Лабораторная работа №3

Перегрузка операций

1. Цель задания:

1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2) Использование перегруженных операций в классах.

2. Теоретические сведения

2.1. Дружественные функции и классы

Если необходимо иметь доступ извне к скрытым полям класса, то есть расширить интерфейс класса, то можно использовать дружественные функции и дружественные классы.

Дружественные функции применяются для доступа к скрытым полям класса и представляют собой альтернативу методам. Метод, как правило, описывает свойство объекта, а в виде дружественных функций оформляются действия, не являющиеся свойствами класса, но концептуально входящие в его интерфейс и нуждающиеся в доступе к его скрытым полям, например, переопределенные операции вывода объектов

Правила описания и особенности дружественных функций:

- Дружественная функция объявляется *внутри класса,* к элементам которого ей нужен доступ, с ключевым словом `friend`. В качестве параметра ей должен передаваться объект или ссылка на объект класса, поскольку указатель `this` ей не передается.
- Дружественная функция может быть обычной функцией или методом другого ранее определенного класса. На нее не распространяется действие спецификаторов доступа, место размещения ее объявления в классе безразлично.
- Одна функция может быть дружественной сразу нескольким классам.

Пример

```
class student;//предварительное описание класса
class teacher
{
…
void teach(student &S);
….
};
class student
{
...
friend void teacher::teach(student&);//дружественная функция,
//имеет доступ к элементам класса student
…
};
```

Использования дружественных функций нужно по возможности избегать, поскольку они нарушают принцип инкапсуляции и, таким образом, затрудняют отладку и модификацию программы.

Если все методы какого-либо класса должны иметь доступ к скрытым полях другого, весь класс объявляется дружественным с помощью ключевого слова `friend`.

```
class student;//предварительное описание класса
class teacher
{
…
void teach(student &S);
….
};
class student
{
...
friend class teacher;//все функции класса teacher являются
// дружественными для класса student
…
};
```

2.3. Перегрузка унарных операций

Унарную операцию можно перегрузить:

- Как компонентную функцию класса
- Как внешнюю (глобальную) функцию

Унарная функция-операция, определяемая *внутри класса,* должна быть представлена с помощью нестатического метода без параметров, при этом операндом является вызвавший ее объект, например:

```
class Person
{
string name;
int age;
public:
Person(string, int);//конструктор
.....
//компонентная функция
Person& operator ++() //префиксная операция
{
++age;
 return *this; //указатель на объект, вызвавший метод

}
};

//в основной функции
Person p1(”Ivanov”,20);
++p1;
p1.Show();
```

Если функция определяется *вне класса,* она должна иметь один параметр типа класса:

```
class Person
{
string name;
int age;
public:
Person( string, int);//конструктор
…..
//внешняя дружественная функция
friend Person & operator ++(Person&) ;
```

```
};

Person & operator ++(Person& p) //префиксная операция
{
++p.age;
 return p;
}
//в основной функции
Person p1("Ivanov",20);
++p1;
p1.Show();
```

Операции постфиксного инкремента и декремента должны иметь первый параметр типа `int`. Он используется только для того, чтобы отличить их от префиксной формы.

2.4. Перегрузка бинарных операций

Бинарную операцию можно перегрузить:

- Как компонентную функцию класса
- Как внешнюю (глобальную) функцию

Бинарная функция-операция, определяемая *внутри класса,* должна быть представлена с помощью нестатического метода с параметрами, при этом вызвавший ее объект считается первым операндом:

```
class Person
{
string name;
int age;
public:
Person( string, int);//конструктор
…..
//компонентная функция
Person & operator +(int x)
{
age+=x;
 return *this;
}
};

//в основной функции
Person p1("Ivanov",20);
p1+2;
p1.Show();
```

Если функция определяется *вне класса*, она должна иметь два параметра типа класса:

```
class Person
{
string name;
int age;
public:
Person(string, int);//конструктор
…..
//внешняя дружественная функция
friend Person & operator +(Person& p, int x) ;
};
```

```
Person & operator +(Person &p, int x)
{
p.age+=x;
 return p;
}

//в основной функции
Person p1("Ivanov",20);
p1+2;
p1.Show();
```

2.5. Перегрузка операции присваивания

Операция присваивания определена в любом классе по умолчанию как поэлементное копирование. Эта операция вызывается каждый раз, когда одному существующему объекту присваивается значение другого. Если класс содержит поля, память под которые выделяется динамически, необходимо определить собственную операцию присваивания. Чтобы сохранить семантику присваивания, операция-функция должна возвращать ссылку на объект, для которого она вызвана, и принимать в качестве параметра единственный аргумент — ссылку на присваиваемый объект.

```
class Person
{
string name;
int age;
public:
Person(string, int);//конструктор
…..
//компонентная функция
Person& operator =(const Person&);
};

Person& Person::operator = (const& Person p)
{
//проверка на самоприсваивание
if (&p==this) return*this;
name = p.name;
age = p.age;
return *this;
 }

//в основной функции
Person p1("Ivanov",20);
Person p2;
p2=p1;
p1.Show();
p2.Show();
```

2.6. Перегрузка операций ввода-вывода

Операции ввода-вывода operator>> и operator<< всегда реализуются как внешние дружественные функции, т. к. левым операндом этих операций являются потоки. Для класса Person соответствующие операции могут выглядеть следующим образом:

```
class Person
{
string name;
int age;
public:
Person(string, int);//конструктор
…..
//дружественная глобальная функция
friend istream& operator>>(istream&in, Person&p);
friend ostream& operator<<(ostream&out, const Person&p);
};
……
istream&operator>>(istream&in, Person &p)
{
cout<<"name?"; in>>p.name;
cout<<"age?"; in>>p.age;
return in;
}
ostream&operator<<(ostream&out, const Person&p)
{

return (out<<p.name<<","<<p.age);
}
```

3. Постановка задачи

1. Определить пользовательский класс.
2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.
3. Определить в классе деструктор.
4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).
5. Перегрузить операцию присваивания.
6. Перегрузить операции ввода и вывода объектов с помощью потоков.
7. Перегрузить операции указанные в варианте.
8. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

4. Ход работы

Задача

Создать класс Time для работы с временными интервалами. Интервал должен быть представлен в виде двух полей: минуты типа int и секунды типа int. при выводе минуты отделяются от секунд двоеточием. Реализовать:

- сложение временных интервалов (+) (учесть, что в минуте не может быть более 60 секунд)
- добавление секунд (++) (постфиксную и префиксную формы)

1. Создать пустой проект способом, рассмотренным в лабораторной работе №2. Добавить в проект 3 файла: Time.h, Time.cpp, Lab3_main.cpp. В файл Time.h ввести следующий текст программы

```
//описание класса
#include <iostream>
using namespace std;
```

```
class Time
{
	int min, sec;
public:
	Time(){min=0;sec=0;};
	Time(int m, int s){min=m;sec=s;}
	Time(const Time&t){min=t.min;sec=t.sec;}
	~Time(){};
	int get_min(){return min;}
	int get_sec(){return sec;}
	void set_min(int m){min=m;}
	void set_sec(int s){sec=s;}
	//перегруженные операции
	Time&operator=(const Time&);
	Time& operator++();
	Time operator++(int);//постфиксная операция
	Time operator+(const Time&);
	//глобальные функции ввода-вывода
	friend istream& operator>>(istream&in, Time&t);
	friend ostream& operator<<(ostream&out, const Time&t);
};
```

2. В проект файл Time.cpp, добавим определение перегруженных операций класса Time

```
#include "Time.h"
#include <iostream>
using namespace std;
//перегрузка операции присваивания
Time&Time::operator=(const Time&t)
{
	//проверка на самоприсваивание
	if(&t==this) return *this;
	min=t.min;
	sec=t.sec;
	return *this;
}
//перегрузка префиксной операции инкремент
Time&Time::operator++()
{
	int temp=min*60+sec;
	temp++;
	min=temp/60;
	sec=temp%60;
	return *this;
}
//перегрузка постфиксной операции инкремент
Time Time::operator ++(int)
{
	int temp=min*60+sec;
	temp++;
	Time t(min,sec);
	min=temp/60;
	sec=temp%60;
	return t;
}
//перегрузка бинарной операции сложения
Time Time::operator+(const Time&t)
{
	int temp1=min*60+sec;
	int temp2=t.min*60+t.sec;
	Time p;
	p.min=(temp1+temp2)/60;
	p.sec=(temp1+temp2)%60;
	return p;
}
```

```
//перегрузка глобальной функции-операции ввода
istream&operator>>(istream&in, Time&t)
{
	cout<<"min?"; in>>t.min;
	cout<<"sec?"; in>>t.sec;
	return in;
}
//перегрузка глобальной функции-операции вывода
ostream&operator<<(ostream&out, const Time&t)
{

	return (out<<t.min<<" : "<<t.sec);
}
```

3. Добавим в файл Lab3_main.cpp основную программу, в которой проверим работу всех перегруженных функций.

```
#include "Time.h"
#include <iostream>
using namespace std;

void main()
{
	Time a;//конструктор без параметров
	Time b; //конструктор без параметров
	Time c; //конструктор без параметров
	cin>>a;//ввод переменной
	cin>>b;//ввод переменной
	++a;//префиксная операция инкремент
	cout<<a<<endl;//вывод переменной
	c=(a++)+b;//сложение и постфиксная операция инкремент
	cout<<"a="<<a<<endl; //вывод переменной
	cout<<"b="<<b<<endl; //вывод переменной
	cout<<"c="<<c<<endl; //вывод переменной
}
```

4. Выполнить компиляцию программы, используя команду Build / Build Solution или функциональную клавишу F7. Исправить имеющиеся синтаксические ошибки и снова запустить программу на компиляцию.
5. Запустить программу на выполнение, используя команду Debug/ Start Without Debugging или комбинацию функциональных клавиш Ctrl+F5.
6. Изучить полученные результаты, сделать выводы.

5. Варианты

№	Задание
1	Создать класс Time для работы с временными интервалами. Интервал должен быть представлен в виде двух полей: минуты типа int и секунды типа int. при выводе минуты отделяются от секунд двоеточием. Реализовать: – сложение временных интервалов (учесть, что в минуте не может быть более 60 секунд) – сравнение временных интервалов (==)
2	Создать класс Time для работы с временными интервалами. Интервал должен быть представлен в виде двух полей: минуты типа int и секунды типа int. при выводе минуты отделяются от секунд двоеточием. Реализовать: – вычитание временных интервалов (учесть, что в минуте не может быть более 60 секунд) – сравнение временных интервалов (!=)

3	Создать класс Time для работы с временными интервалами. Интервал должен быть представлен в виде двух полей: минуты типа int и секунды типа int. при выводе минуты отделяются от секунд двоеточием. Реализовать: – добавление секунд (учесть, что в минуте не может быть более 60 секунд) – вычитание секунд
4	Создать класс Time для работы с временными интервалами. Интервал должен быть представлен в виде двух полей: минуты типа int и секунды типа int. при выводе минуты отделяются от секунд двоеточием. Реализовать: – добавление секунд (учесть, что в минуте не может быть более 60 секунд) – сравнение временных интервалов (== и !=)
5	Создать класс Time для работы с временными интервалами. Интервал должен быть представлен в виде двух полей: минуты типа int и секунды типа int. при выводе минуты отделяются от секунд двоеточием. Реализовать: – вычитание секунд – сравнение временных интервалов (== и !=)
6	Создать класс Money для работы с денежными суммами. Число должно быть представлено двумя полями: типа long для рублей и типа int для копеек. Дробная часть числа при выводе на экран должна быть отделена от целой части запятой. Реализовать: – сложение денежных сумм, – вычитание денежных сумм,
7	Создать класс Money для работы с денежными суммами. Число должно быть представлено двумя полями: типа long для рублей и типа int для копеек. Дробная часть числа при выводе на экран должна быть отделена от целой части запятой. Реализовать: – деление сумм, – умножение суммы на дробное число.
8	Создать класс Money для работы с денежными суммами. Число должно быть представлено двумя полями: типа long для рублей и типа int для копеек. Дробная часть числа при выводе на экран должна быть отделена от целой части запятой. Реализовать: – сложение суммы и дробного числа – операции сравнения (>, <, ==).
9	Создать класс Money для работы с денежными суммами. Число должно быть представлено двумя полями: типа long для рублей и типа int для копеек. Дробная часть числа при выводе на экран должна быть отделена от целой части запятой. Реализовать: – вычитание дробного числа из суммы – операции сравнения (==, !=).
10	Создать класс Money для работы с денежными суммами. Число должно быть представлено двумя полями: типа long для рублей и типа int для копеек. Дробная часть числа при выводе на экран должна быть отделена от целой части запятой. Реализовать: – операции сравнения (==, !=). – вычитание копеек (--) (постфиксная и префиксная формы)

11	Создать класс Money для работы с денежными суммами. Число должно быть представлено двумя полями: типа long для рублей и типа int для копеек. Дробная часть числа при выводе на экран должна быть отделена от целой части запятой. Реализовать: – операции сравнения (<, >). – добавление копеек (++) (постфиксная и префиксная формы)
12	Создать класс Pair (пара чисел). Пара должна быть представлено двумя полями: типа int для первого числа и типа double для второго. Первое число при выводе на экран должно быть отделено от второго числа двоеточием. Реализовать: – операции сравнения (<, >). – операция ++, которая работает следующим образом: если форма операции префиксная, то увеличивается первое число, если форма операции постфиксная, то увеличивается второе число.
13	Создать класс Pair (пара чисел). Пара должна быть представлено двумя полями: типа int для первого числа и типа double для второго. Первое число при выводе на экран должно быть отделено от второго числа двоеточием. Реализовать: – операции сравнения (<, >). – операция --, которая работает следующим образом: если форма операции префиксная, то уменьшается первое число, если форма операции постфиксная, то уменьшается второе число.
14	Создать класс Pair (пара чисел). Пара должна быть представлено двумя полями: типа int для первого числа и типа double для второго. Первое число при выводе на экран должно быть отделено от второго числа двоеточием. Реализовать: – операции сравнения (==,!=). – вычитание константы из пары (уменьшается первое число, если константа целая, второе, если константа вещественная).
15	Создать класс Pair (пара чисел). Пара должна быть представлено двумя полями: типа int для первого числа и типа double для второго. Первое число при выводе на экран должно быть отделено от второго числа двоеточием. Реализовать: – вычитание пар чисел – добавление константы к паре (увеличивается первое число, если константа целая, второе, если константа вещественная).

6. Контрольные вопросы

1. Для чего используются дружественные функции и классы?
2. Сформулировать правила описания и особенности дружественных функций.
3. Каким образом можно перегрузить унарные операции?
4. Сколько операндов должна иметь унарная функция-операция, определяемая внутри класса?
5. Сколько операндов должна иметь унарная функция-операция, определяемая вне класса?
6. Сколько операндов должна иметь бинарная функция-операция, определяемая внутри класса?
7. Сколько операндов должна иметь бинарная функция-операция, определяемая вне класса?
8. Чем отличается перегрузка префиксных и постфиксных унарных операций?
9. Каким образом можно перегрузить операцию присваивания?
10. Что должна возвращать операция присваивания?
11. Каким образом можно перегрузить операции ввода-вывода?

12. В программе описан класс
```
class Student
{
…
Student& operator++();
….
};
```
и определен объект этого класса
Student s;
Выполняется операция
++s;
Каким образом, компилятор будет воспринимать вызов функции-операции?
13. В программе описан класс
```
class Student
{
…
friend Student& operator ++( Student&);
….
};
```
и определен объект этого класса
Student s;
Выполняется операция
++s;
Каким образом, компилятор будет воспринимать вызов функции-операции?
14. В программе описан класс
```
class Student
{
…
bool operator<(Student &P);

….
};
```
и определены объекты этого класса
Student a,b;
Выполняется операция
cout<<a<b;
Каким образом, компилятор будет воспринимать вызов функции-операции?
15. В программе описан класс
```
class Student
{
…
friend bool operator >(const Person&,  Person&)
….
};
```
и определены объекты этого класса
Student a,b;
Выполняется операция
cout<<a>b;

Каким образом, компилятор будет воспринимать вызов функции-операции?

7. Содержание отчета

1) Постановка задачи (общая и конкретного варианта).
2) Описание класса.
3) Определение компонентных функций.
4) Определение глобальных функций.
5) Функция main().
6) Объяснение результатов работы программы.
7) Ответы на контрольные вопросы