Лабораторная работа №7

Шаблоны классов

1. Цель задания:

1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2) Реализация шаблона класса-контейнера.

2. Теоретические сведения

С помощью шаблона функций можно отделить алгоритм от конкретных типов данных, передавая тип в качестве параметра. Шаблоны классов предоставляют аналогичную возможность, позволяя создавать параметризированные классы. Параметризированный класс создает семейство родственных классов, которые можно применять к любому типу данных, передаваемому в качестве параметра. Если использовать в качестве параметризированного класса контейнер (см. прошлую лекцию), то такой контейнер можно будет применять к любым типам данных, не переписывая код.

2.1. Шаблоны функций

Шаблоны вводятся для того, чтобы автоматизировать создание функций, обрабатывающих разнотипные данные. Например, алгоритм сортировки можно использовать для массивов различных типов. При перегрузке функции для каждого используемого типа определяется своя функция. Шаблон функции определяется один раз, но определение параметризируется, т. е. тип данных передается как параметр шаблона. Формат шаблона:

template <параметры_шаблона>
заголовок_функции
{тело функции}

Таким образом, шаблон семейства функций состоит из 2 частей – заголовка шаблона: template<список параметров шаблона> и обыкновенного определения функции, в котором вместо типа возвращаемого значения и/или типа параметров, записывается имя типа, определенное в заголовке шаблона.

Пример.

```
//шаблон функции, которая находит абсолютное значение числа любого типа
template<class type>//type – имя параметризируемого типа
type abs(type x)
{
if(x<0)return -x;
else return x;
}
```

Шаблон служит для автоматического формирования конкретных описаний функций по тем вызовам, которые компилятор обнаруживает в программе. Например, если в программе вызов функции осуществляется как abs(-1.5), то компилятор сформирует определение функции double abs(double x){. . . }.

2.2. Шаблоны классов

Шаблоны классов так же как и шаблоны функций поддерживают парадигму обобщенного программирования, т. е. программирования с использованием типов в качестве параметров. Механизм шаблонов в C++ допускает применение абстрактного типа в качестве параметра при определении класса. После того как шаблон класса

определен, он может использоваться для определения конкретных классов. Процесс генерации компилятором определения конкретного класса по шаблону класса и аргументам шаблона называется **инстанцированием** шаблона.
Определение шаблонного (обобщенного) класса имеет вид:
template <параметры шаблона>
class имя_класса
{…};

Пример:
```
template<classT>
class Point
{
T x,y;//координаты точки
public:
Point(T X=0,T Y=0):x(X),y(Y){}
void Show ();
};

template<classT>
void Point::Show()
{
cout<<”(“<<x<<” , ”<<y<<”)”;
}
};
```
При включении шаблона класса в программу никакие классы на самом деле не генерируются до тех пор, пока не будет создан экземпляр шаблонного класса, в котором вместо параметра шаблона указывается конкретный тип. Экземпляр создается либо объявлением объекта, либо объявлением указателя на инстанцированный шаблонный тип с присваиванием ему адреса с помощью операции new.
```
Point <int> a(13,15);
Point <float>*pa=new Point<float>(10.1,0.55);
```
Встретив такие объявления, компилятор генерирует код исходного класса.
В проекте, состоящем из нескольких файлов, определение шаблона класса обычно выносится в отдельный файл. Но для того, чтобы инстанцировался конкретный экземпляр шаблона класса необходимо, чтобы определение шаблона находилось в одной единице трансляции с этим экземпляром. Поэтому все определение шаблонного класса размещается в заголовочном файле, а затем этот файл подключается к нужным файлам с помощью директивы include. Чтобы этот файл не включался повторно, используется директива ifndef.
Пример.
```
//Point.h
#ifndef POINT_H
#define POINT_H
template<classT>
class Point
{
T x,y;//координаты точки
public:
Point(T X=0,T Y=0):x(X),y(Y){}
void Show () const;
};
template<classT>
```

```
void Point::Show()
{
cout<<”(“<<x<<” , ”<<y<<”)”;
}
};
#endif
////////////////////////////////////////
//main.cpp
#include “Point.h”
…..
void main()
{
Point<double>p1;
Point<int>p2(1,1);
p1.Show();
p2.Show();
}
```

2.3. Правила описания шаблонов

- Шаблоны методов (функций) не могут быть виртуальными.
- Шаблоны классов могут содержать статические элементы, дружественные функции и классы.
- Шаблоны могут быть производными как от шаблонов, так и от обычных классов, а также являться базовыми и для шаблонов, и для обычных классов.

3. Постановка задачи

1. Определить шаблон класса-контейнера (см. лабораторную работу №6).
2. Реализовать конструкторы, деструктор, операции ввода-вывода, операцию присваивания.
3. Перегрузить операции, указанные в варианте.
4. Инстанцировать шаблон для стандартных типов данных (int, float, double).
5. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы стандартных типов данных.
6. Реализовать пользовательский класс (см. лабораторную работу №3).
7. Перегрузить для пользовательского класса операции ввода-вывода.
8. Перегрузить операции необходимые для выполнения операций контейнерного класса.
9. Инстанцировать шаблон для пользовательского класса.
10. Написать тестирующую программу, иллюстрирующую выполнение операций для контейнера, содержащего элементы пользовательского класса.

4. Ход работы

Задача

1. Класс- контейнер ВЕКТОР с элементами типа int.
Реализовать операции:
[] – доступа по индексу;
() – определение размера вектора;
+ число – добавляет константу ко всем элементам вектора;

1. Пользовательский 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.

1. Создать пустой проект.
2. Добавить в него файл Vector. h.
3. В файл Vector.h добавить описание параметризированного класса (шаблона) Вектор:

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

template <class T> //T - параметр шаблона
class Vector
{
public:
//конструктор с параметрами: выделяет память под s элементов и заполняет их
//значением k
	Vector(int s,T k);
//конструктор с параметрами
	Vector(const Vector<T>&a);
//деструктор
	~Vector();
//оператор присваивания
	Vector&operator=(const Vector<T>&a);
//операция доступа по индексу
	T&operator[](int index);
//операция для добавление константы
	Vector operator+(const T k);
//операция, возвращающая длину вектора
	int operator()();
//перегруженные операции ввода-вывода
// <> - указывают на то, что функция является шаблоном
friend ostream& operator<< <>(ostream& out, const Vector<T>&a);
friend istream& operator>> <>(istream& in, Vector<T>&a);
private:
	int size;//размер вектора
	T*data;//укзатель на динамический массив значений вектора
};
```

4. В этот же файл добавить определение методов параметризированного класса Вектор:

```
/определение функций

//конструктор с параметрами
template <class T>
Vector<T>::Vector(int s,T k)
{
	size=s;
	data=new T[size];
	for(int i=0;i<size;i++)
		data[i]=k;
}
//конструктор копирования
template <class T>
Vector<T>::Vector(const Vector&a)
{
	size=a.size;
	data=new T[size];
	for(int i=0;i<size;i++)
		data[i]=a.data[i];
}
//деструктор
template <class T>
Vector<T>::~Vector()
```

```
{
	delete[]data;
	data=0;
}
//операция присваивания
template <class T>
Vector<T>&Vector<T>::operator=(const Vector<T>&a)
{
	if(this==&a)return *this;
	size=a.size;
	if (data!=0) delete[]data;
	data=new T[size];
	for(int i=0;i<size;i++)
		data[i]=a.data[i];
	return *this;
}
//операция доступа по индексу
template <class T>
T&Vector<T>::operator[](int index)
{
	if (index<size) return data[index];
	else  cout<<"\nError! Index>size";
}
//операция для добавления константы
template <class T>
Vector<T> Vector<T>::operator+(const T k)//+k
{

	Vector<T> temp(size,k);//инициализируем временный вектор любым значением
	for (int i=0;i<size;++i)
		temp.data[i]=data[i]+k;
	return temp;
}
//операция для получения длины вектора
template <class T>
int Vector<T>::operator ()()
{
	return size;
}
//операции для ввода-вывода
template <class T>
ostream&operator<< (ostream&out,const Vector<T>&a)
{
	for(int i=0;i<a.size;++i)
		out<<a.data[i]<<" ";
	return out;
}
template <class T>
istream&operator>> (istream&in,Vector<T>&a)
{
	for(int i=0;i<a.size;++i)
		in>>a.data[i];
	return in;
}
```

5. Добавить в проект файл lab7_main().cpp с функцией main(), в которой выполнить тестирование параметризированного класса Вектор для стандартного типа данных.

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

void main()
{
```

```
	//инициализация, ввод и вывод значений вектора
	Vector<int>A(5,0);
	cin>>A;
	cout<<A<<endl;
	//инициализация и вывод значений вектора
	Vector <int>B(10,1);
	cout<<B<<endl;
	//операция присваивания
	B=A;
	cout<<B<<endl;
	//доступ по индексу
	cout <<A[2]<<endl;
	//получение длины вектора
	cout<<"size="<<A()<<endl;
	//операция сложения с константой
	B=A+10;
	cout<<B<<endl;
}
```

6. Добавить в проект класс Time с минимальной функциональностью (конструкторы, деструктор, операция присваивания, операции ввода-вывода). Для этого в файл Time.h добавляем описание класса:

```
#include <iostream>
using namespace std;
class Time
{
public:
	Time(void);
	Time(int, int);
	Time(const Time&);
	Time&operator=(const Time&);
	//перегруженные операции ввода-вывода
friend ostream& operator<< (ostream& out, const Time&);
friend istream& operator>> (istream& in, Time&);
public:
	virtual ~Time(void){};
private:
	int min,sec;
};
```

В файл Time.cpp добавляем определение методов класса

```
#include "Time.h"
Time::Time(void)
{
	min=sec=0;
}


Time::Time(int M, int S)
{
	min=M;sec=S;
}
Time::Time(const Time&t)
{
	min=t.min;
	sec=t.sec;
}
Time&Time::operator =(const Time &t)
{
	min=t.min;
	sec=t.sec;
	return*this;
}
ostream&operator<<(ostream&out, const Time&t)
{
```

```
	out<<t.min<<" : "<<t.sec;
	return out;
}
istream&operator>>(istream&in,Time&t)
{
	cout<<"\nmin?"; in>>t.min;
	cout<<"\nsec?";in>>t.sec;
	return in;
}
```

7. В функцию main() добавить операторы для тестирования класса Time:

```
Time t;
cin>>t;
cout<<t;
```

8. В класс Time добавляем методы для реализации операций контейнера. В классе ВЕКТОР определена операция + число, которая добавляет константу ко всем элементам контейнера. Поэтому необходимо определить операцию + число для класса Time. Это можно сделать следующим образом. В описание класса Time (файл Time.h) добавить метод

```
Time operator+(Time k);
```

В файл Time.cpp добавить реализацию этого метода:

```
Time Time::operator+(Time k)
{
	int t=min*60+sec;
	int kt=k.min*60+k.sec;
	t+=kt;
	Time temp(t/60,t%60);
	return temp;
}
```

9. Протестируем сложение для класса Time. Для этого фукцию main() добавим операторы:

```
int k;
cout<<"k?";
cin>>k;
Time p;
p=t+k;
cout<<p;
```

10. Выполним тестирование параметризированного класса Вектор для пользовательского типа данных Time.

```
. ......
Time t;
cin>>t;
cout<<t;

Vector<Time>A(5,t);
cin>>A;
cout<<A<<endl;

Vector <Time>B(10,t);
cout<<B<<endl;

B=A;
cout<<B<<endl;
cout <<A[2]<<endl;
cout<<"size="<<A()<<endl;

B=A+t;
cout<<B<<endl;
```

5. Варианты

№	Задание
1	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] – доступа по индексу; () – определение размера вектора; + число – добавляет константу ко всем элементам вектора; Пользовательский 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.
2	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: []– доступа по индексу; int() – определение размера вектора; + вектор – сложение элементов векторов a[i]+b[i]; Пользовательский 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.
3	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] – доступа по индексу; + вектор – сложение элементов векторов a[i]+b[i]; + число – добавляет константу ко всем элементам вектора; Пользовательский 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.
4	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] – доступа по индексу; () – определение размера вектора; * число – умножает все элементы вектора на число; Пользовательский 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.
5	Класс- контейнер ВЕКТОР с элементами типа int. Реализовать операции: [] – доступа по индексу; int() – определение размера вектора; * вектор – умножение элементов векторов a[i]*b[i]; Пользовательский 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.

6	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу; () – определение размера множества; + – объединение множеств; Пользовательский 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.
7	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу; int() – определение размера вектора; * – пересечение множеств; Пользовательский 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.
8	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу; == - проверка на равенство; > число – принадлежность числа множеству; Пользовательский 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.
9	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу; != - проверка на неравенство; < число – принадлежность числа множеству; Пользовательский 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.
10	Класс- контейнер МНОЖЕСТВО с элементами типа int. Реализовать операции: [] – доступа по индексу; () – определение размера вектора; - – разность множеств; Пользовательский 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.

11	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: []– доступа по индексу; int() – определение размера списка; + вектор – сложение элементов списков a[i]+b[i]; Пользовательский 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.
12	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] – доступа по индексу; () – определение размера вектора; + число – добавляет константу ко всем элементам вектора; Пользовательский 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.
13	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] – доступа по индексу; + вектор – сложение элементов списков a[i]+b[i]; + число – добавляет константу ко всем элементам списка; Пользовательский 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.
14	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] – доступа по индексу; () – определение размера списка; * число – умножает все элементы списка на число; Пользовательский 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.
15	Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] – доступа по индексу; int() – определение размера списка; * вектор – умножение элементов списков a[i]*b[i]; Пользовательский 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.

6. Контрольные вопросы

1. В чем смысл использования шаблонов?
2. Каковы синтаксис/семантика шаблонов функций?
3. Каковы синтаксис/семантика шаблонов классов?

4. Что такое параметры шаблона функции?
5. Перечислите основные свойства параметров шаблона функции.
6. Как записывать параметр шаблона?
7. Можно ли перегружать параметризованные функции?
8. Перечислите основные свойства параметризованных классов.
9. Все ли компонентные функции параметризованного класса являются параметризованными?
10. Являются ли дружественные функции, описанные в параметризованном классе, параметризованными?
11. Могут ли шаблоны классов содержать виртуальные компонентные функции?
12. Как определяются компонентные функции параметризованных классов вне определения шаблона класса?
13. Что такое инстанцирование шаблона?
14. На каком этапе происходит генерирование определения класса по шаблону?

7. Содержание отчета

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