ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

ПОДАТОЧНИ СТРУКТУРИ И ПРОГРАМИРАЊЕ 2023/2024

Аудиториски вежби 10

Преоптоварување на оператори

Преоптоварување на оператори

Во C++ операторите за корисничките типови на податоци како што се класите, претставуваат специјални операторски функции. Операторските функции се именувани како `operator@`, каде `@` претставува стандарден оператор вграден во програмскиот јазик C++.

Операторските функции вршат преоптоварување (преклопување) на стандарните оператори: +, -, *, /, +=, -=, *=, /=, ++, --, <<, >>, итн.

Задачи

1. Да се напише класа за опис на ученици. За секој ученик се чува името како динамички алоцирано поле од знаци, просекот како децимален број и школска година како цел број. Класата треба да ги содржи следните функции и преоптоварени оператори:

- Конструктори
- get() и set() методи
- **Оператор =** за изедначување на два ученици
- **Оператор ++** за зголемување на запишаната школска година за еден.
- **Оператор <<** за печатење на ученик со сите негови податоци.

Потоа треба да се креира класа паралелка која во себе содржи динамички алоцирано поле од ученици, како и број на елементи во полето. Класата треба да ги содржи следните функции и преоптоварени оператори:

- Конструктори
- **Оператор ++** за зголемување на запишаната школска година за еден за сите ученици во полето.
- **Оператор +=** за додавање на еден ученик во полето од ученици

Оператор << за печатење на сите ученици во класата

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

class Uchenik
{
	char *ime;
	float prosek;
	int godina;

public:
	Uchenik()
	{
		ime = new char[1];
		strcpy(ime, "");
		prosek = 0.0;
		godina = 0;
	}

	Uchenik(char *i, float p, int g) :prosek(p), godina(g)
	{
		ime = new char[strlen(i) + 1];
		strcpy(ime, i);
	}
```

```
	Uchenik(const Uchenik &u) :prosek(u.prosek), godina(u.godina)
	{
		ime = new char[strlen(u.ime) + 1];
		strcpy(ime, u.ime);
	}

	~Uchenik() { delete[] ime; }

	char *getIme() { return ime; }
	float getProsek() { return prosek; }
	int getGodina() { return godina; }

	void setIme(char *i)
	{
		delete[] ime;
		/*не заборавајте најпрво да ја ослободите меморијата која ја зафаќа ime,
		а потоа да алоцирате нова меморија, онолку колку што зафаќа аргументот i*/
		ime = new char[strlen(i) + 1];
		strcpy(ime, i);
	}

	void setProsek(float p) { prosek = p; }
	void setGodina(int g) { godina = g; }

	Uchenik & operator= (Uchenik &u)
	{
		setIme(u.ime); /* ја повикуваме погоре дефинираната функција, која врши од
новo динамичка деалокација и алокација на меморија */
		prosek = u.prosek;
		godina = u.godina;
		return *this;
	}

	Uchenik & operator++() /*преоптоварување на операторот за инкрементирање */
	{
		godina++;
		return *this;
	}

	friend ostream & operator<<(ostream &out, Uchenik &u)
	{
		out << "Ime: " << u.ime << "\n\tProsek: " << u.prosek << "\n\tGodina: " <<
u.godina << endl;
		return out;
	}
};


class Paralelka
{
	Uchenik *klas;
	int broj;

public:
	Paralelka(Uchenik *u = NULL, int br = 0)
	{
		broj = br;
		klas = new Uchenik[broj];
		for (int i = 0; i < broj; i++)
			/* Напомена: ако не го имаме преоптоварено операторот =, ваквото
доделување може да доведе до неправилно функционирање на програмата */
			klas[i] = u[i];
	}

	Paralelka(const Paralelka &p)
```

```
	{
		broj = p.broj;
		klas = new Uchenik[broj];
		for (int i = 0; i < broj; i++)
			klas[i] = p.klas[i];
	}

	~Paralelka() { delete[] klas; }
	Paralelka operator++()
	{
		for (int i = 0; i < broj; i++)
			++klas[i];
		/*го повикуваме операторот ++ од класата Uchenik кој ја зголемува годината
на секој ученик за 1*/
		return *this;
	}

	Paralelka & operator+=(Uchenik &u)
	{
		Uchenik *pom;
		pom = new Uchenik[broj + 1];
		/* креираме помошно поле од ученици со простор за еден ученик повеќе */
		broj += 1;
		/* вкупниот број на ученици во полето ќе се зголеми за 1 */
		for (int i = 0; i < broj - 1; i++)
			pom[i] = klas[i];
		/* ги копираме сите тековни ученици во помошното поле */
		pom[broj - 1] = u;
		/* и го додаваме новиот ученик на последна позиција */
		delete[] klas;
		/* ослободуваме меморија за претходното поле од ученици */
		klas = pom;
		/* покажувачот да покажува кон полето со еден ученик повеќе*/
		return *this;
		/* го враќаме променетиот објект од класата Paralelka */
	}
	friend ostream & operator<<(ostream &out, Paralelka &p)
	{
		out << "Klasot ima " << p.broj << " uchenici:" << endl;
		for (int i = 0; i < p.broj; i++)
			out << p.klas[i];
		/* го повикуваме операторот << од класата Uchenik */
		return out;
	}
};

int main()
{
	Uchenik u[4] = { Uchenik(), Uchenik("Prv", 5.0, 2012), Uchenik("Vtor", 4.5, 2012),
Uchenik("Tret", 5.0, 2011) };
	cout << u[1]; /* оператор << од класата Uchenik */
	u[1] = u[2];  /* оператор = од класата Uchenik */
		      /* се повикува функцијата u[1].operator=(u[2]) */
	cout << u[1]; /* оператор << од класата Uchenik */
	Paralelka p = Paralelka(u, 4);
	cout << p;    /* оператор << од класата Paralelka */
	Uchenik nov = Uchenik("Nov", 5.0, 2012);
	p += nov;     /* оператор += од класата Paralelka */
		      /* се повикува функцијата p.operator+=(nov) */
	cout << p;    /* оператор << од класата Paralelka */
	++p;          /* оператор ++ од класата Paralelka */
	cout << p;
}
```