Stek

Apstraktni tip podatka stek se obično implementira na dva načina, i to:

1. pomoću nizova
2. pomoću povezane reprezentacije.

Za implementaciju steka pomoću povezane reprezentacije koriste se čvorovi koji su međusobno jednostruko povezani.

U nastavku je prikazan C++ kôd povezane reprezentacije reprezentacije steka.

StekPov.h

```
#ifndef __STEK_POV_H__
#define __STEK_POV_H__

#include "Cvor.h"

template <class Tip>
class StekPov
{
	Cvor<Tip>* prvi;
public:
	StekPov()
	{
		prvi = NULL;
	}
	void dodaj(Tip v)
	{
		Cvor<Tip>* n = new Cvor<Tip>(v, prvi);
		prvi = n;
	}

	Tip ukloni()
	{
		if (IsPrazan())
			throw exception("Greska. Nije moguce ukloniti elemenat iz
praznog steka.");
		Cvor<Tip>* t = prvi;
		prvi = prvi->next;
		Tip x = t->info;
		delete t;
		return x;
	}

	bool IsPrazan()
	{
		return (prvi == NULL);
	}

	void print()
	{
		Cvor<Tip>* t = prvi;
		while (t != NULL)
		{
			cout << "{" << t->info << "} ";
			t = t->next;
		}
	}
};

#endif
```

U nastavku je prikazan C++ kôd sekvencijalne reprezentacije steka.

StekSekv.h

```
#ifndef __STEK_SEKV_H__
#define __STEK_SEKV_H__

#include "Cvor.h"

template <class Tip>
class StekSekv
{
private:
	Tip* N;
	int brojac;
	int max;
	void prosiriStek()
	{
		//Studenti ne moraju poznavati implementaciju ove funkcije za ispit.
		//U tom slučaju je potrebno prikazati grešku da je stek pun.
		int newMax = max * 2;
		Tip* novi = new Tip[newMax * 2];
		for (int i = 0; i < max; i++)
		{
			novi[i] = N[i];
		}
		delete[] N;
		N = novi;
		max = newMax;
		cout << "\nSTEK: rekonstrukcija - nova velicina je " << newMax << endl;
	}
public:
	StekSekv(int max = 3)
	{
		this->max = max;
		N = new Tip[max];
		brojac = 0;
	}
	void dodaj(Tip v)
	{
		if (max == brojac)
		{
			prosiriStek(); //ili throw exception("Greska. Stek je pun.");
		}

		N[brojac] = v;
		brojac++;
	}

	Tip ukloni()
	{
		if (IsPrazan())
		{
			throw exception("Greska. Nije moguce ukloniti elemenat iz
praznog steka.");
		}

		brojac--;
		Tip t = N[brojac];
		return t;
	}

	bool IsPrazan()
	{
		return (brojac == 0);
	}
```

```
        void print()
        {
                cout << "\nSTEK: ";
                int i = 0;
                while (i < brojac)
                {
                        cout << "{" << N[i++] << "} ";
                }
        }
};

#endif
```

U nastavku je prikazan glavni program u kojem se demonstrira dodavanje i uklanjanje elemenata iz steka koji čuva elemente tipa **Osoba*.**

Main.cpp

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

#include "StekSekv.h"
#include "StekPov.h"

void testiranje_steka_pov(StekPov<Osoba*>* stek)
{
        cout << "=====TESTIRANJE STEKA ====" << endl;

        Osoba* p21 = new Osoba(21, "p21");
        Osoba* p22 = new Osoba(22, "p22");
        Osoba* p23 = new Osoba(23, "p23");
        Osoba* p24 = new Osoba(24, "p24");
        Osoba* p25 = new Osoba(25, "p25");

        stek->dodaj(p21);
        stek->print();

        stek->dodaj(p22);
        stek->print();

        Osoba* pOsoba1 = stek->ukloni();
        stek->print();
        cout << " uklonjeno sa vrha {" << *pOsoba1 << "} " << endl;

        stek->dodaj(p23);
        stek->print();

        stek->dodaj(p24);
        stek->print();

        stek->dodaj(p25);
        stek->print();

        while (!stek->IsPrazan())
        {
                Osoba* pOsobaW = stek->ukloni();
                stek->print();
                cout << " uklonjeno sa vrha {" << *pOsobaW << "} " << endl;
        }
}
```

```
void testiranje_steka_sekv(StekSekv<Osoba*>* stek)
{
	cout << "=====TESTIRANJE STEKA ====" << endl;

	Osoba* p21 = new Osoba(21, "p21");
	Osoba* p22 = new Osoba(22, "p22");
	Osoba* p23 = new Osoba(23, "p23");
	Osoba* p24 = new Osoba(24, "p24");
	Osoba* p25 = new Osoba(25, "p25");

	stek->dodaj(p21);
	stek->print();

	stek->dodaj(p22);
	stek->print();

	Osoba* pOsoba1 = stek->ukloni();
	stek->print();
	cout << " uklonjeno sa vrha {" << *pOsoba1 << "} " << endl;

	stek->dodaj(p23);
	stek->print();

	stek->dodaj(p24);
	stek->print();

	stek->dodaj(p25);
	stek->print();

	while (!stek->IsPrazan())
	{
		Osoba* pOsobaW = stek->ukloni();
		stek->print();
		cout << " uklonjeno sa vrha {" << *pOsobaW << "} " << endl;
	}
}


void main()
{
	StekPov<Osoba*>* S1 = new StekPov <Osoba*>;
	testiranje_steka_pov(S1);

	StekSekv<Osoba*>* S2 = new StekSekv <Osoba*>;
	testiranje_steka_sekv(S2);

}
```

DODATAK – Apstraktna klasa Stek

U prethodnom primjer je vidljivo da se funkcije **testiranje_steka_sekv** i **testiranje_steka_pov** razlikuju samo u jednoj liniji kôda, i to u tipu podatka formalnog parametra **stek**:

- `StekSekv<Osoba*>* stek`
- `StekPov<Osoba*>* stek`

Pozivi funkcija steka su identični u oba slučaja. Praktično je bolje rješenje da se koristiti jedna funkcija koja će kao argument primati različite vrste stekova. To se može postići ukoliko se implementiraju bazna klasa **Stek** (sa virtuelnim funkcijama) i izvedene klase **StekSekv** i **RedSekv.**

Virtuelne funkcije omogućavaju da se prilikom poziva funkcije preko pokazivača tipa bazne klasu ili reference baze klase poziva funkcija iz izvedene klase (ukoliko postoji nova definicija), a to se naziva "Polimorfizam". Ovo će biti detaljnije obrađeno u sklopu predmeta Programiranje III. U nastavku slijede jednostavni primjeri sa virtuelnim funkcijama.

U narednom primjeru je pozvana virtuelna funkcija **info** preko pokazivača **p** tipa baze klase koji pokazuje na objekat izvedene klasu.

```
class Osoba
{
public:
	int Id;
	string ime;
	Osoba(int id, string ime)
	{
		this->ime = ime;
		this->Id = id;
	}
	virtual void info()
	{
		cout << "Ja sam osoba " << ime << endl;
	}
};

class Student: public Osoba
{
public:
	int brojIndeksa;
	Student(int id, string ime, int brojIndeksa) : Osoba(id, ime)
	{
		this->brojIndeksa = brojIndeksa;
	}

	void info()
	{
		cout << "Ja sam student " << ime << ". Broj indeksa = " << brojIndeksa;
	}
};

void main()
{
	Student a(1, "A", 1001);
	a.info(); //ispis: "Ja sam student A. Broj indeksa = 1001"

	Osoba* p = &a;
	p->info(); //ispis: "Ja sam student A. Broj indeksa = 1001"
}
```

```
C:\WINDOWS\system32\cmd.exe
Ja sam student A. Broj indeksa = 1001
Ja sam student A. Broj indeksa = 1001
Press any key to continue . . .
```

Ukoliko se izostavi ključna riječ **virtual** onda se neće aktivirati polimorfizam:

```
class Osoba
{
public:
...
        virtual void info()
        {
                cout << "Ja sam osoba " << ime << endl;
        }
};

void main()
{
        Student a(1, "A", 1001);
        a.info(); //ispis: "Ja sam student A. Broj indeksa = 1001"

        Osoba* p = &a;
        p->info(); //ispis: "Ja sam osoba A. "
}
```

```
C:\WINDOWS\system32\cmd.exe
Ja sam student A. Broj indeksa = 1001
Ja sam osoba A
Press any key to continue . . .
```

Ukoliko se ne koristi pokazivač ili referenca (iako je funkcija **info** postavljena kao **virtual**) također se neće aktivirati polimorfizam:

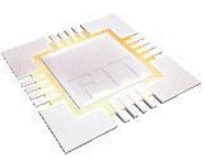
```
class Osoba
{
public:
...
        virtual void info()
        {
                cout << "Ja sam osoba " << ime << endl;
        }
};

void main()
{
        Student a(1, "A", 1001);
        a.info(); //ispis: "Ja sam student A. Broj indeksa = 1001"

        Osoba p = a;
        p.info(); //ispis: "Ja sam osoba A."
}
```

```
C:\WINDOWS\system32\cmd.exe
Ja sam student A. Broj indeksa = 1001
Ja sam osoba A
Press any key to continue . . .
```

Čiste virtuelne funkcije (pure virtual function) su virtualne funkcije koje nema definiciju u baznoj klasu. Umjesto bloka (vitičastih zagrada) stavlja se vrijednost 0, npr. "void naziv_funkcije=0;".



Apstrakta klasa je klasa koja ima više od jedne **čiste virtuelne funkcije** (pure virtual function).
Nije moguće instanciranje objekata apstraktne klase. Moguće instancirati samo objekte izvedene klase u kojoj su implementirane funkcije koje su u baznoj klasi označene kao čiste virtuelne.

U nastavku je data definicija apstraktne klase **Osoba**.

```
class Osoba
{
public:
...
	virtual void info() = 0;
};

class Student: public Osoba
{
public:
...
	void info()
	{
		cout << "Ja sam student " << ime << ". Broj indeksa = " << brojIndeksa;
	}
};
```

U nastavku je prikazan greška koja je pojavljuje prilikom instanciranja apstraktne klase **Osoba**.

```
void main()
{
    Osoba b(2, "B");
}
```

Error: object of abstract class type "Osoba" is not allowed:
function "Osoba::info" is a pure virtual function

Iako instaniciranje apstraktne klase nije moguće, dozvoljeno je koristiti reference ili pokazivače tipa bazne klase (koji pokazuju na objekat izvedene klase).

```
void main()
{
    Student a(1, "A", 1001);
    a.info();

    Osoba* p = &a;
    p->info();

    Osoba& b = a;
}
```

Naredni primjer apstraktne klase jeste bazna klasa Stek. Ona omogućava da druge klase implementiraju stek na različite načine ali sa istim ponašanjem (ukoliko se implementacija posmatra kao „crna kutija").

U nastavku je prikazana apstraktna klasa **Stek**:

Stek.h

```
#ifndef __STEK_H__
#define __STEK_H__

#include "Cvor.h"

template <class Tip>
class Stek
{

public:
	virtual void dodaj(Tip v) = 0;

	virtual Tip ukloni() = 0;

	virtual bool IsPrazan() = 0;

	virtual void print() = 0;
};

#endif
```

Funkcije steka su implementirane u izvedenim klasama (StekSekv i StekPov).

StekPov.h

```
#ifndef __STEK_POV_H__
#define __STEK_POV_H__

#include "Stek.h"

template <class Tip>
class StekPov: public Stek<Tip>
{
private:
	Cvor<Tip>* prvi;
public:
	StekPov() {  ...pogledati definiciju na str1...  }

	void dodaj(Tip v) {   ...pogledati definiciju na str1...    }

	Tip ukloni() {  ...pogledati definiciju na str1...    }

	bool IsPrazan() {  ...pogledati definiciju na str1...    }

	void print() {   ...pogledati definiciju na str1...    }
};

#endif
```

StekSekv.h

```
#ifndef __STEK_SEKV_H__
#define __STEK_SEKV_H__

#include "Stek.h"

template <class Tip>
class StekSekv: public Stek<Tip>
{
private:
	Tip* N;
```

```
        int brojac;
        int max;
        void prosiriStek() { ...pogledati definiciju na str2...  }
public:
        StekPov() {  ...pogledati definiciju na str2...  }

        void dodaj(Tip v) {  ...pogledati definiciju na str2...   }

        Tip ukloni() {  ...pogledati definiciju na str2...  }

        bool IsPrazan() {  ...pogledati definiciju na str2...  }

        void print() {   ...pogledati definiciju na str2...  }
};

#endif
```

Main.cpp

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

#include "StekSekv.h"
#include "StekPov.h"

void testiranje_steka (Stek<Osoba*>* stek)
{
        ...pogledati definiciju na str3 ili str4...
}

void main()
{
        Stek<Osoba*>* S1 = new StekPov <Osoba*>;
        testiranje_steka(S1);

        Stek<Osoba*>* S2 = new StekSekv <Osoba*>;
        testiranje_steka(S2);
}
```

Studenti ne moraju za prvi parcijalni ispit poznavati implementaciju apstrakte klase **Stek** (a za integralni ispit je to potrebno).