Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

Лабораторная работа №7
«Шаблоны классов»

Выполнил:
студент первого курса
ЭТФ группы РИС-23-3б
Акбашева Софья Руслановна

Проверила:
Доцент кафедры ИТАС О. А. Полякова

Пермь, 2024

Шаблоны классов

Цель задания

1) Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.

2) Реализация шаблона класса-контейнера.

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.

Задание

Класс- контейнер СПИСОК с ключевыми значениями типа int. Реализовать операции: [] – доступа по индексу;

int() – определение размера списка;

* вектор – умножение элементов списков a[i]*b[i];

Пользовательский класс Pair (пара чисел). Пара должна быть представлено двумя полями: типа int для первого числа и типа double для второго. Первое число при выводе на экран должно быть отделено от второго числа двоеточием.

Анализ задачи

1) Необходимо реализовать класс Pair. В классе надо прописать конструкторы: по умолчанию, с параметрами и копирования; а также деструктор, для предотвращения утечки памяти.

2) В классе Pair необходимо прописать операторы перегрузки для необходимых операций.

3) Для реализации класса List необходимо создать структуру Node. При этом структура должна быть шаблонной, что бы она могла хранить разные типы данных.

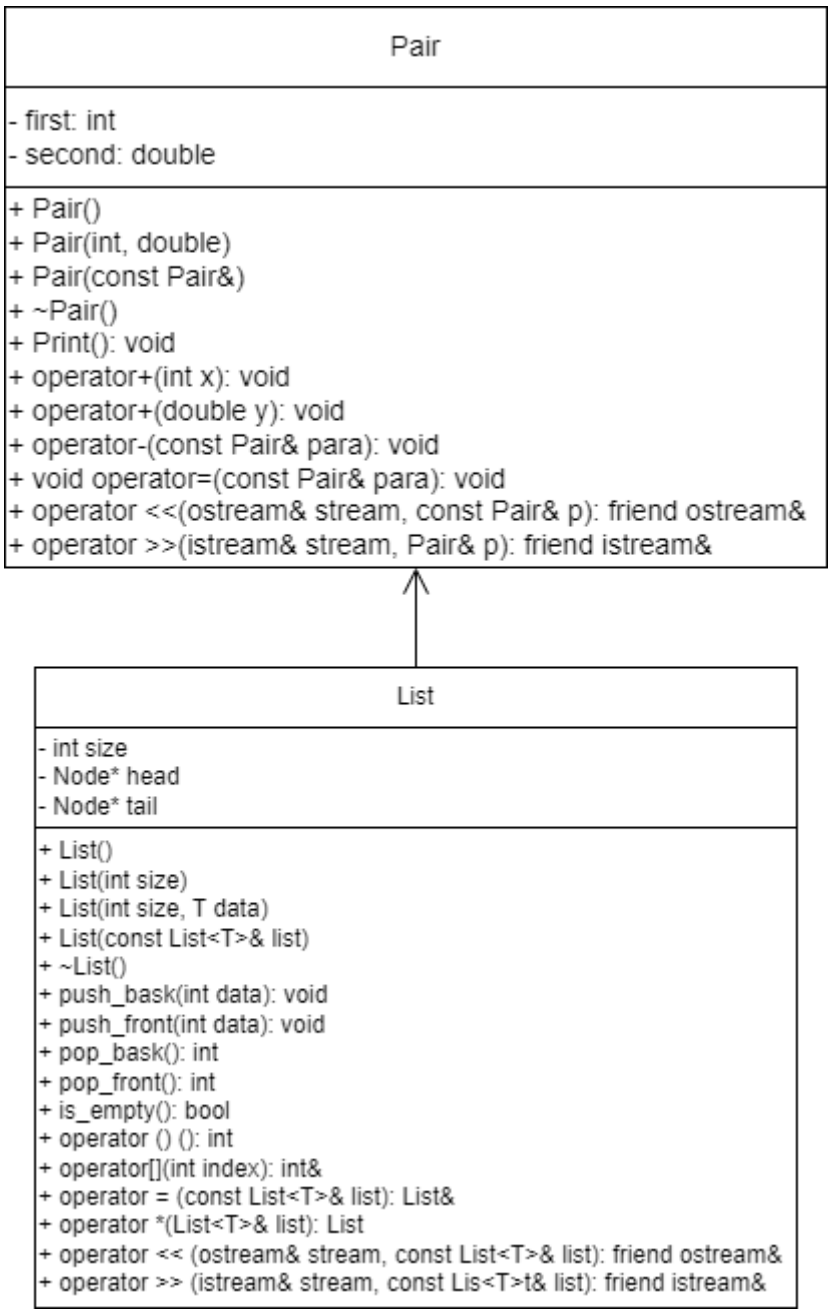
```
template <typename T>
struct Node {
	T data; //данные
	Node* ptr_to_prev_node = nullptr; //указатель на предыдущий элемент
	Node* ptr_to_next_node = nullptr; //указатель на следующий элемент
};
```

4) Для класса List добавить определение методов параметризированного класса.

5) Класс также включает в себя конструктор для создания списка с заданным размером и данными, копирования списка и деструкторы для очистки памяти.

6) В основной функции необходимо продемонстрировать возможности шаблонного класса.

UML диаграмма



Код

Файл Pair.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include <iostream> //стандартные потоки ввода и вывода
using namespace std;

class Pair {
	int first;
	double second;

public:
	Pair() {
		this->first = 0;
		this->second = 0;
	}

	Pair(int first, double second) {
```

```
		this->first = first;
		this->second = second;
	}

	Pair(const Pair& other) {//конструктор копирования
		this->first = other.first;
		this->second = other.second;
	}

	~Pair() {}

	void Print() {//вывод параметров класса person
		cout << "(" << this->first << " : " << this->second << ")" << endl;
	}

	Pair operator+(const int& x) {
		return (Pair(this->first + x, this->second));
	}

	Pair operator+(const double& y) {
		return (Pair(this->first, this->second + y));
	}

	Pair operator-(const Pair& para) {
		return (Pair(this->first - para.first, this->second - para.second));
	}

	void operator=(const Pair& para) {
		this->first = para.first;
		this->second = para.second;
	}

	friend ostream& operator <<(ostream& stream, const Pair& p) {
		stream << "(" << p.first << ":" << p.second << ")";
		return stream;
	}

	friend istream& operator >>(istream& stream, Pair& p) {
		cout << "Введите число first и second через пробел: ";
		stream >> p.first >> p.second;
		return stream;
	}
};
```

Файл List.h

```
#pragma once //предотвращает повторную загрузку заголовочного файла, если он уже был включен
#include <iostream> //стандартные потоки ввода и вывода
using namespace std;

template <typename T>
struct Node {
	T data; //данные
	Node* ptr_to_prev_node = nullptr; //указатель на предыдущий элемент
	Node* ptr_to_next_node = nullptr; //указатель на следующий элемент
};

template <typename T>
class List {
private:
	int size; //размер
	Node<T>* head; //указатель на голову
```

```
	Node<T>* tail; //указатель на хвост
public:
	List() {
		this->size = 0;
		this->head = nullptr; //указатель на голову
		this->tail = nullptr; //указатель на хвост

	};
	List(int size) {//конструктор с длиной
		this->size = size;
		if (size > 0) {//если длина больше 0
			Node<T>* node = new Node<T>;//создаю узел
			this->head = node; //указатель на голову
			this->tail = node; //указатель на хвост
			for (int i = 1; i < size; i++) {//заполняю со второго номера
				Node<T>* New_Node = new Node<T>;//новый узел
				tail->ptr_to_next_node = New_Node;//хвост указывает на новый узел
				New_Node->ptr_to_prev_node = tail;//новый узел ук-т на хвост как на пред-й
				tail = New_Node;//новый узел становится хвостом
			}
			tail->ptr_to_next_node = nullptr;//до хвоста элементов нет
		}
		else {
			this->head = nullptr; //указатель на голову
			this->tail = nullptr; //указатель на хвост
		}
	}

	List(int size, T data) {//конструктор с длиной и данными
		this->size = size;
		if (size > 0) {//если длина больше 0
			Node<T>* node = new Node<T>;//создаю узел
			node->data = data; //данные для узла
			this->head = node; //указатель на голову
			this->tail = node; //указатель на хвост
			for (int i = 1; i < size; i++) {//заполняю со второго номера
				Node* New_Node = new Node<T>;//новый узел
				New_Node->data = data; //данные для нового узла
				tail->ptr_to_next_node = New_Node;//хвост указывает на новый узел
				New_Node->ptr_to_prev_node = tail;//новый узел ук-т на хвост как на пред-й
				tail = New_Node;//новый узел становится хвостом
			}
			tail->ptr_to_next_node = nullptr;//до хвоста элементов нет
		}
		else {
			this->head = nullptr; //указатель на голову
			this->tail = nullptr; //указатель на хвост
		}
	}

	List(const List<T>& list) {//конструктор копирования
		this->head = nullptr; //указатель на голову
		this->tail = nullptr; //указатель на хвост
		this->size = 0;

		Node<T>* current_node = list.head;//создаю узел
		while (current_node != nullptr) {//пока не пройду весь список
			push_bask(current_node->data);//добавляю в конец эл-т
			current_node = current_node->ptr_to_next_node;//беру следующий элемент
		}
	}
```

```
~List() {//деструктор
	Node<T>* current_node = head;//голова
	while (current_node != nullptr) {//пока не пройду весь список
		Node<T>* next_node = current_node->ptr_to_next_node;//следующий элемент
		delete current_node;//удаляю текущий элемент
		current_node = next_node;//беру новый элемент
	}
	head = nullptr; //голова пустая
}

void push_bask(T data) {//вставка элемента в конец
	Node<T>* New_Node = new Node<T>;//новый узел
	New_Node->data = data; //данные для нового узла
	New_Node->ptr_to_next_node = nullptr;//новый узел пока никуда не указывает
	if (this->head == nullptr) {//если список пустой
		this->head = New_Node;//новый узел - голова
		this->tail = New_Node;//новый узел - хвост
	}
	else {
		tail->ptr_to_next_node = New_Node;//хвост указывает на узел
		New_Node->ptr_to_prev_node = tail;
		tail = New_Node; ////новый узел - хвост
	}
	this->size++;//увеличиваю длину
}

void push_front(T data) {//вставка элемента в НАЧАЛО
	Node<T>* New_Node = new Node<T>;//новый узел
	New_Node->data = data; //данные для нового узла

	if (this->head == nullptr) {//если список пустой
		this->head = New_Node;//новый узел - голова
		this->tail = New_Node;//новый узел - хвост
	}
	else {
		head->ptr_to_prev_node = New_Node;//голова указывает на узел
		New_Node->ptr_to_next_node = head;
		head = New_Node; //новый узел - голова
	}
	this->size++;//увеличиваю длину
}

T pop_bask() {//удаление последнего
	T temp;
	if (this->tail != nullptr) {//если список не пустой
		Node<T>* current_node = this->tail;//узел предыдущий
		tail = current_node->ptr_to_prev_node;//меняю хвост
		temp = current_node->data;
		tail->ptr_to_next_node = nullptr;
		this->size--;//уменьшаю размер
	}
	return temp;//значение из удаленной переменной
}

T pop_front() {//удаление первого
	T temp;
	if (this->tail != nullptr) {//если список не пустой
		Node<T>* current_node = this->head;//узел предыдущий
		head = current_node->ptr_to_next_node;//меняю хвост
		temp = current_node->data;
		head->ptr_to_prev_node = nullptr;
		this->size--;//уменьшаю размер
```

```
		}
		return temp;//значение из удаленной переменной
}

bool is_empty() {
		return this->size == 0;//если пустой, то вернет 1 иначе 0
}

int operator () () {//размер списка
		return this->size;
}

T& operator[](int index) {
		if (index < this->size && index >= 0) {//если индекс меньше размера и больше -
				Node<T>* current_node = this->head;
				for (int i = 0; i != index; i++) {
						current_node = current_node->ptr_to_next_node;//беру следующий элемент
				}
				return current_node->data;
		}
		else {
				cout << "Индекс вне цикла";
				exit(0);
		}
}

List& operator = (const List<T>& list) {
		if (this == &list) {//если элементы и так уже равны
				return *this;
		}
		while (head != nullptr) {//сначала очищаю список полностью
				Node<T>* temp = head;
				head = head->ptr_to_next_node;
				delete temp;
		}
		size = 0;

		Node<T>* current_node = list.head;//голова
		while (current_node != nullptr) {//пока не рпойду весь спиок
				push_bask(current_node->data);//добавляю очередной элемент в список
				current_node = current_node->ptr_to_next_node;//беру следующий элемент
		}
		return *this;
}

List operator *(List<T>& list) {
		int temp_size;
		if (this->size > list.size) {//если первый размер списка больше размера 2 списка
				temp_size = list.size;//наименьший размер из двух
		}
		else {
				temp_size = this->size;//наименьший размер из двух
		}

		List temp(temp_size, 0);//список с наименьшей длиной
		for (int i = 0; i < temp_size; i++) {//пока не пройду по списку наименьшей длины
				temp[i] = (*this)[i] * list[i];
		}
		return temp;
}

friend ostream& operator <<(ostream& stream, const List<T>& list) {//вывод
```

```
                cout << "Вывод элементов списка:" << endl;
                Node<T>* current_node = list.head;
                while (current_node != nullptr) {//пока не пройду весь список
                        stream << current_node->data << ' ';
                        current_node = current_node->ptr_to_next_node;//беру следующий элемент
                }
                cout << endl;
                return stream;
        }

        friend istream& operator >>(istream& stream, List<T>& list) {//ввод
                cout << "Введите элементы списка" << endl;
                Node<T>* current_node = list.head;
                while (current_node != nullptr) {//пока не пройду весь список
                        cin >> current_node->data;
                        current_node = current_node->ptr_to_next_node;//беру следующий элемент
                }
                return stream;
        }
};
```

Файл Class_7.cpp

```
#include <iostream>
#include "Pair.h"
#include "List.h"

using namespace std;

int main(){
   system("chcp 1251 > Null");
   srand(time(0));

   List<int> list_1(5);
   List<double> list_2(5);
   List<float> list_3(5);
   List<Pair> list_4(3);
   List<int> list_5(5);

   cout << "list_1 - int" << endl;
   cin >> list_1;
   cout << list_1;
   cout << "Вывод элемента под номером 2" << endl;
   cout << list_1[1] << endl;

   cout << endl << "list_2 - double" << endl;
   cin >> list_2;
   cout << list_2;
   cout << "Вывод элемента под номером 3" << endl;
   cout << list_2[2] << endl;

   cout << endl << "list_3 - float" << endl;
   cin >> list_3;
   cout << list_3;
   cout << "Вывод элемента под номером 4" << endl;
   cout << list_3[3] << endl;

   cout << endl << "list_4" << endl;
   cin >> list_4;
   cout << list_4;
   cout << "Вывод элемента под номером 1" << endl;
   cout << list_4[0] << endl;
```

```
    cout << endl << "list_5 - int" << endl;
    list_5 = list_1;
    cout << list_5;
}
```

Результаты работы

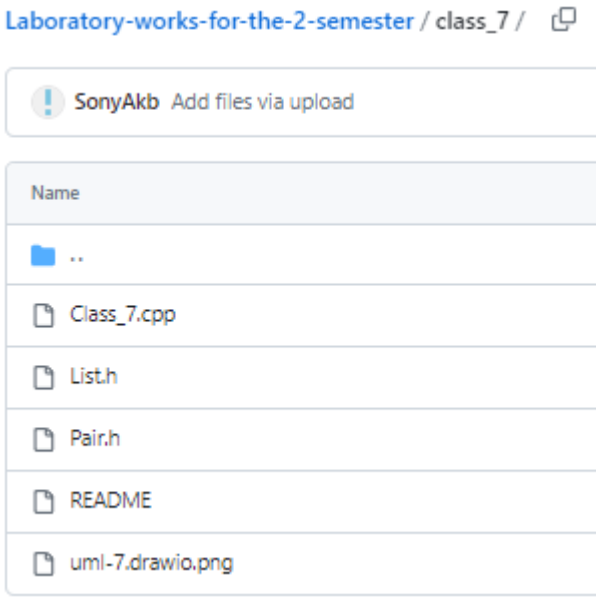
```
list_1 - int
Введите элементы списка
12
43
8765
0
9
Вывод элементов списка:
12 43 8765 0 9
Вывод элемента под номером 2
43

list_2 - double
Введите элементы списка
45.6
987.236
4.5
71.41
68.5
Вывод элементов списка:
45.6 987.236 4.5 71.41 68.5
Вывод элемента под номером 3
4.5

list_3 - float
Введите элементы списка
4.7
9.5
0.78
364.2
7
Вывод элементов списка:
4.7 9.5 0.78 364.2 7
Вывод элемента под номером 4
364.2

list_4
Введите элементы списка
Введите число first и second через пробел: 78 98.7
Введите число first и second через пробел: 123 45.77
Введите число first и second через пробел: 56 4862.01
Вывод элементов списка:
(78:98.7) (123:45.77) (56:4862.01)
Вывод элемента под номером 1
(78:98.7)

list_5 - int
Вывод элементов списка:
12 43 8765 0 9
```

Вывод

В ходе работы я применила знания о работе с шаблонами классов. С помощью шаблона классов я отделила алгоритм от конкретных типов данных, передавая тип в качестве параметра. С помощью шаблонов было автоматизировано создание функций, обрабатывающих разнотипные данные. В ходе работы были реализованы класс контейнер – список, конструкторы, деструктор, операции ввода-вывода, операцию присваивания. Также был создан пользовательский класс Pair, который содержит первое число типа int и второе число типа double. В основной функции была продемонстрирована работа класса List с разными типами данных, в том числе и с пользовательским классом.

GitHub

Ссылка: https://github.com/SonyAkb/Laboratory-works-for-the-2-semester/tree/main/class_7



Контрольные вопросы

1. В чем смысл использования шаблонов?

Шаблоны вводятся для того, чтобы автоматизировать создание функций, обрабатывающих разнотипные данные.

2. Каковы синтаксис/семантика шаблонов функций?

template<параметры_шаблона>

заголовок функции

{тело функции}

Пример:

```
template<typename type>
type abs(type 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;
public:
Point(T x = 0, T y = 0):x(x), y(y){}
void Show();
{;
```

4. Что такое параметры шаблона функции?

Параметры шаблона функции — это типы или значения, которые определяют конкретную версию функции. Они указываются внутри угловых скобок после ключевого слова template перед именем функции.

5. Перечислите основные свойства параметров шаблона функции.

- шаблоны не могут быть виртуальными

- шаблоны могут содержать статические элементы, дружественные функции и классы

- шаблоны могут быть производными как от шаблонов, так и от обычных классов, а также являться базовыми и для шаблонов, и для обычных классов.

6. Как записывать параметр шаблона?

Параметр шаблона записывается внутри угловых скобок после ключевого слова template, например <int>, <double>, <T>, где T - это псевдоним типа, который будет определен позже.

7. Можно ли перегружать параметризованные функции?

Да, можно перегружать параметризованные функции. Перегрузка осуществляется так же, как и для обычных функций, с учетом параметров шаблона.

8. Перечислите основные свойства параметризованных классов.

- Параметры шаблона могут быть как значениями, так и типами. Например, можно создать шаблон класса с параметром, который задает размер массива.

- Параметры шаблона могут быть любыми типами данных, включая примитивные типы, пользовательские типы и другие шаблоны классов.

-Шаблонные классы могут быть унаследованы от других шаблонных классов и могут иметь шаблонные функции-члены.

-Параметры шаблона могут использоваться внутри класса для объявления переменных, функций и типов.

-Каждый конкретный экземпляр параметризованного класса является отдельным типом, который определяется параметрами шаблона.

9. Все ли компонентные функции параметризованного класса являются параметризованными?

Да, все компонентные функции (методы) параметризированного класса являются параметризованными, то есть они используют те же параметры шаблона, что и сам класс.

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

Да

11. Могут ли шаблоны классов содержать виртуальные компонентные функции?

Да, шаблоны классов могут содержать виртуальные компонентные функции. Виртуальность работает так же, как и для обычных классов, с учетом параметров шаблона.

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

Компонентные функции параметризованных классов могут быть определены вне определения шаблона класса с помощью использования template<class T> перед каждым определением функции, где T - это параметр шаблона.

13. Что такое инстанцирование шаблона?

Инстанцирование шаблона — это процесс генерации конкретной версии шаблона для определенных типов. Оно происходит автоматически, когда компилятор встречает использование шаблона с конкретными типами.

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

Генерирование определения класса по шаблону происходит во время компиляции, когда компилятор встречает использование шаблона класса с конкретными типами.