Лабораторная работа №6 «Шаблоны классов»

Цель

Научиться создавать шаблоны классов, применять шаблоны как для встроенных типов данных, так и для пользовательских классов, понимать особенности применения пользовательских классов в шаблонах.

Теоретическая информация

Понятие шаблона

В C++ определено два вида шаблонов: шаблоны-классы и шаблоны-функции. Основное отличие шаблона-функции от шаблона-класса в том, что не нужно сообщать компилятору, к каким типам параметров применяется функция, он сам может определить это по типам ее формальных параметров.

Шаблоны функций

Объявление шаблона функции начинается с заголовка, состоящего из ключевого слова template, за которым следует список параметров-типов шаблона:

```
template <class X> X min (X a, X b) {    // Описание шаблона функции
       return a<b ? a : b;
}
```

Идентификатор в списке параметров-типов шаблона X означает имя любого типа.

В описании заголовка функции этот же идентификатор означает тип возвращаемого функцией значения и типы параметров функции.

```
int m = min (1, 2); // Использование шаблона функции
```

Экземпляр шаблона функции породит компилятор (эта операция называется **инстанцированием** шаблона):

```
int min (int a, int b) { return a<b ? a : b; }
```

В списке параметров шаблона слово class может также относится к обычному типу данных. Так как T является параметром, обозначающим тип, шаблоны иногда называют параметризованными типами.

Требования к фактическим параметрам шаблона

Шаблон может быть использован не для любого типа данных. Ограничения на тип данных, который можно использовать для инстанцирования шаблона, зависит от характера операций, выполняемых в теле функций шаблона.

Используя свой класс для инстанцирования шаблонов, необходимо убедиться в том, что все операции, выполняемые с ним в шаблоне, определены для этого класса.

Шаблоны классов

Можно создавать шаблоны для классов, что позволяет работать с разными типами данных.

```
template<class T>
class MyArray {                    //создание шаблона
	T* arr;                    //массив
	int count = 0;             //количество добавленных элементов в массив
	int size;                  //размер массива
public:
	MyArray(int n) {
		arr = new T[n];
		size = n;
	}
	void addItem(T obj) {      //метод добавления элемента в массив
		if (count < size) {
			arr[count] = obj;
			count++;
		}
	}
};
```

В данном примере параметр-тип обозначен идентификатором T. Можно обратить внимание, что конструктор с параметрами для шаблона реализован в самом шаблоне. Если же реализация методов вынесена в отдельный файл, то она должна быть выполнена в форме шаблона функции в формате:

```
template<class T> MyArray<T>:: MyArray(int n) {
		arr = new T[n];
		size = n;
	}
```

Или для второго метода:

```
template<class T> void MyArray<T>:: addItem(T obj) {
		if (count < size) {
			arr[count] = obj;
			count++;
		}
	}
```

В шаблоне класса могут содержаться методы содержащие, например, операторы:

```
	int findItem(T obj) {		//подразумевается реализация внутри шаблона класса
		int index = -1;
		for (int i = 0; i < count; i++) {
			if (arr[i] == obj) {		//сравнение объектов оператором ==
				index = i;
				break;
			}
		}
		return index;
	}
```

Ранее уже отмечалось, что шаблоны могут работать только для тех типов данных или классов, которые поддерживают необходимые операции. В данном примере тип, которым может быть инстанцирован шаблон, должен иметь перегруженный оператор ==.

Список параметров-типов

Параметров-типов у шаблона может быть несколько. Они должны быть перечислены через запятую. Некоторые параметры-типы могут быть заданы явным указанием типа:

```
template <class T, int n> class Vector {
	T *coord;
	int current;
public:
	Vector();
	~Vector() {delete[] coord;}
};
```

Значение n, заданное в заголовке шаблона не используется в описании класса, но применяется в описании его методов. Конструктор Vector, использующий значение n для задания размера массива, выглядит так:

```
// конструктор
template <class T, int n> Vector <T, n>::Vector() {
	coord = new T[n]; current = 0;
}
```

Имеется несколько вариантов использования шаблонов с параметрами-значениями для динамического создания массивов различных размеров. Например, можно передать размер массива конструктору.

Задание

1. Дополнить и при необходимости модифицировать приложение, разработанное согласно варианта лабораторной работы №5.
2. Создать шаблон класса, в нем определить конструктор, в котором реализовано создание массива на заданное количество элементов n (количество передается в параметре конструктору).
3. В шаблоне определить метод добавления элемента массива и метод получения элемента массива по индексу.
4. В шаблоне определить метод *int findItem(T obj)*, который ищет элемент в массиве и возвращает индекс элемента в массиве, а в случае неудачи возвращает -1. Для работы этого метода потребуется перегрузка оператора == в классах-наследниках.
5. В шаблоне определить метод *T min()* и метод *T max()*, которые возвращают минимальное и максимальное значение из объектов массива. Для работы этих методов потребуется перегрузка операторов сравнения в классах-наследниках. Сравнение объектов производить по любому из атрибутов класса.
6. В шаблоне определить метод void *sort()*, который должен отсортировать массив по возрастанию.
7. В основном теле программы применить шаблон для разных типов данных: int, char, каждый из классов-наследников.
8. Реализовать все необходимые проверки
9. Дополнить меню программы чтобы можно было продемонстрировать работу всех функций.
10. Сделать выводы.

Пример упрощенного варианта реализации

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

using namespace std;

template<class T>
class MyArray {                    //создание шаблона
	T* arr;                    //массив
	int count = 0;             //количество добавленных элементов в массив
	int size;                  //размер массива
public:
	MyArray(int n) {
		arr = new T[n];
		size = n;
	}
	~MyArray() {
		delete[] arr;
	}
	void addItem(T obj) {      //метод добавления элемента в массив
		if (count < size) {
			arr[count] = obj;
			count++;
		}
	}

	T getItem(int i) {         //метод извлечения элемента из массива
		return arr[i];
	}

	int findItem(T obj) {      //метод поиска элемента в массиве
		int index = -1;
		for (int i = 0; i < count; i++) {
			if (arr[i] == obj) {
				index = i;
				break;
			}
		}
		return index;
	}


};


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() {
		cout << "Scientific -=-=- Name: " << getName() << " Author: "
			<< getAuthor() << " Science: " << getScience() << endl;
	}

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

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;
	}
```

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

int main()
{
	cout << "template with int" << endl;
	MyArray<int> intArr(3);			//применение шаблона к типу int
	intArr.addItem(12);
	intArr.addItem(22);
	intArr.addItem(31);
	for (int i = 0; i < 3; i++) {
		cout << intArr.getItem(i) << endl;
	}
	cout << "Index of 22: " << intArr.findItem(22) << endl;

	cout << endl << "template with char" << endl;
	MyArray<char> charArr(3);  //применение шаблона к типу char
	charArr.addItem('a');
	charArr.addItem('j');
	charArr.addItem('c');
	for (int i = 0; i < 3; i++) {
		cout << charArr.getItem(i) << endl;
	}
	cout << "Index of 'd': " << charArr.findItem('d') << endl;

	cout << endl << "template with Scientific" << endl;
	Scientific sc1;
	Scientific sc2("Biology article", "Petrov", "ecology");

	MyArray<Scientific> myArr1(2); //применение шаблона к пользовательскому классу
	myArr1.addItem(sc1);
	myArr1.addItem(sc2);
	for (int i = 0; i < 2; i++) {
		cout << myArr1.getItem(i).getName() << endl;
	}
	cout << "Index of Article " << sc2.getName() << ": " << myArr1.findItem(sc2)
		<< endl;

	cout << endl << "template with News" << endl;
	MyArray<News> myArr2(2); //применение шаблона к пользовательскому классу
	News n1;
	News n2("Top news", "Sidorov", "Politics");
	myArr2.addItem(n1);
	myArr2.addItem(n2);
	for (int i = 0; i < 2; i++) {
		cout << myArr2.getItem(i).getName() << endl;
	}
	cout << "Index of Article " << n1.getName() << ": " << myArr2.findItem(n1)
		<< endl;
}
```

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. Как создать шаблон класса?
2. Можно ли применять один и тот же шаблон для разных типов данных?
3. Может ли быть несколько параметров-типов у шаблона класса?
4. Можно ли в качестве параметра-типа указать пользовательский класс?
5. Если в методах шаблона класса применяется, например, оператор <, то какие ограничения накладываются на параметр-тип?