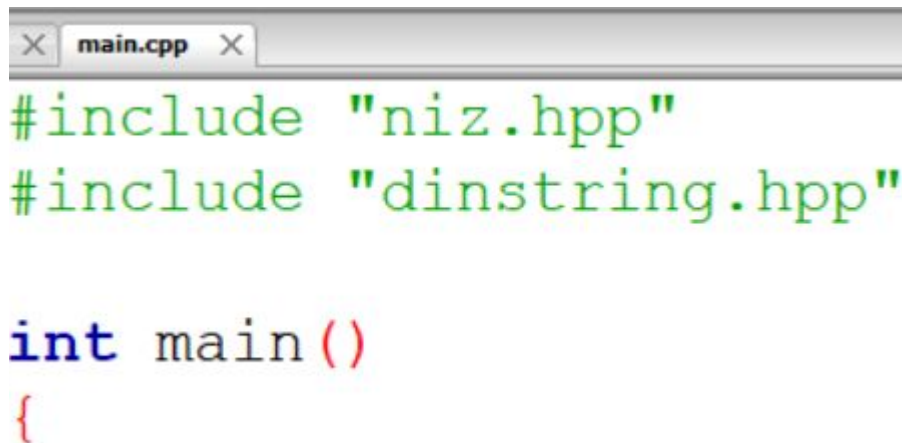
Rešenje: 1.5. Generička klasa **Niz** - operatorske funkcije

- Poziv preklopljenog operatora **!=** za proveru nejednakosti
- *main.cpp*:

```
iNiz2 = iNiz1;  
iNiz2[0] = 10;  
cout << endl;  
if(iNiz1 != iNiz2)  
    cout << "iNiz1 i iNiz2 nisu jednaki" << endl;  
else  
    cout << "iNiz1 i iNiz2 su jednaki" << endl;
```

Rešenje: 1.6. Generička klasa **Niz**

- Budući da se u zadatku traži testiranje koje uključuje objekte klase **DinString**, potrebno je u fajlu *main.cpp* uključiti *dinstring.hpp* fajl.



```
main.cpp
#include "niz.hpp"
#include "dinstring.hpp"

int main()
{
```

Rešenje: 1.6. Generička klasa **Niz**

- Testiranje sa objektima klase **DinString** kao elementima niza
- *main.cpp*:

```
cout << endl;  
cout << "Niz stringova duzine 20:" << endl;  
Niz<DinString,20> dsNiz1, dsNiz2;
```

```
dsNiz1.insertNiz("Jedan");  
dsNiz1.insertNiz("Dva");  
dsNiz1.insertNiz("Tri");  
dsNiz1.insertNiz("Cetiri");  
dsNiz1.insertNiz("Pet");  
dsNiz1.insertNiz("Sest");  
dsNiz1.printNiz();
```

Rešenje: 1.6. Generička klasa **Niz**

- Testiranje sa objektima klase **DinString** kao elementima niza
- *main.cpp*:

```
dsNiz2=dsNiz1;  
cout << endl;  
if(dsNiz1==dsNiz2)  
    cout<<"dsNiz1 i dsNiz2 su jednaki"<<endl;  
else  
    cout<<"dsNiz1 i dsNiz2 nisu jednaki"<<endl;  
  
dsNiz2=dsNiz1;  
dsNiz2[0]="Deset";  
cout << endl;  
if(dsNiz1!=dsNiz2)  
    cout<<"dsNiz1 i dsNiz2 nisu jednaki"<<endl;  
else  
    cout<<"dsNiz1 i dsNiz2 su jednaki"<<endl;
```


Primer 2

Napisati generičku klasu **Trezor<class SADRZAJ, int KAPACITET>** koja opisuje trezor sa sefovima.

U svakom sefu trezora je moguće držati po jedan predmet tipa **SADRZAJ**, a ukupan broj sefova odgovara vrednosti **KAPACITET**. Prilikom kreiranja trezora svi sefovi u trezoru su prazni.

(videti tekst zadatka 1-1 u folderu “Tekstovi”)

Rešenje: 2.1. Generička klasa **Trezor** - templejt

- Iznad naziva generičke klase potrebno je definisati njen šablon. To se postiže korišćenjem ključne reči **template** iza koje se definišu elementi šablona.
- *trezor.hpp*:

```
template <class SADRZAJ, int KAPACITET>  
class Trezor {};
```

Rešenje: 2.2. Generička klasa **Trezor** - polja

- Na osnovu teksta zadatka, generička klasa **Trezor** treba da sadrži polje/objekat član niz **sefovi** koji sadrži maksimalno **KAPACITET** elemenata tipa **SADRZAJ**, kao i polje niz **popunjenost** koji predstavlja informaciju o tome koji su sefovi popunjeni, a koji ne.

```
template <class SADRZAJ, int KAPACITET>
class Trezor {
    private:
        SADRZAJ sefovi[KAPACITET]; /// ubacuju se predmeti
        bool popunjenost[KAPACITET]; /// pomocni niz za pracenje popunjenosti sefa
};
```

Rešenje: 2.3. Generička klasa **Trezor** - konstruktor

- Potrebno je kreirati konstruktor bez parametara.
- Preporuka je da se odmah nakon kreiranja konstruktora i ostalih metoda u fajlu *trezor.hpp* izvrši njihovo testiranje u fajlu *main.cpp* (kreiranjem objekta ili pozivom metode u *main* funkciji).
- Na taj način smanjujemo mogućnost od gomilanja grešaka.

Rešenje: 2.3. Generička klasa **Trezor** - konstruktor

- *trezor.hpp*:

```
public:
    Trezor() {
        for(int i = 0; i < KAPACITET; i++) {
            popunjenost[i] = false;
        }
    }
```

- Instanciranje objekta generičke klase **Trezor** u fajlu *main.cpp* biće testirano nakon što implementiramo klasu **Dijamant**.

Rešenje: Klasa **Dijamant**

Napisati klasu **Dijamant** koja opisuje dijamant sa sledećim karakteristikama: vrednost (double) i broj karata (double).

(videti tekst zadatka 1-1 u folderu "Tekstovi")

Rešenje: 2.4. Klasa **Dijamant** - polja

- Na osnovu teksta zadatka, klasa **Dijamant** treba da sadrži polje **vrednost**, kao i polje **karat** koje označava broj karata.



```
#include <iostream>
using namespace std;

class Dijamant {
    private:
        double vrednost;
        double karat;
};
```

Rešenje: 2.5. Klasa **Dijamant** - konstruktori

- Potrebno je kreirati konstruktor bez parametara, konstruktor sa parametrima i konstruktor kopije.
- *dijamant.hpp*:

public:

```
Dijamant() : vrednost(10000), karat(1) {}
```

```
Dijamant(double v, double k) : vrednost(v), karat(k) {}
```

```
Dijamant(const Dijamant& d) : vrednost(d.vrednost) , karat(d.karat) {}
```

- *main.cpp*:

```
Dijamant d1, d2(100000, 50), d3(d2);
```


Rešenje: 2.6. Klasa **Dijamant** - operator ispisa

- Potrebno je preklopiti operator << za ispis prijateljskom funkcijom.
- *dijamant.hpp*:

```
friend ostream& operator<<(ostream& out, const Dijamant &d) {  
    out << "Dijamant: vrednost = " << d.vrednost << " karat = " << d.karat;  
    return out;  
}
```

- *main.cpp*:

```
cout << d1 << endl;  
cout << d2 << endl;  
cout << d3 << endl;
```

Rešenje: 2.7. Generička klasa **Trezor** - testiranje konstruktora

- Nakon što je klasu **Dijamant** kreirana, može se nastaviti sa implementacijom i testiranjem metoda generičke klase **Trezor**.
- *main.cpp*:

```
Trezor<Dijamant, 100> trezor;
```

Rešenje: 2.8. Generička klasa **Trezor** - metode

- U generičkoj klasi **Trezor**, pored konstuktora, potrebno je napisati metodu **getBrojPopunjenihSefova** koja vraća broj popunjenih sefova, metodu **ubaciSadrzaj** koja prosleđeni predmet smešta u prvi prazan sef i metodu **izbaciSadrzaj** koja uklanja iz sefa predmet sa zadatim rednim brojem.

Rešenje: 2.8. Generička klasa **Trezor** - metode

- Metodu koja vraća broj popunjenih sefova - **getBrojPopunjenihSefova**.
- *trezor.hpp*:

```
int getBrojPopunjenihSefova() {  
    int n = 0;  
  
    for(int i = 0; i < KAPACITET; i++) {  
        if(popunjenost[i]) {  
            n++;  
        }  
    }  
    return n;  
}
```

- *main.cpp*:

```
cout << "Trezor: broj popunjenih = " << trezor.getBrojPopunjenihSefova() << endl;
```

Rešenje: 2.8. Generička klasa **Trezor** - metode

- Metodu koja uklanja iz sefa predmet sa zadatim rednim brojem - **ubaciSadrzaj**.

- ```
int ubaciSadrzaj(const SADRZAJ& predmet) {
 for(int i = 0; i < KAPACITET; i++) {
 if(!popunjenost[i]) {
 popunjenost[i] = true;
 sefovi[i] = predmet;
 return i;
 }
 }

 return -1;
}
```

- *main.cpp*:

```
cout << "Trezor: ubaci D1 rezultat = " << trezor.ubaciSadrzaj(d1) << endl;
```

## Rešenje: 2.8. Generička klasa **Trezor** - metode

- Metodu koja prosleđeni predmet smešta u prvi prazan sef - **izbaciSadrzaj**.
- *trezor.hpp*:

```
bool izbaciSadrzaj(int sef) {
 bool ret = false;

 if(popunjenost[sef]) {
 popunjenost[sef] = false;
 ret = true;
 }
 return ret;
}
```

- *main.cpp*:

```
cout << "Trezor: izbaci 0 rezultat = " << trezor.izbaciSadrzaj(0) << endl;
```

# Klasa List

- Klasa List se koristi za čuvanje podataka strukturiranih u povezanoj listi
- Čuvanje različitih tipova podataka je omogućeno činjenicom da je klasa List generička, ovo je postignuto kao i do sad korišćenjem ključne reči **template**
- Struktura listEl predstavlja jedan element unutar liste i čuva pored samog podatka (polje content) i pokazivač na sledeći element u listi (polje next)
- Klasa List ima kao polja pokazivače na prvi element i poslednji element (head i tail) kao i ukupan broj elemenata u listi (noEl)

```
template <class T>
class List{
 private:
 struct listEl {
 T content;
 struct listEl* next;
 };

 listEl *head;
 listEl *tail;
 int noEl;
```

# Klasa List

- Klasa List ima implementiran konstruktor bez parametara koji kreira praznu listu (postavlja pokazivače na NULL i broj elemenata na 0)
- Konstruktor kopije kopira već postojeću listu element po element i kreira novu strukturu sa istim vrednostima
- Operator = postojeću listu prazni, da bi zatim dodao sve elemente iz liste kojom postojeću listu izjednačavamo

```
public:
 List() {
 head = tail = NULL;
 noEl = 0;
 }

 List(const List<T>&);

 virtual ~List();

 List<T>& operator=(const List<T>&);
```



# Klasa List

- Metoda size vraća ukupan broj elemenata u listi

```
int size() const { return noEl; }
```

- Metoda empty vraća informaciju o tome da li je lista prazna ili ne (true ako je prazna, false ako nije)

```
bool empty() const { return head == NULL ? 1 : 0; }
```

- Metode clear briše sve elemente iz liste

```
void clear();
```

# Klasa list - metoda add

- Metoda add se koristi za dodavanje elementa u listu

```
bool add(int, const T&);
```

- Prvi parametar metode predstavlja poziciju na koju želimo da dodamo element, dok drugi predstavlja sam element koji se dodaje
- Povratna vrednost govori da li je operacija dodavanaj uspešno izvršena
- Tipični primeri neuspešno izvršene operacije su dodavanje elementa na pozicije su negative kao i pozicije koje su veće od ukupnog broja elemenata (možemo dodavati samo na pozicije sa vrednostima od 0 do list.size()+1)

# Klasa list - metoda add

- Primer kreiranja liste i dodavanja elementa

```
List<int> lista;
lista.add(1, 10);
```

- Primer dodavanja na kraj liste

```
lista.add(lista.size()+1, 20);
```

- Napomena: Indeksi u klasi List kreću od 1, ne od 0 kao kod nizova

# Klasa List - metoda remove

- Metoda remove koristi se za izbacivanje elemenata iz liste

```
bool remove(int);
```

- Jedini parametar u metodi predstavlja indeks elementa koji je potrebno izbaciti iz liste, dok povratna vrednost govori o tome da li je operacije uspešno izvršena
- Tipičan primer neuspešno izvršene operacije je pokušaj izbacivanja elementa sa negativnim indeksom ili inteksom većim od indeksa poslednjeg elementa
- Na sledećoj slici je primer izbacivanja elementa sa prve pozicija, a zatim i sa poslednje

```
lista.remove(1);
lista.remove(lista.size());
```

# Klasa List - metoda read

- Metoda read čita vrednost elementa sa odgovarajuće pozicije u listi  
`bool read(int, T&) const;`
- Bitno je приметiti da povratna vrednost metode nije pročitana vrednost već je tipa bool, tj. govori o tome da li je metoda uspešno izvršena ili ne
- Tipičan primer kada se metoda ne bi uspešno izvršila je prilikom prosleđivanja negativnih ili indeksa većih od indeksa poslednjeg elementa
- Prvi parametar određuje sa koje pozicije je potrebno pročitati vrednost, dok drugi parametar predstavlja referencu na objekat u koji želimo da sačuvamo pročitano vrednost

# Klasa List - metoda read

- Primer čitanja i ispisa vrednosti prvog elementa liste, primetimo da je neophodno napraviti pomoćnu promenljivu u koju ce se pročitana vrednot sačuvati

```
int pomocna;
lista.read(1, pomocna);
```

```
cout<<pomocna<<endl;
```

- Primer čitanaj i ispisa vrednosti poslednjeg elementa

```
int pomocna;
lista.read(lista.size(), pomocna);
```

```
cout<<pomocna<<endl;
```

# Literatura

1. Kupusinac A. : Zbirka rešenih zadataka iz programskog jezika C++. Novi Sad: FTN, 2011.