**Лабораторная работа № 4. “ШАБЛОНЫ ФУНКЦИЙ И**  
**КЛАССОВ”**  
**Цель.** Получить практические навыки создания шаблонов и использования их в программах С++ / C# / Java.  
**Основное содержание работы.**  
Создать шаблон заданного класса и использовать его для данных различных типов.  
**Краткие теоретические сведения.**  
**Шаблон функции.**  
Шаблон функции (иначе параметризированная функция) определяет общий набор операций (алгоритм), которые будут применяться к данным различных типов. При этом тип данных, над которыми функция должна выполнять операции, передается ей в виде параметра на стадии компиляции.  
В С++ параметризированная функция создается с помощью ключевого слова **template**. Формат шаблона функции:  
*template <class тип\_данных> тип\_возвр\_значения*  
*имя\_функции(список\_параметров){тело\_функции}*  
**Основные свойства параметров шаблона функции.**  
•Имена параметров шаблона должны быть уникальными во всем определении шаблона.  
•Список параметров шаблона не может быть пустым.  
•В списке параметров шаблона может быть несколько параметров, и каждому из них должно предшествовать ключевое слово class.  
•Имя параметра шаблона имеет все права имени типа в определенной шаблоном функции.  
•Определенная с помощью шаблона функция может иметь любое количество непараметризованных формальных параметров. Может быть непараметризованно и возвращаемое функцией значение.  
•В списке параметров прототипа шаблона имена параметров не обязаны совпадать с именами тех же параметров в определении шаблона. Имя параметра шаблона имеет все права имени типа в определенной шаблоном функции.  
•Определенная с помощью шаблона функция может иметь любое количество непараметризованных формальных параметров. Может быть непараметризованно и возвращаемое функцией значение.  
•В списке параметров прототипа шаблона имена параметров не обязаны совпадать с именами тех же параметров в определении шаблона.  
•При конкретизации параметризованной функции необходимо, чтобы при вызове функции типы фактических параметров, соответствующие одинаково параметризованным формальным параметрам, были одинаковы.

**Шаблон класса.**

Шаблон класса (иначе параметризованный класс) используется для построения родового класса. Создавая родовой класс, вы создаете целое семейство родственных классов, которые можно применять к любому типу данных. Таким образом, тип данных, которым оперирует класс, указывается в качестве параметра при создании объекта, принадлежащего к этому классу. Подобно тому, как класс определяет правила построения и формат отдельных объектов, шаблон класса определяет способ построения отдельных классов. В определении класса, входящего в шаблон, имя класса является не именем отдельного класса, а параметризованным именем семейства классов.  
Общая форма объявления параметризованного класса:  
*template <class тип\_данных> class имя\_класса { . . . };*  
**Основные свойства шаблонов классов.**  
•Компонентные функции параметризованного класса автоматически являются параметризованными. Их не обязательно объявлять как параметризованные с помощью *template.*  
•Дружественные функции, которые описываются в параметризованном классе, не являются автоматически параметризованными функциями, т.е. по умолчанию такие функции являются дружественными для всех классов, которые организуются по данному шаблону.  
•Если *friend*-функция содержит в своем описании параметр типа параметризованного класса, то для каждого созданного по данному шаблону  
класса имеется собственная *friend*-функция.  
•В рамках параметризованного класса нельзя определить *friend*-шаблоны (дружественные параметризованные классы).  
•С одной стороны, шаблоны могут быть производными (наследоваться) как от шаблонов, так и от обычных классов, с другой стороны, они могут использоваться в качестве базовых для других шаблонов или классов.  
•Шаблоны функций, которые являются членами классов, нельзя описывать как *virtual.*  
Локальные классы не могут содержать шаблоны в качестве своих элементов.  
**Компонентные функции параметризованных классов.**  
Реализация компонентной функции шаблона класса, которая находится вне определения шаблона класса, должна включать дополнительно следующие два элемента:

Определение должно начинаться с ключевого слова *template*, за которым следует такой же *список\_параметров\_типов* в угловых скобках, какой указан в определении шаблона класса.  
•За *именем\_класса,* предшествующим операции области видимости  
(**::**), должен следовать *список\_имен\_параметров* шаблона.  
*template<список\_типов>тип\_возвр\_значения имя\_класса<список\_*  
*имен\_ параметров> : : имя\_функции(список\_параметров){ . . . }*  
**Порядок выполнения работы.**  
1. Создать шаблон заданного класса. Определить конструкторы, деструктор, перегруженную операцию присваивания (“=”) и операции, заданные в варианте задания.  
2. Написать программу тестирования, в которой проверяется использование шаблона для стандартных типов данных.  
3. Выполнить тестирование.  
4. Определить пользовательский класс, который будет использоваться в качестве параметра шаблона. Определить в классе необходимые функции и перегруженные операции.  
5. Написать программу тестирования, в которой проверяется использование шаблона для пользовательского типа.  
6. Выполнить тестирование.  
**Методические указания.**  
1. Класс АТД реализовать как динамический массив. Для этого определение класса должно иметь следующие поля:  
− указатель на начало массива;  
− максимальный размер массива;  
− текущий размер массива.  
2. Для вода и вывода определить в классе функции **input** и **print**.  
3. Чтобы у вас не возникало проблем, аккуратно работайте с константными объектами. Например:  
•конструктор копирования следует определить так:  
MyTmp (**const** MyTmp& ob);  
•операцию присваивания перегрузить так:  
MyTmp& operator = (**const** MyTmp& ob);

4. Для шаблонов множеств, списков, стеков и очередей в качестве стандартных типов использовать символьные, целые и вещественные типы. Для пользовательского типа взять класс из лабораторной работы № 1.  
5. Для шаблонов массивов в качестве стандартных типов использовать целые и вещественные типы. Для пользовательского типа взять класс  
“комплексное число” *complex*.  
*class complex{*  
*int re; //действительная часть*  
*int im; // мнимая часть*  
*public;*  
*//необходимые функции и перегруженные операции*  
*};*  
6. Реализацию шаблона следует разместить вместе с определением в  
заголовочном файле.

7. Тестирование должно быть выполнено для всех типов данных и  
для всех операций.

8. Реализовать версии на C#, Java.

Пример С++.

#include <string.h>

#include <iostream>

using namespace std;

template <class T>

class Array

{

private:

T\* mass;

int size;

int index;

public:

Array(int n = 0); //конструктор по умолчанию

~Array(){

if(mass){

delete [] mass;

}

} //деструктор

Array(const Array& tam); //конструктор копирования

T& operator = (Array& tam); //перегрузка операции присваивания

const T& operator [] (int); //перегрузка оператора индексирования

void add();

void show\_list(); //вывод на экран

};

template <class T>Array<T>::Array(int n){

size = n;

index = 0;

if(n)

{

mass = new T [size];

}

else

{

mass = 0;

}

}

template<class T>Array<T>::Array(const Array& tam){

size = tam.size;

index = 0;

if(tam.mass)

{

mass = new T [size];

for(int i=0;i<size;i++)

{

mass[i] = tam.mass[i];

}

}

else

{

mass = 0;

}

}

template<class T>void Array<T>::show\_list(){

for(int i=0;i<index;i++){

cout<<mass[i]<<" ";

}

cout<<endl;

}

template<class T>const T& Array<T>::operator [] (int i){

if(i < size){

return mass[i];

}

else{

cout<<"Error"<<endl;

exit(1);

}

}

template<class T>T& Array<T>::operator = (Array& tam){

if (&tam == this) return \*this;//сравнение адресов

if (mass) delete[] mass;

size = tam.size;

if (tam.mass){

mass = new T [size];

for(int i=0;i<size;i++){

mass[i] = tam.mass[i];

}

}else

mass = 0;

return \*this;

}

template<class T>void Array<T>::add(){

T tamp;

for(int i=0;i<size;i++){

cout<<"Введите значение: ";

cin>>tamp;

mass[i] = tamp;

++index;

}

}

int \_tmain(int argc, \_TCHAR\* argv[])

{

setlocale(LC\_ALL,"Russian");

Array<char>tam1(3);

cout << "char pressed" << endl;

tam1.add();

Array<int>tam2(3);

cout << "int pressed" << endl;

tam2.add();

Array<float>tam3(3);

cout << "float pressed" << endl;

tam3.add();

system("pause");

cout << "CHAR :" << endl;

tam1.show\_list();

cout << "INT :" << endl;

tam2.show\_list();

cout << "FLOAT :" << endl;

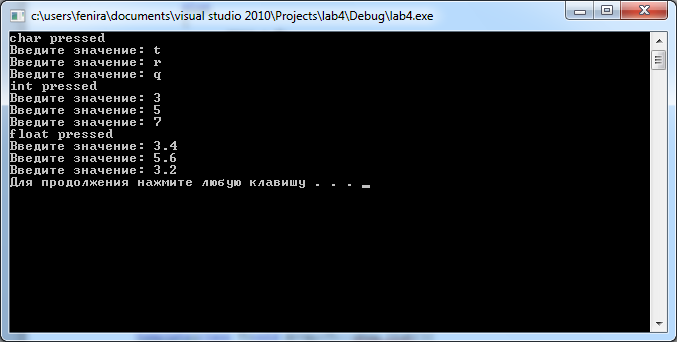
tam3.show\_list();

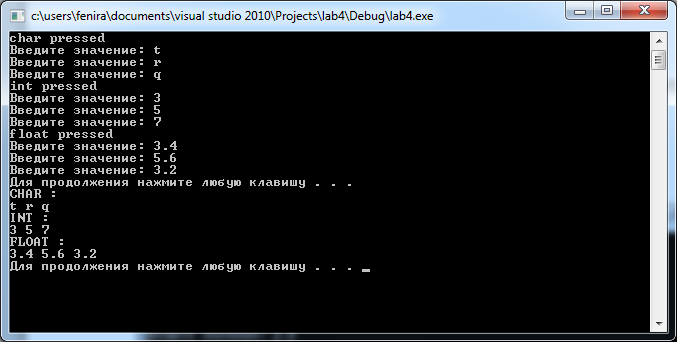
system("pause");

return 0;

}

Скриншоты работы:





Пример С#.

namespace lab4sharp

{

class Array<T>

{

private T[] data;

private int size;

public Array(int count)

{

size = count;

data = new T[size];

for (int i = 0; i < size; i++)

{

Console.Write("Введите значение: ");

data[i] = (T)Convert.ChangeType(Console.ReadLine(), typeof(T));

//Предыдущую строчку осознать не получится, просто запомните

}

}

public void show()

{

for (int i = 0; i < size; i++)

{

System.Console.Write(data[i]);

System.Console.Write(" ");

}

System.Console.WriteLine();

}

}

class Program

{

static void Main(string[] args)

{

System.Console.WriteLine("Вводим int");

Array<int> rr1 = new Array<int>(3);

System.Console.WriteLine("Вводим double");

Array<double> rr2 = new Array<double>(3);

System.Console.WriteLine("Вводим char");

Array<char> rr3 = new Array<char>(3);

System.Console.WriteLine("=================================");

System.Console.WriteLine("INT: ");

rr1.show();

System.Console.WriteLine("DOUBLE: ");

rr2.show();

System.Console.WriteLine("CHAR: ");

rr3.show();

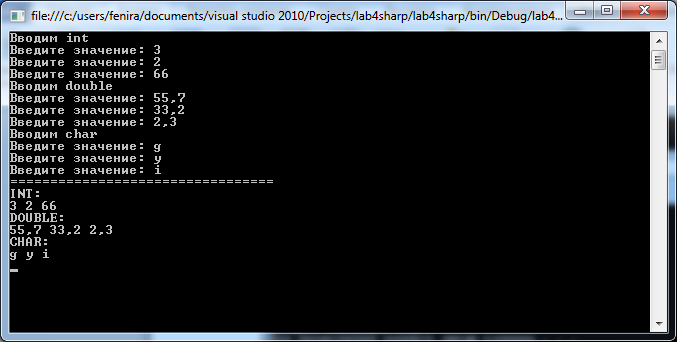
System.Console.ReadKey();

}

}

}

Скриншоты:



**Варианты заданий.**  
1.Класс − одномерный массив. Дополнительно перегрузить следую-  
щие операции:  
\* − умножение массивов;  
[] − доступ по индексу.  
2. Класс − одномерный массив. Дополнительно перегрузить следующие операции:  
int() − размер массива;  
[] − доступ по индексу.  
3. Класс − одномерный массив. Дополнительно перегрузить следующие операции:  
[] − доступ по индексу;  
== − проверка на равенство;  
!= − проверка на неравенство.  
4. Класс − множество set**.** Дополнительно перегрузить следующие  
операции:  
+ − добавить элемент в множество (типа set+item);  
+ − объединение множеств;  
\* − пересечение множеств;  
5. Класс − множество set**.** Дополнительно перегрузить следующие  
операции:  
+ − добавить элемент в множество (типа item + set);  
+ − объединение множеств;  
== − проверка множеств на равенство.  
6. Класс − множество set**.** Дополнительно перегрузить следующие  
операции:  
- − удалить элемент из множества (типа set-item);  
\* − пересечение множеств;  
< − сравнение множеств.  
7. Класс − множество set**.** Дополнительно перегрузить следующие  
операции:  
- − удалить элемент из множества (типа set-item);  
> − проверка на подмножество;  
!= − проверка множеств на неравенство.  
8.Класс − множество set. Дополнительно перегрузить следующие  
операции:  
+ − добавить элемент в множество (типа set+item);  
\* − пересечение множеств;  
int()− мощность множества.  
9. Класс − множество set**.** Дополнительно перегрузить следующие  
операции:  
() − конструктор множества (в стиле конструктора для множествен-  
ного типа в языке Pascal);  
+ − объединение множеств;  
<= − сравнение множеств.  
10. Класс − множество set**.** Дополнительно перегрузить следующие  
операции:  
> − проверка на принадлежность (типа операции **in** множественного  
типа в языке Pascal);  
\* − пересечение множеств;  
< − проверка на подмножество.  
11. Класс − однонаправленный список list**.** Дополнительно перегрузить следующие операции:  
+ − добавить элемент в начало (list+item);  
-- − удалить элемент из начала (--list);  
== − проверка на равенство.

12. Класс − однонаправленный список list**.** Дополнительно перегрузить следующие операции:  
+ − добавить элемент в начало (item+list);  
-- − удалить элемент из начала (--list);  
!= − проверка на неравенство.  
13. Класс − однонаправленный список list**.** Дополнительно перегрузить следующие операции:  
+ − добавить элемент в конец (list+item);  
-- − удалить элемент из конца (типа list--);  
!= − проверка на неравенство.  
14. Класс − однонаправленный список list**.** Дополнительно перегрузить следующие операции:  
[] − доступ к элементу в заданной позиции, например:  
Type c;  
int i;  
list L;  
c=L[i];  
+ − объединить два списка;  
= = − проверка на равенство.  
15. Класс − однонаправленный список list**.** Дополнительно перегрузить следующие операции:  
[] − доступ к элементу в заданной позиции, например:  
int i; Type c;  
list L;  
c=L[i];  
+ − объединить два списка;  
!= − проверка на неравенство.  
16. Класс − однонаправленный список list**.** Дополнительно перегрузить следующие операции:  
() − удалить элемент в заданной позиции, например:  
int i;  
list L;  
L(i);  
() − добавить элемент в заданную позицию, например:  
int i;  
Type c;  
list L;

L(с,i);  
!= − проверка на неравенство.  
17. Класс − стек stack**.** Дополнительно перегрузить следующие операции:  
+ − добавить элемент в стек;  
-- − извлечь элемент из стека;  
bool() − проверка, пустой ли стек.  
18. Класс − очередь queue. Дополнительно перегрузить следующие  
операции:  
+ − добавить элемент;  
-- − извлечь элемент;  
bool() − проверка, пустая ли очередь.  
19. Класс − одномерный массив. Дополнительно перегрузить следующие операции:  
+ − сложение массивов;  
[] − доступ по индексу;  
+ − сложить элемент с массивом.  
20. Класс − одномерный массив. Дополнительно перегрузить следующие операции:  
- − вычитание массивов;  
[] − доступ по индексу;  
- − вычесть из массива элемент.