# Лабораторная работа № 4 Тема. Шаблоны классов

**Теоретическое введение**. С помощью шаблонов можно создавать родовые ***(generic)*** функции и классы. Родовая функция определяет базовый набор операций, которые будут применяться к разным типам данных, получаемых функцией в качестве параметра.

Определение функции с ключевым словом ***template*** (шаблон) имеет вид:

**template<class *Ttype*>тип имя\_функции(список аргументов)**

**{//тело функции}**

Здесь ***Ttype*** – фиктивный тип, который используется при объявлении аргументов и локальных переменных функции. Компилятор заменит этот фиктивный тип на один из реальных и создаст соответственно несколько перегружаемых функций. При этом перегружаемые функции являются ограниченными, поскольку выполняют одни и те же действия над данными различных типов.

С помощью шаблона класса можно создать класс, реализующий стек, очередь, дерево и т. д. для любых типов данных. Компилятор будет генерировать правильный тип объекта на основе типа, задаваемого при создании объекта. Общая форма объявления шаблона класса:

**template <class *Ttype*> class имя\_класса {**

**//имена класса, поля класса**

**}**

Здесь ***Ttype*** – фиктивное имя типа, которое заменится компилятором на фактическое. Шаблон класса может иметь больше одного родового типа данных:

**template<class Type1,class Type2> class m {Type1 a;Type2 b;}**

**Пример.** Создается шаблон класса для работы с односвязным списком. Тестирование класса выполняется с использованием меню, которое позволяет создать список из чисел типов integer, float, double и выполнить типичные операции с ним.

#include <conio.h>

#include <string.h>

#include <iostream.h>

#include"vip\menu.cpp"

void ListForInt();

void ListForFloat();

void ListForDouble();

char bufRus[256];

char\*Rus(const char\*text){

CharToOem(text,bufRus);

return bufRus;

}

//шаблон класса для работы с односвязным списком

template<class mytype>class List {

//внутренний класс, для представления элементов списка

class Node{

public:

mytype d;

Node\* next;

Node(mytype dat=0){d=dat; next=0;}

};

Node\* pbeg; //указатель на начало списка

public:

List(){pbeg=0;} //конструктор

~List(); //деструктор

Node \* Add(mytype d); //добавление в конец списка

Node \* Find(mytype key); //поиск по ключу

Node \* Insert(mytype key,mytype d); //вставка узла d после узла с

// ключом key

bool Remove(mytype key); //удаление узла

void Print(); //печать списка

};

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*~List() \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//ДЕСТРУКТОР. Освобождает память для всех узлов списка

template<class mytype> List<mytype>::~List(){

if(pbeg!=0){

Node\* pv=pbeg;

while(pv){

pv=pv->next;

delete pbeg;

pbeg=pv;

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* void Add(mytype d) \*\*\*\*\*\*\*\*\*\*

//Добавляет узел в конец списка и возвращает указатель

// на вставленный узел.

template<class mytype> List<mytype>::Node\*

List<mytype>::Add(mytype d){

Node\* pv=new Node(d); //Создание нового узла

if(pbeg==0)pbeg=pv; //первый узел списка

else {

Node\* rab=pbeg;

while(rab!=0){

if((rab->next)==0){rab->next=pv;return pv;}

rab=rab->next;

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Node\* Find(mytype key)

//Выполняет поиск узла с заданным ключом и возвращает указатель

// на него в случае успешного поиска и 0 в случае отсутствия узла в списке

template<class mytype> List<mytype>::Node\*

List<mytype>::Find(mytype key){

Node\* pv=pbeg;

while(pv){

if((pv->d)==key)break;

pv=pv->next;

}

return pv;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\* Node\* Insert(mytype key,mytype d)

//Вставляет в список узел после узла с ключом key и возвращает

// указатель на вставленный узел. Если такого узла в списке нет,

// вставка не выполняется и возвращается значение 0

template<class mytype> List<mytype>::Node\*

List<mytype>::Insert(mytype key,mytype d){

if(Node\* pkey=Find(key)) //поиск узла с ключом key

{

Node\* pv=new Node(d);

//выделение памяти под новый узел и его инициализация

pv->next=pkey->next;

//установление связи нового узла с последующим

pkey->next=pv; //установление связи предыдущего узла с новым

return pv;

}

return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* bool Remove(mytype key)

//Удаляет узел с заданным ключом из списка и возвращает значение true при

//успешном удалении и false, если узел с таким ключом не найден

template<class mytype> bool List<mytype>::Remove(mytype key){

if(Node\* pkey=Find(key)){

if(pkey==pbeg)pbeg=pbeg->next; //удаление из начала списка

else{ //Находим указатель на узел,

Node\*rab=pbeg; //стоящий в списке перед

while(rab) //удаляемым узлом.

{ //rab-этот указатель.

if((rab->next)==pkey)break;

rab=rab->next;

}

rab->next=pkey->next;

}

delete pkey;

return true;

}

return false;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* void Print() -Печать списка

template<class mytype> void List<mytype>::Print(){

Node\*pv=pbeg;

cout<<Rus("Наш список:");cout<<endl;

while(pv){

cout<<pv->d<<' ';

pv=pv->next;

}

cout<<endl;}

//--------------------------- MAIN ---------------------------------------------

int main(int argc, char\* argv[]){

int k=0,max,kol;

char menu[][100]= {{" ListForInt "}, {" ListForFloat "},

{" ListForDouble "}, {" EXIT "}, };

kol=4; //КОЛИЧЕСТВО СТРОК МЕНЮ. Это используется в выравнивании

//строк меню по центру.

//----ВЫРАВНИВАНИЕ СТРОК МЕНЮ ПО ЦЕНТРУ------------------

max=viravnivaniestrok(menu,kol);

//----------------- МЕНЮ НА ЭКРАНЕ---------------------------------------

textmode(C80);

while(1){

switch(mmm(kol,menu,max,k))

{ case 0: {

ListForInt();

k=0;break;

}

case 1: {

ListForFloat();

k=1;break;

}

case 2: {

ListForDouble();

k=2;break;

}

case 3:{

exit(0);

}

}

}

return 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* void ListForInt()

//Эта функция вызывается из главного меню.

void ListForInt(){

List<int>l1;

int k=0,max,kol;

char menu[][100]=

{ {" PrintList "}, {" Add "}, {" Find "}, {" Insert "},

{" Remove "}, {" EXIT "}, {" Back "} };

kol=7; //КОЛИЧЕСТВО СТРОК МЕНЮ.

max=viravnivaniestrok(menu,kol);

//------------------------ МЕНЮ НА ЭКРАНЕ-----------------------------

textmode(C80);

while(1){

switch(mmm(kol,menu,max,k))

{ case 0: {

l1.Print();

while(!kbhit())

k=0;break;}

case 1: {

cout<<Rus("введите число, которое надо вставить:");

int t;cin>>t;

if( (l1.Add(t)) )cout<<Rus("вставка осуществлена");

else cout<<Rus("вставка не осуществлена");

while(!kbhit());

k=1;break;}

case 2: {

cout<<Rus("введите искомое число:");

int t;

cin>>t;

if(l1.Find(t))cout<<Rus("искомое число есть в списке.");

else cout<<Rus("искомого числа нет в списке.");

while(!kbhit());

k=2;break;}

case 3: {

cout<<Rus("введите число, которое надо вставить:");

int t;cin>>t;

cout<<Rus("введите число, после которого надо вставить:");

int key;cin>>key;

if( (l1.Insert(key,t)) )cout<<Rus("вставка осуществлена");

else cout<<Rus("вставка не осуществлена");

while(!kbhit());

k=3;break;}

case 4: {

cout<<Rus("введите число, которое надо удалить:");

int t;cin>>t;

if( (l1.Remove(t)) )cout<<Rus("удаление осуществлено");

else cout<<Rus("такого числа нет в списке.");

while(!kbhit());

k=4;break;}

case 5:{exit(0);}

} } }

Таким же образом, как и функция ListForInt(), реализуются функции List­ForFloat() и ListForDouble(), предназначенные для тестирования списков из чисел типа float и double, соответственно.

Задания для самостоятельного решения

Для разработки шаблонов классов можно использовать результаты выполнения лабораторных работ № 2 и № 3. При тестировании со-зданных шаблонов классов необходимо создавать объекты с различными допустимыми значениями параметров шаблона (например, компоненты вектора могут быть целыми, действительными или комплексными числами).

1. Создать шаблон класса для работы со стеком. Применить его для решения задач № 1 – 4 (лаб. работа № 3).
2. Создать шаблон класса для работы с одномерным массивом. Выполнить тестирование путем создания и обработки массивов, содержащих элементы различных типов (например, сортировка элементов массивов различными методами).
3. Создать шаблон класса **Vector** размерности *n* (см. задачу № 3, лаб. работа № 2).
4. Создать шаблон класса «Квадратная матрица» – **Matrix** размерно­сти *n×n* (см. задачу № 4, лаб. работа № 2).
5. Создать шаблон класса **Polynom** степени *n* (см. задачу № 5, лаб. работа № 2) или создать шаблон класса для работы с односвязным списком. Применить его для решения задачи № 5 (лаб. работа № 3).
6. Создать шаблон класса для работы с односвязным списком. При­менить шаблон класса для решения задачи № 6 (лаб. работа № 3).
7. Создать шаблон класса для работы с односвязным списком. При­менить шаблон класса для решения задачи № 7 (лаб. работа № 3).
8. Создать шаблон класса для работы с бинарным деревом. Приме­нить его для сортировки действительных чисел и строк, вводимых с клавиатуры или из файла.
9. Создать шаблон класса **Set** (множество) мощности *n* (см. задачу № 9, лаб. работа № 2 ) или создать шаблон класса для работы с односвязным списком. Применить шаблон класса для решения задачи № 9 (лаб. работа № 3).
10. Создать шаблон класса для работы с односвязным списком. Применить шаблон класса для решения задачи № 10 (лаб. работа № 3).
11. Создать шаблон класса, обеспечивающего описание матрицы заданного размера *n×m* и любого минора в ней (см. задачу № 11, лаб. работа № 2) или создать шаблон класса для работы с двусвязным списком. Применить шаблон класса для решения задачи № 11 (лаб. работа № 3).
12. Создать шаблон класса для работы с бинарным деревом. Применить шаблон класса для решения задачи № 12 (лаб. работа № 3).
13. Создать шаблон класса для работы с бинарным деревом. Применить шаблон класса для решения задачи № 13 (лаб. работа № 3).
14. Создать шаблон класса для работы с двусвязным списком. Применить шаблон класса для решения задачи № 15 (лаб. работа № 3).
15. Создать шаблон класса для работы с односвязным списком. Применить шаблон класса для решения задачи № 16 (лаб. работа № 3).

Тесты

1. Отметьте все утверждения, которые считаете верными:

1) нельзя с помощью шаблона создать функцию с таким же име­нем, как у явно определенной функции;

2) цель введения шаблонов – создание функций, которые могут обрабаты­вать однотипные данные;

\*3) при использовании шаблонов функций возможна перегрузка как с по­мощью шаблонов, так и функций;

\*4) в качестве описания шаблона функции используется прото­тип шаб­лона:



2. Можно ли задать шаблон со значением параметра по умолча­нию?

Варианты ответа:

\*1) да; 2) нет.

3. Каков правильный заголовок шаблона?

Варианты ответа:

\*1) template<class t1,class t2>; 2) template <class t1,t2>; 3) template <class t,class t>;

4. Сколько параметров может быть у шаблона при определении шаблона функции?

Варианты ответа:

1) 1; 2) столько, сколько аргументов у функции; \*3) столько, сколько типов ис­пользуется для параметризации.

5. Отметьте правильный вариант описания шаблона семейства функций:

1) template(class T)

void func(T\* p1,T\* p2){…}

2) template <class T>;

void func(T\* p1,T\* p2){…}

\*3) template<class T>

void func(T\* p1,T\* p2){…}

4) template<class T>

void func(T\* p1,T\* p2){…}

6. Можно ли использовать класс-шаблон в качестве базового класса?

Варианты ответа:

\*1) да; 2) нет.