Лабораторная работа №5 «Перегрузка операторов»

Цель

Научиться перегружать бинарные и унарные операторы для пользовательского класса, применять перегруженные операторы для операций над объектами классов.

Теоретическая информация

Перегрузка операторов

В языке C++ для перегрузки операций (операторов) используется ключевое слово operator, с помощью которого определяется специальная операция-функция (operator function). Примерами перегружаемых операторов могут быть: +, -, *, /, ||, ==, ++, --, [], () и другие. Формат перегрузки операторов:

```
тип_возвр_значения operator знак_операции (специф_параметров)
{операторы_тела_функции}
```

Перегрузка унарных операторов

Любой унарный оператор (например, ++, --) может быть определен двумя способами: либо как функция-член без параметров, либо как глобальная функция с одним параметром. Унарные операторы, перегружаемые в рамках определенного класса, могут перегружаться только через нестатическую функцию без параметров. Вызываемый объект класса автоматически воспринимается как единственный операнд. Унарные операторы, перегружаемые вне области класса, должны иметь один параметр типа класса. Передаваемый через этот параметр объект воспринимается как операнд.

Для того, чтобы при реализации в классе, отличить **префиксную** (++a) и **постфиксную** (a++) формы реализации унарных операторов ++ или --, нужно придерживаться следующих правил:

– если перегружается префиксная форма оператора ++ или --, то в классе нужно реализовать operator++() или operator--() без параметров;

– если перегружается постфиксная форма оператора ++ или --, то в классе нужно реализовать operator++(int d) или operator--(int d) с одним целочисленным параметром. В этом случае параметр d не используется в функции. Он указывается только для того, чтобы указать что это именно постфиксная реализация оператора. Имя d параметра может быть заменено другим именем или вообще отсутствовать.

```
class Newspaper {          //класс
	Article* arr[4];
	int countArr;
public:
	void setArticle(Article* art, int i) {
		arr[i] = art;
		countArr++;
	}
	int getCountArray() {
		return countArr;
	}
	void operator++ () {//перегрузка оператора ++ внутри класса. Операция ++a
		if (getCountArray() !=0) {
			this->setArticle(arr[0], getCountArray()); //какие-то действия внутри
	//оператора. В данном случае добавляется еще один элемент массива, равный 0-му
		}
	}
	void operator++ (int) {    //перегрузка оператора ++ внутри класса. Операция a++
		if (getCountArray() !=0) {
			this->setArticle(arr[0], getCountArray());
		}
	}
};
```

Перегрузка бинарных операторов

Любой бинарный оператор может быть определен двумя способами: либо как функция класса с одним параметром, либо как глобальная функция с двумя параметрами.

```
class Newspaper {          //класс
	string name;
	Article* arr[4];
	int countArr;
public:
	string getName() {
		return name;
	}
	void setArticle(Article* art, int i) {
		arr[i] = art;
		countArr++;
	}
	int getCountArray() {
		return countArr;
	}
	void operator+ (Article* art) {   //перегрузка оператора + внутри класса
		this->setArticle(art, getCountArray());
	}
```

```
	Article* operator[](int i) {	//перегрузка оператора [] внутри класса
		return this->arr[i];
	}
};
```

Операторы, перегружаемые внутри класса, могут перегружаться только нестатическими функциями с параметром. Вызываемый объект класса автоматически воспринимается в качестве первого операнда. Операторы, перегружаемые вне области класса, должны иметь два операнда, один из которых должен иметь тип класса.

```
ostream& operator << (ostream &os, Newspaper &np)//глобальная перегрузка оператора <<
{
	return os << np.getName();
}
```

Перегрузка оператора присваивания

Оператор присваивания отличается тремя особенностями:

– операция не наследуется;

– операция определена по умолчанию для каждого класса в качестве операции поразрядного копирования объекта, стоящего справа от знака операции, в объект, стоящий слева;

– операция может перегружаться только в области определения класса. Это гарантирует, что первым операндом всегда будет левое выражение.

Формат перегруженного оператора присваивания:

```
имя_класса& operator=(имя_класса& объект);
```

Отметим две важные особенности функции operator=. Во-первых, в ней используется параметр-ссылка. Это необходимо для предотвращения создания копии объекта, передаваемого через параметр по значению. Во-вторых, operator=() возвращает не объект, а ссылку на него. Смысл этого тот же, что и при использовании параметра-ссылки – чтобы избежать создания временного объекта.

```
	Newspaper& operator=(Newspaper& np) {	//перегрузка оператора =
		for (int i = 0; i < np.getCountArray(); i++) {
			this->arr[i] = np.arr[i];
		}
		this->countArr = np.getCountArray();
		return *this;
	}
```

Перегрузка операторов сравнения

Перегруженный оператор сравнения не должен изменять операнды и должен возвращать bool.

Чаще всего пользовательские типы перегружают операторы < и ==, для того чтобы элементы этого типа можно было хранить в контейнерах и использовать в алгоритмах.

```
bool operator<(Scientific& obj2) {
	if (getName() < obj2.getName()) {
		return true;
	}
	else return false;
}
```

Задание

---

1. Дополнить и при необходимости модифицировать приложение, разработанное согласно варианта лабораторной работы №4.
2. Описать в основном классе оператор + для добавления объекта одного из классов-наследников во внутренний массив.
3. Описать в основном классе операторы ++ префиксный и постфиксный для добавления объектов по умолчанию в массив (по умолчанию считаем 0-й объект в массиве).
4. Описать в основном классе оператор [] для доступа к элементу массива по индексу.
5. Описать глобальный оператор << для вывода данных на экран.
6. Описать оператор =.
7. В классах-наследниках описать операторы сравнения <, >, ==. Сравнение производить по любому атрибуту базового класса или самого класса.
8. В основном теле программы дополнить основное меню, чтобы можно было продемонстрировать применение всех операторов, включая сравнение пар объектов классов-наследников.
9. Реализовать все необходимые проверки на ввод значений.
10. Сделать выводы.

Пример упрощенного варианта реализации

```
#include "pch.h"
#include <iostream>
#include <string>

using namespace std;

class Article {
	string name;
	string author;
public:
	string getName() {
		return name;
	}
	void setName(string name) {
		this->name = name;
	}
	string getAuthor() {
		return author;
	}
	void setAuthor(string author) {
		this->author = author;
	}
	Article() {
		name = "No name";
		author = "No author";
	}
	Article(string _name, string _author) {
		name = _name;
		author = _author;
	}
	~Article() {
	}
	virtual void print() = 0;
};

class Scientific : public Article {
	string science;
public:
	string getScience() {
		return science;
	}
	void setScience(string science) {
		this->science = science;
	}
	Scientific() {
		science = "No science";
	}
	Scientific(string _name, string _author, string science) : Article(_name, _author)
{
		this->science = science;
	}
	~Scientific() {
	}
	void print() {			//метод вывода
		cout << "Scientific -=-=- Name: " << getName() << " Author: " <<
getAuthor() << " Science: " << getScience() << endl;
	}
};

class News : public Article {
	string area;
public:
	string getArea() {
```

```
		return area;
	}
	void setArea(string area) {
		this->area = area;
	}
	News() {
		area = "No area";
	}
	News(string _name, string _author, string area) {
		this->setName(_name);
		this->setAuthor(_author);
		this->area = area;
	}
	~News() {
	}
	void print() {
		cout << "News -=-=- Name: " << getName() << " Author: " << getAuthor() << "
Area: " << getArea() << endl;
	}
};

class Newspaper {			//основной класс
	string name;
	Article* arr[6];
	int countArr=0;
public:
	string getName() {
		return name;
	}
	void setName(string name) {
		this->name = name;
	}
	void setArticle(Article* art, int i) {
		arr[i] = art;
		countArr++;
	}
	Article* getArticle(int i) {
		return arr[i];
	}
	void show() {
		cout << "-=-=- Newspaper -=-=- Name: " << getName() << endl;
		for (int i = 0; i < getCountArray(); i++) {
			arr[i]->print();
		}
	}
	int getCountArray() {
		return countArr;
	}
	void operator+ (Article* art) {				//перегрузка оператора +
		this->setArticle(art, getCountArray());
	}
	Article* operator[](int i) {				//перегрузка оператора []
		return this->arr[i];
	}
	void operator++() {					//перегрузка оператора ++: ++a
		if (getCountArray() !=0) {
			this->setArticle(arr[0], getCountArray());
		}
	}
	void operator++(int) {					//перегрузка оператора ++: a++
		if (getCountArray() != 0) {
			this->setArticle(arr[0], getCountArray());
		}
	}
	Newspaper& operator=(Newspaper& np) {			//перегрузка оператора =
```

```
            for (int i = 0; i < np.getCountArray(); i++) {
                this->arr[i] = np.arr[i];
            }
            this->countArr = np.getCountArray();
            return *this;
        }
};

ostream& operator << (ostream &os, Newspaper &np)//глобальная перегрузка оператора <<
{
        return os << np.getName();
}

int main()
{

        Scientific* sc1 = new Scientific();
        Scientific* sc2 = new Scientific("Biology article", "Petrov", "ecology");

        News* n1 = new News();
        News* n2 = new News("Top news", "Sidorov", "Politics");

        Newspaper np;
        np.setName("Vestnik");

        cout << "-=-=- Operator + -=-=- " << endl;
        np + sc1;                                   //применение оператора +
        np + sc2;
        np + n1;
        np + n2;
        np++;                                       //применение оператора a++
        ++np;                                       //применение оператора ++a
        np.show();

        cout << "-=-=- Operator [] -=-=- " << endl;
        np[3]->print();                             //применение оператора []

        cout << "-=-=- Operator << -=-=- " << endl;
        cout << np << endl;                         //применение оператора <<

        Newspaper np2;

        cout << "-=-=- Operator = -=-=- " << endl;
        np2 = np;                                   //применение оператора =
        np2.setName("My2");
        np2.show();

        delete sc1;
        delete sc2;
        delete n1;
        delete n2;
}
```

Варианты

Варианты распределяются аналогично предыдущим лабораторным работам.

Итоговый отчет

Укажите в отчете:

1. Тему лабораторной работы, свою фамилию и имя, группу, вариант задания;
2. Листинг программы;
3. Скриншоты результатов выполнения программы;
4. Краткие пояснения к алгоритму работы;
5. Выводы.

Контрольные вопросы

1. Чем отличаются бинарные и унарные операторы?
2. Как реализовать префиксные и постфиксные унарные операторы?
3. Можно ли несколько раз перегрузить один и тот же оператор в одном классе?
4. Как применить перегруженный оператор для операций с объектами своего класса?
5. Чем отличается перегрузка операторов внутри класса и перегрузка оператора как глобальной функции?