Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Классы и объекты. Шаблоны классов»

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Желнин Н.А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

Постановка задачи

Определить шаблон класса - контейнера, конструкторы, операции ввода, вывода, присваивания, перегрузить операции, указанные в варианте.

Класс – контейнер СПИСОК.

Операции:

Доступ по индексу []

Определение размера списка ()

Умножение всех элементов на число \*

Пользовательский класс pair.

Алгоритм решения

1. Создаем класс – контейнер СПИСОК.
2. Создаем класс pair.
3. Создаем методы инициализации, ввода, вывода и т.д.
4. Создаем класс – итератор для проходки по списку.
5. В главной функции последовательно проверяем работоспособность всех функции.

Код программы

*MAIN*

*#include "list.h"*

*#include "pair\_head.h"*

*int main() {*

*Pair p1(2, 2.5);*

*List<Pair> a(3, p1);*

*cout << a << endl;*

*List<Pair> b(a);*

*cout << b << endl;*

*List<Pair> c = a \* b;*

*cout << c << endl;*

*Pair num;*

*cout << "Write a number:\n"; cin >> num;*

*b.pushback(num);*

*b.popfront();*

*for (Iterator<Pair> iter = b.first(); iter != b.last(); ++iter)*

*cout << \*iter << '\n';*

*return 0;*

*}*

*LIST.H*

*#pragma once*

*#include <iostream>*

*using namespace std;*

*#ifndef LIST\_H*

*#define LIST\_H*

*template <class T>*

*class List;*

*template <class T>*

*struct Node {*

*Node<T>\* prev = nullptr;*

*Node<T>\* next = nullptr;*

*T data;*

*};*

*template <class T>*

*class Iterator {*

*template <class A>*

*friend class List;*

*Node<T>\* elem;*

*public:*

*Iterator() { elem = nullptr; }*

*Iterator(const Iterator<T>& it) { elem = it.elem; }*

*Iterator<T>& operator=(const Iterator<T>& a) {*

*elem = a.elem;*

*return \*this;*

*}*

*bool operator==(const Iterator<T>& it) { return elem == it.elem; }*

*bool operator!=(const Iterator<T>& it) { return elem != it.elem; };*

*Iterator<T>& operator++() {*

*elem = elem->next;*

*return \*this;*

*}*

*Iterator<T>& operator--() {*

*elem = elem->prev;*

*return \*this;*

*}*

*Iterator<T>& operator+(const int& a) {*

*for (int i = 0; i < a; i++) elem = elem->next;*

*return \*this;*

*}*

*Iterator<T>& operator-(const int& a) {*

*for (int i = 0; i < a; i++) elem = elem->prev;*

*return \*this;*

*}*

*T& operator \*() const { return elem->data; }*

*};*

*template <class T>*

*class List {*

*public:*

*T front();*

*T back();*

*void pushback(T data);*

*void pushfront(T data);*

*void popback();*

*void popfront();*

*bool empty();*

*List<T>& operator=(const List<T>& a);*

*T& operator[](int index);*

*int& operator()();*

*List<T> operator\*(List<T>&);*

*template <class T2>*

*friend ostream& operator<< (ostream&, const List<T2>&);*

*template <class T2>*

*friend istream& operator>> (istream&, const List<T2>&);*

*Iterator<T> first() { return beg; }*

*Iterator<T> last() { return end; }*

*List();*

*List(int s, T k);*

*List(const List<T>&);*

*~List();*

*protected:*

*int size;*

*Node<T>\* head = nullptr;*

*Node<T>\* tail = nullptr;*

*Iterator<T> beg, end;*

*};*

*template <class T>*

*List<T>::List(int s, T k) {*

*size = s;*

*Node<T>\* node = new Node<T>;*

*node->data = k;*

*head = node;*

*tail = node;*

*for (int i = 0; i < size - 1; i++) {*

*Node<T>\* node = new Node<T>;*

*node->data = k;*

*tail->next = node;*

*node->prev = tail;*

*tail = node;*

*}*

*tail->next = nullptr;*

*beg.elem = head;*

*end.elem = tail->next;*

*}*

*template <class T>*

*List<T>::List() {*

*size = 0;*

*head = nullptr;*

*tail = nullptr;*

*}*

*template <class T>*

*List<T>::List(const List<T>& a) {*

*Node<T>\* node = a.head;*

*while (node != nullptr) {*

*pushback(node->data);*

*node = node->next;*

*}*

*beg = a.beg;*

*end = a.end;*

*}*

*template <class T>*

*List<T>::~List() {*

*Node<T>\* curr = head;*

*while (curr != nullptr) {*

*head = curr->next;*

*delete curr;*

*curr = head;*

*}*

*}*

*template <class T>*

*void List<T>::pushback(T data) {*

*Node<T>\* newNode = new Node<T>;*

*newNode->data = data;*

*if (head == nullptr) {*

*head = newNode;*

*tail = newNode;*

*size++;*

*end.elem = tail->next;*

*}*

*else {*

*tail->next = newNode;*

*newNode->prev = tail;*

*tail = newNode;*

*size++;*

*end.elem = tail->next;*

*}*

*}*

*template <class T>*

*void List<T>::pushfront(T data) {*

*Node<T>\* newNode = new Node<T>;*

*newNode->data = data;*

*if (head == nullptr) {*

*head = newNode;*

*tail = newNode;*

*size++;*

*beg.elem = head;*

*}*

*else {*

*head->prev = newNode;*

*newNode->next = head;*

*head = newNode;*

*size++;*

*beg.elem = head;*

*}*

*}*

*template <class T>*

*T List<T>::front() { return head->data; }*

*template <class T>*

*T List<T>::back() { return tail->data; }*

*template <class T>*

*void List<T>::popback() {*

*if (head != nullptr) {*

*Node<T>\* curr = tail;*

*tail = curr->prev;*

*delete curr;*

*tail->next = nullptr;*

*size--;*

*end.elem = tail->next;*

*}*

*}*

*template <class T>*

*void List<T>::popfront() {*

*if (head != nullptr) {*

*Node<T>\* curr = head;*

*head = curr->next;*

*delete curr;*

*head->prev = nullptr;*

*size--;*

*beg.elem = head;*

*}*

*}*

*template <class T>*

*bool List<T>::empty() { return size == 0; }*

*template <class T>*

*List<T>& List<T>::operator=(const List<T>& a) {*

*if (this == &a) return \*this;*

*Node<T>\* node = head;*

*while (node != nullptr) {*

*head = node->next;*

*delete node;*

*node = head;*

*size--;*

*}*

*Node<T>\* curr = a.head;*

*while (curr != nullptr) {*

*pushback(curr->data);*

*curr = curr->next;*

*}*

*beg = a.beg;*

*end = a.end;*

*return \*this;*

*}*

*template <class T>*

*T& List<T>::operator[](int index) {*

*if (index < size && index >= 0) {*

*Node<T>\* curr = head;*

*for (int i = 0; i < index; i++) curr = curr->next;*

*return curr->data;*

*}*

*else {*

*cout << "Bad index!" << endl;*

*exit(222);*

*}*

*}*

*template <class T>*

*int& List<T>::operator ()() {*

*return size;*

*}*

*template <class T>*

*List<T> List<T>::operator\*(List<T>& a) {*

*int tempsize;*

*if (size > a.size) tempsize = a.size;*

*else tempsize = size;*

*List<T> templist(tempsize, a[0]);*

*for (int i = 0; i < tempsize; i++)*

*templist[i] = (\*this)[i] \* a[i];*

*return templist;*

*}*

*template <class T>*

*ostream& operator<< (ostream& out, const List<T>& a) {*

*Node<T>\* curr = a.head;*

*while (curr != nullptr) {*

*out << curr->data << '\n';*

*curr = curr->next;*

*}*

*return out;*

*}*

*template <class T>*

*istream& operator>>(istream& in, const List<T>& a) {*

*Node<T>\* curr = a.head;*

*int i = 1;*

*while (curr != nullptr) {*

*cout << i++ << ":\n";*

*in >> curr->data;*

*curr = curr->next;*

*}*

*return in;*

*}*

*#endif*

*PAIR.H*

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Pair {

private:

int first;

double second;

public:

Pair() {

first = 0;

second = 0;

};

Pair(int F, double S) {

first = F;

second = S;

};

Pair(const Pair& t) {

first = t.first;

second = t.second;

};

~Pair() {};

int get\_sirst() { return first; }

double get\_second() { return second; }

void set\_first(int F) { first = F; }

void set\_second(double S) { second = S; }

Pair operator -(const Pair& p);

Pair& operator +(const int X);

Pair& operator +(const double X);

Pair operator \*(Pair&);

friend istream& operator>>(istream& in, Pair& p);

friend ostream& operator<<(ostream& out, const Pair& p);

};

Pair& Pair::operator +(const int X) {

first += X;

return \*this;

}

Pair& Pair::operator +(const double X) {

second += X;

return \*this;

}

Pair Pair::operator -(const Pair& p) {

Pair temp;

temp.first = this->first - p.first;

temp.second = this->second - p.second;

return temp;

}

istream& operator >>(istream& in, Pair& t) {

cout << "int first = "; in >> t.first;

cout << "double second = "; in >> t.second;

return in;

}

ostream& operator <<(ostream& out, const Pair& t) {

return (out << t.first << " : " << t.second);

}

Pair Pair::operator\*(Pair& p) {

Pair temp;

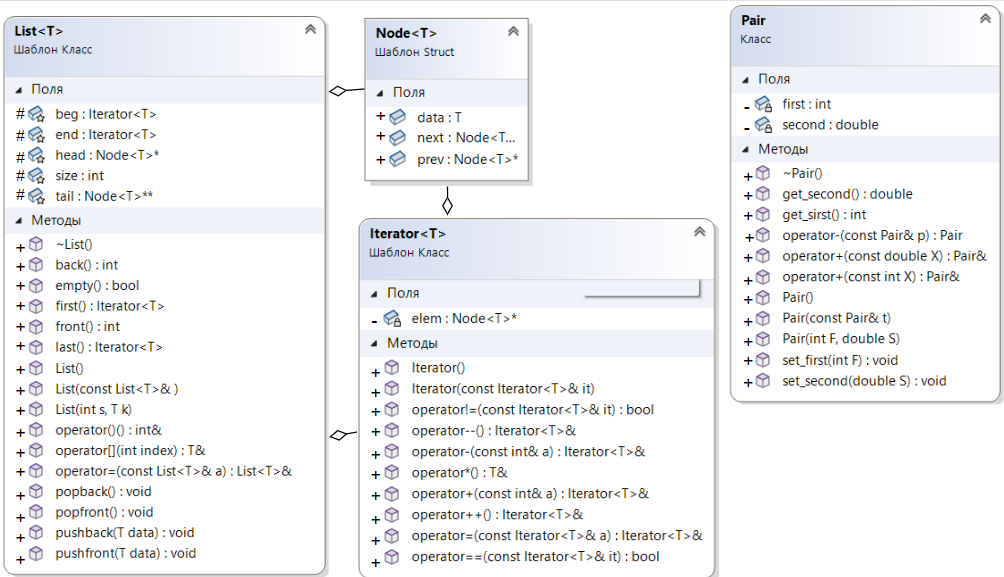
temp.first = (\*this).first \* p.first;

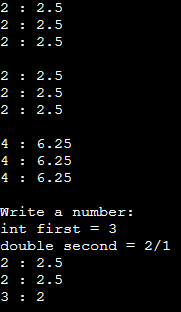
temp.second = (\*this).second \* p.second;

return temp;

}

Диаграмма и вывод в консоль





Контрольные вопросы

1. В чем смысл использования шаблонов?

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, то такой контейнер можно будет применять к любым типам данных, не переписывая код.

Шаблон служит для автоматического формирования конкретных описаний

функций по тем вызовам, которые компилятор обнаруживает в программе.

2. Каковы синтаксис/семантика шаблонов функций?

template <параметры\_шаблона>

заголовок\_функции (параметры\_функции)

{тело функции}

*Пример*

template <typename T>

T abs(T x) {

if (x>0) return x;

else return -x;

}

3. Каковы синтаксис/семантика шаблонов классов?

template <параметры шаблона>

class имя\_класса

{…};

*Пример*

template <class T>

class Point {

T x,y;//координаты точки

public:

Point(T X=0,T Y=0):x(X),y(Y){}

void Show ();

};

template<class T>

void Point::Show() { cout<<”(“<<x<<” , ”<<y<<”)”;}

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);

4. Что такое параметры шаблона функции?

Можно считать, что параметры шаблона являются его формальными параметрами, а типы тех параметров, которые используются в конкретных обращениях к функции, служат фактическими параметрами шаблона. Именно по ним выполняется параметрическая настройка и с учетом этих типов генерируется конкретный текст определения функции.

template <typename T>

T abs(T x) {

if (x>0) return x;

else return -x;

}

Если в программе вызов функции осуществляется как abs(-1.5), то компилятор формирует определение функции double abs(double x)

5. Перечислите основные свойства параметров шаблона функции.

* Имена параметров шаблона должны быть уникальными во всем определении шаблона.
* Список параметров шаблона не может быть пустым, для того, чтобы компилятор мог инстанцировать шаблон.
* В списке параметров шаблона может быть несколько параметров, и каждому из них должно предшествовать ключевое слово class.  
  template<class type1, class type2>  
  Соответственно, неверен заголовок:  
  template<class type1, type2, type3>
* Недопустимо использовать в заголовке шаблона параметры с одинаковыми именами, то есть ошибочен такой заголовок:  
  template<class t, class t, class t>
* Имя параметра шаблона имеет все права имени типа в определенной шаблоном функции.
* Все параметры шаблона функций должны быть обязательно использованы в спецификациях параметров определения функции. Таким образом, будет ошибочным такой шаблон:

template<class A, class B, class C>  
B func(A n, C m) { B value; };

* Определенная с помощью шаблона функция может иметь любое количество непараметризованных формальных параметров. Может быть непараметризованно и возвращаемое функцией значение. Например, в следующей программе шаблон определяет семейство функций, каждая из которых подсчитывает количество нулевых элементов одномерного массива параметризованного типа:

template<class D> long count0(int, D \*); //Прототип шаблона

int main(void) {

int A[] = { 1, 0, 6, 0, 4, 10 };

int n = sizeof(A) / sizeof A[0];

cout << "\ncount0(n,A) = " << count0(n, A);

float X[] = { 10.0, 0.0, 3.3, 0.0, 2.1 };

n = sizeof(X) / sizeof X[];

cout << "\ncount0(n,X) = " << count0(n, X);

return 0; }

* В списке параметров прототипа шаблона имена параметров не обязаны совпадать с именами тех же параметров в определении шаблона.
* При конкретизации шаблонного определения функции необходимо, чтобы при вызове функции типы фактических параметров, соответствующие одинаково параметризованным формальным параметрам, были одинаковыми. Для определенного ниже шаблона функций с данным прототипом недопустимо использовать такое обращение к функции:

template<class E> void swap(E, E);  
int n = 4;   
double d = 4.3;   
swap(n, d); // Ошибка в типах параметров  
swap(double(n), d); // Правильные типы параметров

Для правильного обращения к такой функции требуется явное приведение типа одного из параметров.

* При использовании шаблонов функций возможна перегрузка как шаблонов, так и функций. Могут быть шаблоны с одинаковыми именами, но разными параметрами. Или с помощью шаблона может создаваться функция с таким же именем, что и явно определенная функция. В обоих случаях "распознавание" конкретного вызова выполняется по сигнатуре, т.е. по типам, порядку и количеству фактических параметров.

6. Как записывать параметр шаблона?

* Каждому из разных типов параметров должно предшествовать ключевое слово class. *Пример:* template<class type1, class type2>
* Все параметры шаблона функций должны быть обязательно использованы в спецификациях параметров определения функции.
* Недопустимо использовать в заголовке шаблона параметры с одинаковыми именами, то есть ошибочен такой заголовок:  
  template<class t, class t, class t>

7. Можно ли перегружать параметризованные функции?

Да. Шаблон функции может перегружать функции, отличные от шаблона, с тем же именем. В этом сценарии компилятор сначала пытается разрешить вызов функции с помощью вычета аргументов шаблона для создания экземпляра шаблона функции с уникальной специализацией.

*Пример*

template <typename T>

void print(T value) {cout << “Шаблон” << value;}

template <>

void print(int value) {cout << “Специализация для int” << value;}

void print(int value) {cout << “Перегрузка для int” << value;}

8. Перечислите основные свойства параметризованных классов.

* В определении класса, входящего шаблон, имя класса является не именем отдельного класса, а параметризованным именем семейства 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.
* 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.
* Локальные классы не могут содержать шаблоны в качестве своих элементов.

9. Все ли компонентные функции параметризованного класса являются

параметризованными?

* Компонентные функции параметризованного класса автоматически являются параметризованными. Их не обязательно объявлять как параметризованные с помощью template.

10. Являются ли дружественные функции, описанные в параметризованном классе, параметризованными?

В рамках параметризованного класса нельзя определить friend-шаблоны (дружественные параметризованные классы).

11. Могут ли шаблоны классов содержать виртуальные компонентные функции?

Шаблоны функций, которые являются членами классов, нельзя описывать как virtual.

12. Как определяются компонентные функции параметризованных классов вне определения шаблона класса?

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){ . . . }

13. Что такое инстанцирование шаблона?

Инстанцирование шаблона – это генерация кода функции или класса по шаблону для конкретных параметров.

Неявное инстанцирование:

template <typename T>

T sqrt(T x) {return x\*x;}

Если в программе вызов функции осуществляется как sqrt(-1), то компилятор формирует определение функции int sqrt(int x)

Явное инстанцирование:

template <typename T>

T sqrt(T x) {return x\*x;}

template double sqrt(double);

template int sqrt(int);

Через явное инстанцирование будут доступны только те типы, которые были явно инстанцированы ключевым словом template.

14. На каком этапе происходит генерирование определения класса по шаблону?

Компилятор производит генерацию кода по шаблону, когда происходит явное или неявное инстанцирование. Это может существенно увеличить время компиляции и компоновки, особенно в тех случаях, когда шаблон инстанцируется с одинаковыми параметрами во многих модулях. С помощью extern теперь можно указать компилятору не инстанцировать шаблон в данной единице трансляции (файле)

extern template vector<int>;

Естественно, в одном из модулей должно быть сделано явное или неявное инстанцирование, чтобы при компоновке программы все ссылки были разрешены.

template vector<int>;