Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

направление подготовки: 09.03.01 - «Информатика и вычислительная техника»

**О Т Ч Е Т**

**по лабораторной работе “Классы” №7**

**по дисциплине**

**«Основы алгоритмизации и программирования» семестр 2**

Выполнил студент гр. ИВТ-21-1б

Ипатов Дмитрий Сергеевич

Проверил:

Ст. Преподаватель кафедры ИТАС

Яруллин Д.В.

(оценка) (подпись)

г. Пермь-2022

**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.

**Анализ задачи:**

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.

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.

**Код программы:**

#include <iostream>

using namespace std;

template <class T>

class List

{

protected:

int size;

T\* data;

public:

List(int s, T k);

List(const List<T>& p); #include <iostream>

using namespace std;

template <class T>

class List

{

protected:

int size;

T\* data;

public:

List(int s, T k);

List(const List<T>& p);

~List();

T& operator[](int index);

List operator+(const T k);

List operator+(const List<T>& p);

List& operator=(const List<T>& p);

int& operator()();

friend ostream& operator<< <>(ostream& out, const List<T>& p);

friend istream& operator>> <>(istream& in, List<T>& p);

};

template <class T>

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

{

size = s;

data = new T[size];

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

data[i] = k;

}

template <class T>

List<T>::List(const List<T>& p)

{

size = p.size;

data = new T[size];

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

data[i] = p.data[i];

}

template <class T>

List<T>::~List()

{

delete[]data;

data = 0;

}

template <class T>

List<T>& List<T>::operator=(const List<T>& p)

{

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

size = p.size;

if (data != 0) delete[]data;

data = new T[size];

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

data[i] = p.data[i];

return \*this;

}

template <class T>

List<T> List<T>::operator+(const List<T>& p)

{

int l;

if (size < p.size) l = p.size;

else l = size;

T jija(0, 0);

List<T> temp1(l, jija);

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

{

temp1.data[i].min += data[i].min;

temp1.data[i].sec += data[i].sec;

}

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

{

temp1.data[i].min += p.data[i].min;

temp1.data[i].sec += p.data[i].sec;

}

return temp1;

}

template <class T>

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

{

if (index < size) return data[index];

else cout << "ОШИБКА ДОСТУПА! (index>size)";

}

template <class T>

List<T> List<T>::operator+(const T k)

{

List<T> temp(size, k);

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

temp.data[i] = data[i] + k;

return temp;

}

template <class T>

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

{

return size;

}

template <class T>

ostream& operator<<(ostream& out, const List<T>& p)

{

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

out << p.data[i] << " ";

return out;

}

template <class T>

istream& operator>>(istream& in, List<T>& p)

{

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

in >> p.data[i];

return in;

}

class Time

{

public:

int min, sec;

public:

Time(void);

Time(int, int);

Time(const Time&);

~Time(){};

Time& operator=(const Time&);

Time operator+(Time k);

friend ostream& operator<< (ostream& out, const Time&);

friend istream& operator>> (istream& in, Time&);

};

Time::Time(void)

{

min = sec = 0;

}

Time::Time(int min, int sec)

{

this->min = min;

this->sec = sec;

}

Time::Time(const Time& p)

{

this->min = p.min;

this->sec = p.sec;

}

Time& Time::operator=(const Time& p)

{

this->min = p.min;

this->sec = p.sec;

return \*this;

}

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;

}

ostream& operator<< (ostream& out, const Time& p)

{

out << p.min << ":" << p.sec;

return out;

}

istream& operator>> (istream& in, Time& p)

{

cout << "\nMin: "; in >> p.min;

cout << "\nSec: "; in >> p.sec;

if (p.sec > 60)

{

p.min += p.sec / 60;

p.sec = p.sec % 60;

}

return in;

}

int main()

{

setlocale(LC\_ALL, "rus");

int k;

cin >> k;

/\*Time j;

cin >>j;

Time p;

p = j + k;

cout << p;\*/

Time t;

cin >> t;

cout << t << '\n';

List<Time> a(5, t);

cout << a << '\n';

cin >> a;

cout << a;

cout << '\n';

List<Time> c(5, t);

List<Time> b(10, t);

cout << b << '\n';

b = a;

cout << b << '\n';

cout << a[2] << '\n';

cout << "size a: " << a() << '\n';

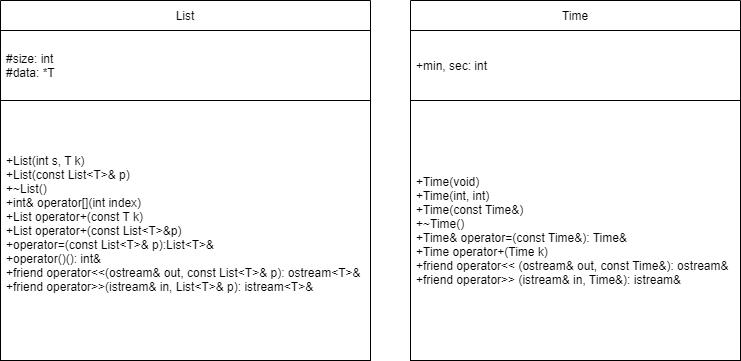
b = a + c;

cout << b << '\n';

return 0;

}

**UML:**

****

**Контрольные вопросы:**

* 1. В чем смысл использования шаблонов?

Шаблоны позволяют обобщать функции и классы для работы с различными типами данных, например если нам потребуется изменить вводимый тип данных для переменной, то во всем коде нам не придется переписывать тип данных там, где используется эта переменная.

* 1. Каковы синтаксис/семантика шаблонов функций?

Для создания шаблона нужно написать ключевое слово template, поставить треугольные скобки, в них написать typename (тип данных) и указать название типа данных, например T.

**Пример с функцией:**

template <typename T>

T abs(T x)

{

if (x<0) return -x;

else return x;

}

* 1. Каковы синтаксис/семантика шаблонов классов?

template <параметры\_шаблона>

class имя\_класса

{…};

**Пример с классом:**

template <class T>

class Point

{

private:

T x,y;

public:

Point(T x, T y)

{

this->x=x;

this->y=y;

}

void Show();

};

template <class T>

Void Point::Show()

{

cout<<”(“<<x<<”,”<<y<<”)”;

}

* 1. Что такое параметры шаблона функции?

**Параметры** **шаблона** **являются** **его** **формальными** **аргументами**, а типы тех аргументов, которые используются в конкретных обращениях к функции, служат фактическими аргументами шаблона.

**Пример:**

template <class T> ; Где T – параметр шаблона.

* 1. Перечислите 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 не параметризованных 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;
  1. Как записывать параметр шаблона?

В треугольных скобках после ключевого слова class или typename (название параметра может быть любое, но последующие не могут повторяться)

* 1. Можно ли перегружать параметризованные функции?

Да, меняя тип данных параметра, либо меняя параметры местами, в том случае, если они разного типа, а также можно выполнять перегрузку функции добавляя или исключая параметры.

* 1. Перечислите основные свойства параметризованных 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;
* Статические переменные шаблонов классов необходимо инициализировать для каждого используемого типа данных.
  1. Все ли компонентные функции параметризованного класса являются параметризованными?

Компонентные функции параметризованного класса автоматически являются параметризованными.

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

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.

* 1. Могут ли шаблоны классов содержать виртуальные компонентные функции?

Шаблоны функций, которые являются членами классов, нельзя описывать какvirtual.

* 1. 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.

* 1. Что такое инстанцирование шаблона?

На каком этапе происходит генерирование определения класса по шаблону?

Процесс порождения функции или класса из шаблона называется инстнацированием