Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Лабораторная работа № 7

Тема: «Шаблоны классов»

Выполнил: студент группы РИС-22-2б

Мизёв В.А. ф

Проверил: доцент кафедры ИТАС

Полякова О.А.

г. Пермь – 2024

*Постановка задачи:*

1. Определить шаблон класса-контейнера.
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
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.

*Задание вариант 1:*

Класс – контейнер ВЕКТОР с элементами типа int

Реализовать операции:

[] – доступ по индексу;

() – определение размера вектора;

+ число – добавляет константу ко всем элементам вектора;

Пользовательский класс Time для работы с временными интервалами. Интервал должен быть представлен в виде двух полей: минуты типа int и секунды типа int. При выводе минуты отделяются от секунд двоеточием.

*Анализ задачи:*

Основываясь на задании необходимо реализовать 2 класса, вектор и время. Класс вектор будет иметь один конструктор по умолчанию, а также соответствующие перегруженные операции из варианта задания. Основные поля класса m\_data – указатель на область памяти, m\_capacity – кол-во выделенной памяти из кучи, m\_size – кол-во элементов в векторе и private функция realloc(), которая позволяет динамически выделять память на базе capacity и копировать данные массива из старой памяти в новую. Класс Time будет иметь поля m\_minutes и m\_seconds, а также метод reset для сброса времени в 0. Для установки и получения минут и секунд реализовать get’еры и set’еры. Также для вектор предусмотреть параметризированный параметр (шаблон).

*Код программы:*

Прикреплён в приложении 1.

*UML-диаграмма класса:*

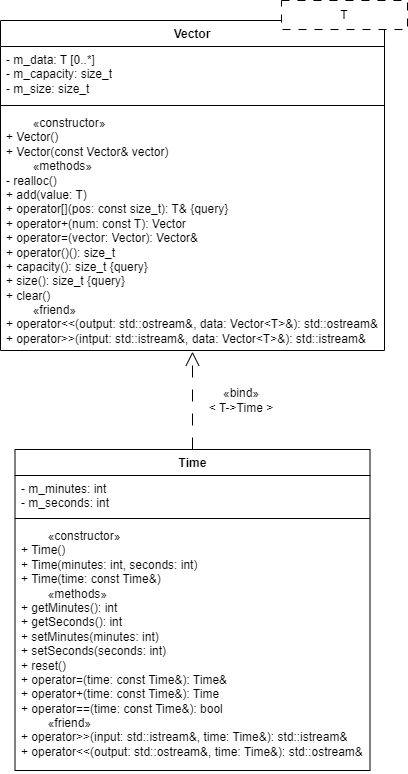


Рисунок 1 - UML-диаграмма классов

*Скриншот работы программы:*

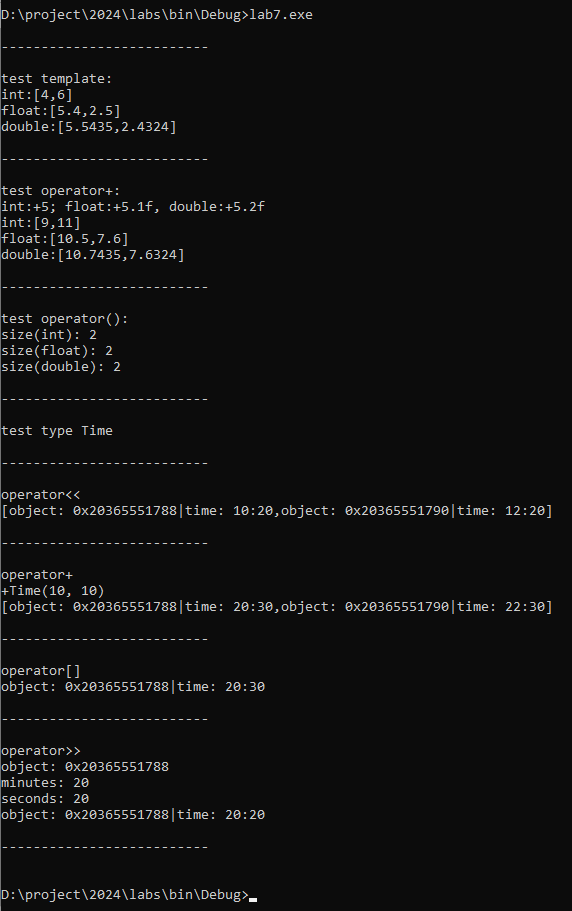


Рисунок 2 – Результат работы программы

*Контрольные вопросы:*

1. В чем смысл использования шаблонов?

Шаблоны позволяют писать обобщённый код, который может работать с различными типами данных без дублирования кода. Это делает код более гибким и расширяемым.

template <typename T>

void print(T value) {

std::cout << value << std::endl;

}

1. Каковы синтаксис/семантика шаблонов функций?

Шаблон функции определяется с помощью ключевого слова template, за которым следует параметр шаблона в угловых скобках, за которым идет определение функции.

template <typename T>

void func(T arg);

1. Каковы синтаксис/семантика шаблонов классов?

Шаблон класса также начинается с ключевого слова template, за которым следуют параметры шаблона в угловых скобках, а затем определяется сам класс.

template <typename T>

class MyClass {};

1. Что такое параметры шаблона функции?

Параметры шаблона функции — это специальные типы или значения, которые используются для определения того, с какими типами данных будет работать шаблон.

1. Перечислите основные свойства параметров шаблона функции.

1. Они могут быть любого типа.

2. Они могут быть определены как типы или значения.

3. Параметры могут иметь несколько типов.

1. Как записывать параметр шаблона?

Параметр шаблона записывается в угловых скобках сразу после ключевого слова template.

template <typename T, int size>

void printArray(T (&arr)[size]) { /\*...\*/ }

1. Можно ли перегружать параметризованные функции?

Да, параметризованные функции можно перегружать, как и обычные функции.

template <typename T>

void func(T arg) {}

template <typename U>

void func(U arg) {}

1. Перечислите основные свойства параметризованных классов.

1. Могут принимать любые типы данных.

2. Позволяют создавать обобщённые структуры данных.

3. Параметры могут использоваться в методах и членах класса.

9. Все ли компонентные функции параметризованного класса

являются параметризованными?

Нет, не все функции должны быть параметризованными. Часто в классе можно иметь обычные методы, не зависящие от параметров шаблона.

10. Являются ли дружественные функции, описанные в параметризованном

классе, параметризованными?

Нет, дружественные функции, объявленные в параметризованном классе, не являются параметризованными автоматически. Им может потребоваться указать параметры шаблона.

11. Могут ли шаблоны классов содержать виртуальные компонентные функции?

Да, в шаблонах классов можно объявлять виртуальные функции, и они могут быть переопределены в наследующих классах.

12. Как определяются компонентные функции параметризованных классов вне

определения шаблона класса?

Функции определяются используя полный путь к классу и параметры шаблона.

template <typename T>

void MyClass<T>::memberFunction() { /\*...\*/ }

13. Что такое инстанцирование шаблона?

Инстанцирование шаблона — это процесс создания конкретной версии шаблона на основе определённых значений или типов параметров.

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

Генерация происходит на этапе компиляции, когда компилятор сталкивается с конкретным использованием шаблона с заданными параметрами.

**ПРИЛОЖЕНИЕ 1**

lab7.cpp

//-------------------------------------

#include <iostream>

#include "lab7\_vector.hpp"

#include "lab7\_time.hpp"

//-------------------------------------

#define IDENT\_PRINT printf("\n--------------------------\n\n")

//-------------------------------------

int main() {

Vector<int> dataInt;

Vector<float> dataFloat;

Vector<double> dataDouble;

dataInt.add(4);

dataInt.add(6);

dataFloat.add(5.4f);

dataFloat.add(2.5f);

dataDouble.add(5.5435f);

dataDouble.add(2.4324f);

IDENT\_PRINT;

std::cout << "test template:" << std::endl;

std::cout << "int:";

std::cout << dataInt << std::endl;

std::cout << "float:";

std::cout << dataFloat << std::endl;

std::cout << "double:";

std::cout << dataDouble << std::endl;

IDENT\_PRINT;

std::cout << "test operator+:" << std::endl;

std::cout << "int:+5; float:+5.1f, double:+5.2f" << std::endl;

dataInt = dataInt + 5;

dataFloat = dataFloat + 5.1f;

dataDouble = dataDouble + 5.2f;

std::cout << "int:";

std::cout << dataInt << std::endl;

std::cout << "float:";

std::cout << dataFloat << std::endl;

std::cout << "double:";

std::cout << dataDouble << std::endl;

IDENT\_PRINT;

std::cout << "test operator():" << std::endl;

std::cout << "size(int): " << dataInt() << std::endl;

std::cout << "size(float): " << dataFloat() << std::endl;

std::cout << "size(double): " << dataDouble() << std::endl;

IDENT\_PRINT;

Vector<Time> dataTime;

dataTime.add(Time(10, 20));

dataTime.add(Time(12, 20));

std::cout << "test type Time\n";

IDENT\_PRINT;

std::cout << "operator<<" << std::endl;

std::cout << dataTime << std::endl;

IDENT\_PRINT;

std::cout << "operator+" << std::endl;

std::cout << "+Time(10, 10)" << std::endl;

dataTime = dataTime + Time(10, 10);

std::cout << dataTime << std::endl;

IDENT\_PRINT;

std::cout << "operator[]" << std::endl;

std::cout << dataTime[0] << std::endl;

IDENT\_PRINT;

std::cout << "operator>>" << std::endl;

std::cin >> dataTime[0];

std::cout << dataTime[0] << std::endl;

IDENT\_PRINT;

return 0;

}

//-------------------------------------

lab7\_vector.hpp

//-------------------------------------

#ifndef LAB7\_VECTOR\_HPP\_INCLUDED

#define LAB7\_VECTOR\_HPP\_INCLUDED

//-------------------------------------

#include <iostream>

#include <cstring>

#include <exception>

//-------------------------------------

#define BASE\_SIZE 4

//-------------------------------------

template <class T>

class Vector {

public:

Vector();

Vector(const Vector& vector);

~Vector();

void add(T value);

T& operator[](const size\_t pos) const;

Vector operator+(const T num);

Vector& operator=(Vector vector);

size\_t operator()(){ return m\_size; }

size\_t capacity() const { return m\_capacity; }

size\_t size() const { return m\_size; }

void clear();

template <class Output>

friend std::ostream& operator<<(std::ostream& output, Vector<Output>& data);

template <class Input>

friend std::istream& operator>>(std::istream& input, Vector<Input>& data);

private:

T\* m\_data;

size\_t m\_capacity,

m\_size;

void realloc();

};

//-------------------------------------

template <class T>

Vector<T>::Vector() {

m\_size = 0;

m\_capacity = 0;

this->realloc();

}

//-------------------------------------

template <class T>

Vector<T>::Vector(const Vector<T>& vector) : Vector() {

for(size\_t i = 0; i < vector.size(); ++i)

this->add(vector[i]);

}

//-------------------------------------

template <class T>

Vector<T>::~Vector() {

this->clear();

}

//-------------------------------------

template <class T>

void Vector<T>::realloc() {

if(m\_size < m\_capacity)

return;

if(m\_capacity == 0) {

m\_capacity = BASE\_SIZE;

m\_data = new T[m\_capacity];

return;

}

m\_capacity\*=2;

T\* array = new T[m\_capacity];

for(size\_t i = 0; i < m\_size; ++i)

array[i] = m\_data[i];

//memcpy(array, m\_data, m\_size\*sizeof(T));

delete []m\_data;

m\_data = array;

}

//-------------------------------------

template <class T>

void Vector<T>::add(T value) {

this->realloc();

m\_data[m\_size] = value;

m\_size++;

}

//-------------------------------------

template <class T>

T& Vector<T>::operator[](const size\_t pos) const {

if(m\_size < pos)

throw std::range\_error("m\_size < pos");

return m\_data[pos];

}

//-------------------------------------

template <class T>

Vector<T> Vector<T>::operator+(const T num) {

Vector<T> vector;

for(size\_t i = 0; i < this->size(); ++i)

vector.add((\*this)[i]+num);

return vector;

}

//-------------------------------------

template <class T>

Vector<T>& Vector<T>::operator=(Vector vector) {

if(&vector == this)

return \*this;

this->clear();

for(size\_t i = 0; i < vector.size(); ++i)

this->add(vector[i]);

return \*this;

}

//-------------------------------------

template <class T>

void Vector<T>::clear() {

if(m\_data != nullptr)

delete []m\_data;

m\_size = 0;

m\_capacity = 0;

}

//-------------------------------------

template <class T>

std::ostream& operator<<(std::ostream& output, Vector<T>& data) {

output << "[";

for(size\_t i = 0; i < data.size(); ++i) {

output << data[i];

if(data.size()-1 != i)

output << ",";

}

output << "]";

return output;

}

//-------------------------------------

template <class T>

std::istream& operator>>(std::istream& input, Vector<T>& data) {

std::cout << "add " << data.size() << " elements: ";

int a;

input >> a;

data.add(a);

return input;

}

//-------------------------------------

#endif // LAB7\_VECTOR\_HPP\_INCLUDED

//-------------------------------------

lab7\_time.hpp

//-------------------------------------

#ifndef LAB3\_TIME\_HPP\_INCLUDED

#define LAB3\_TIME\_HPP\_INCLUDED

//-------------------------------------

#include <iostream>

#include <cstdint>

//-------------------------------------

#define MAX\_SECONDS 60

//-------------------------------------

class Time {

public:

Time();

Time(int minutes, int seconds);

Time(const Time& time);

~Time();

int getMinutes() { return m\_minutes; }

int getSeconds() { return m\_seconds; }

void setMinutes(int minutes) { m\_minutes = minutes; }

void setSeconds(int seconds) { m\_seconds = seconds % MAX\_SECONDS; }

void reset() { m\_minutes = 0; m\_seconds = 0; }

Time& operator=(const Time& time);

Time operator+(const Time& time);

bool operator==(const Time& time) const;

friend std::istream& operator>>(std::istream& input, Time& time);

friend std::ostream& operator<<(std::ostream& output, Time& time);

private:

int m\_minutes,

m\_seconds;

};

//-------------------------------------

#endif // LAB3\_TIME\_HPP\_INCLUDED

//-------------------------------------

lab7\_time.cpp

//-------------------------------------

#include "lab3\_time.hpp"

//-------------------------------------

Time::Time() {

m\_minutes = 0;

m\_seconds = 0;

}

//-------------------------------------

Time::Time(int minutes, int seconds) {

this->setMinutes(minutes);

this->setSeconds(seconds);

}

//-------------------------------------

Time::Time(const Time& time) {

m\_minutes = time.m\_minutes;

m\_seconds = time.m\_seconds;

}

//-------------------------------------

Time::~Time() {

//

}

//-------------------------------------

Time& Time::operator=(const Time& time) {

if(this == &time)

return \*this;

m\_minutes = time.m\_minutes;

m\_seconds = time.m\_seconds;

return \*this;

}

//-------------------------------------

Time Time::operator+(const Time& time) {

uint32\_t newtime = time.m\_minutes\*MAX\_SECONDS + time.m\_seconds;

newtime += (m\_minutes\*MAX\_SECONDS + m\_seconds);

return Time(newtime/MAX\_SECONDS, newtime%MAX\_SECONDS);

}

//-------------------------------------

bool Time::operator==(const Time& time) const {

return time.m\_minutes == m\_minutes && time.m\_seconds == m\_seconds;

}

//-------------------------------------

std::istream& operator>>(std::istream& input, Time& time) {

#ifdef debug\_v

std::cout << "object: " << &time << std::endl;

#endif //debug\_v

std::cout << "minutes: ";

input >> time.m\_minutes;

std::cout << "seconds: ";

input >> time.m\_seconds;

return input;

}

//-------------------------------------

std::ostream& operator<<(std::ostream& output, Time& time) {

return (output <<

#ifdef debug\_v

"object: " << &time << "|" <<

#endif // debug\_v

"time: " << time.getMinutes() << ":" << (int)time.getSeconds());

}

//-------------------------------------