Федеральное агентство связи

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М. А. БОНЧ-БРУЕВИЧА» (СПбГУТ)

Факультет информационных технологий и программной инженерии Кафедра: Программная инженерия. Разработка программного обеспечения и приложений искусственного интеллекта в киберфизических системах

ЛАБОРАТОРНАЯ РАБОТА №5

по дисциплине **«Объектно-ориентированное программирование»**

Тема: Шаблоны классов. Работа с исключительными ситуациями языка C++

Выполнил: студент 2-го курса группы ИКПИ-42 Терещенко Максим Андреевич
Преподаватель: Петрова Ольга Борисовна

Санкт-Петербург 2025

Постановка задачи

В рамках лабораторной работы необходимо решить две задачи, связанные с шаблонами классов в C++:

1. **Преобразование числового класса в шаблон.** Взять числовой класс, разработанный в первой лабораторной работе класс `Complex`, и преобразовать его в шаблонный класс.
2. **Разработка шаблона контейнера.** Разработать шаблон класса для стека, построенного на основе массива с динамически выделяемой памятью (вариант №4).

Таблица идентификаторов

Идентификатор	Описание
Complex()	Конструктор по умолчанию класса `Complex<T>`, инициализирует нулевые значения (0, 0).
Complex(T r, T i)	Конструктор, задающий действительную (`r`) и мнимую (`i`) части комплексного числа.
Complex(const Complex& other)	Конструктор копирования класса `Complex<T>`.
operator+(const Complex& other)	Перегруженный оператор сложения для комплексных чисел.
operator-(const Complex& other)	Перегруженный оператор вычитания для комплексных чисел.
operator*(const Complex& other)	Перегруженный оператор умножения для комплексных чисел.
operator/(const Complex& other)	Перегруженный оператор деления с обработкой деления на ноль (`std::runtime_error`).
print()	Метод для вывода комплексного числа в формате `(real, imag)`.
Stack(size_t initCapacity = 10)	Конструктор класса `Stack<T>` с начальной ёмкостью (по умолчанию — 10). Если `initCapacity == 0`, `data = nullptr`.
~Stack()	Деструктор класса `Stack<T>`, освобождающий динамическую память (`delete[] data`).
Stack(const Stack& other)	Конструктор копирования класса `Stack<T>`.
operator=(const Stack& other)	Перегруженный оператор присваивания (правило трёх).
push(const T& value)	Добавляет элемент в вершину стека. При переполнении вызывает `resize()`.
pop()	Удаляет верхний элемент. При пустом стеке выбрасывает `std::underflow_error`.
top()	Возвращает ссылку на верхний элемент (неконстантная версия).

Идентификатор	Описание
top() const	Возвращает константную ссылку на верхний элемент.
empty() const	Проверяет, пуст ли стек (`size == 0`).
getSize() const	Возвращает текущий размер стека (`size`).
resize()	Внутренний метод: увеличивает ёмкость по формуле `capacity * 2 + 1` , копирует данные.
print() const	Выводит содержимое стека от основания к вершине: `[ 0 10 ... ] (top is right)` .

Анализ задачи

Алгоритмическая часть

1. **Шаблон класса** `Complex<T>` :
   - Реализован шаблонный класс для работы с комплексными числами.
   - Поддерживаются арифметические операции: сложение, вычитание, умножение, деление.
   - Для деления реализована обработка исключительной ситуации (деление на ноль).
   - Вывод результата осуществляется с помощью метода `print` .
2. **Шаблон класса** `Stack<T>` :
   - Реализован как динамический массив с автоматическим расширением.
   - **Начальная ёмкость**: задаётся в конструкторе (в тесте — `5`).
   - **Стратегия роста**: `newCapacity = capacity * 2 + 1` .
   - Поддерживаемые операции:
     - `push(const T&)` — добавление элемента.
     - `pop()` — удаление верхнего элемента.
     - `top()` / `top() const` — доступ к верхнему элементу.
     - `empty()` , `getSize()` — проверка пустоты и размера.
     - `print()` — вывод содержимого (от основания к вершине, справа — верхний элемент).
   - **Исключения**:
     - `std::underflow_error` при `pop()` или `top()` на пустом стеке.
   - **Управление памятью**:
     - Выделение через `new T[]` , освобождение через `delete[]` .

Формулы

Для класса `Complex<T>` используются следующие формулы для арифметических операций:

1. **Сложение**:

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$

2. **Вычитание**:

$$(a + bi) - (c + di) = (a - c) + (b - d)i$$

3. **Умножение**:

$$(a + bi) \cdot (c + di) = (ac - bd) + (ad + bc)i$$

4. **Деление**:

$$\frac{a + bi}{c + di} = \frac{(ac + bd) + (bc - ad)i}{c^2 + d^2}, \quad c^2 + d^2 \neq 0$$

Если знаменатель $c^2 + d^2 = 0$, выбрасывается исключение `std::runtime_error` .

Контрольный расчет

Пример расчета для комплексных чисел:

1. Сложение:
   $ci1 = (3, 4)$, $ci2 = (1, -2)$
   $ci1 + ci2 = (3 + 1) + (4 + (-2))i = (4, 2)$
   Результат в программе: $(4, 2)$
2. Умножение:
   $cd1 = (2.5, 1.5)$, $cd2 = (1.0, -0.5)$

   $cd1 \cdot cd2 = (2.5 \cdot 1.0 - 1.5 \cdot (-0.5)) + (2.5 \cdot (-0.5) + 1.5 \cdot 1.0)i = (2.5 + 0.75) + (-1.25 + 1.5)i = (3.25, 0.25)$

   Результат в программе: $(3.25, 0.25)$

Для `Stack<T>` :

- Создан стек `stk` типа `int` с начальной ёмкостью **5**.
- Добавлены элементы: `0, 10, 20, 30, 40` → размер = 5 (не переполнен).
- Добавлены ещё: `50, 60` → при `push(50)` размер = 5 → переполнение → вызов `resize()` → **новая ёмкость**: $5 \times 2 + 1 = 11$.
- После всех `push` :
  [0 10 20 30 40 50 60] (top is right)
- `top()` → `60`
- `pop()` → удалён `60` , новый `top()` → `50`
- Полное опустошение → попытка `pop()` из пустого стека → **исключение**: `std::underflow_error("Stack is empty")`

Копии экранов выполнения

Вывод программы:

```
(base) PS D:\Educational activities\Proga\university_labs\2_course\OOP\lab5> ./lab5
=== Test Complex<T> ===
ci1 + ci2 = (4, 2)
cd1 * cd2 = (3.25, 0.25)

=== Test Stack<T> ===
Stack after pushes: [ 0 10 20 30 40 50 60 ] (top is right)
Top: 60
After pop, top: 50
Exception: Stack is empty
```

Список тестов, которые проводили

Список тестов, которые проводили

№	Тест	Описание	Ожидаемый результат
1	Complex: сложение	"(3,4) + (1,-2)"	"(4,2)"
2	Complex: умножение	"(2.5,1.5) * (1.0,-0.5)"	"(3.25,0.25)"
3	Stack: создание	Stack stk(5)	ёмкость = 5, размер = 0
4	Stack: добавление 7 элементов	push(0..60)	размер = 7, ёмкость = 11
5	Stack: top() после добавления	—	60
6	Stack: pop()	удалить верхний	top() → 50
7	Stack: полное опустошение	цикл while (!empty()) pop()	стек пуст
8	Stack: pop() из пустого	—	std::underflow_error
9	Stack: print()	—	[0 10 ... 60] (top is right)

Выводы

Результаты выполнения лабораторной работы

Шаблонный класс Complex

- Успешно преобразован класс `Complex` в шаблонный `Complex<T>`, поддерживающий арифметику для типов `int` и `double`.

Шаблонный класс Stack

- Реализован шаблонный класс `Stack<T>`, построенный на основе динамического массива:
  - Автоматическое расширение по формуле `capacity * 2 + 1`.
  - Реализованы принципы "правила трёх" (копирование, присваивание, деструктор).

- Обработка исключений при операциях над пустым стеком с использованием `std::underflow_error`.

Проведённое комплексное тестирование:

- **Корректность LIFO** — проверка работы стека по принципу "последний пришёл, первый ушёл".
- **Расширение ёмкости** — тестирование динамического расширения контейнера.
- **Устойчивость к ошибкам** — проверка поведения при операциях с пустым стеком.
- **Вывод содержимого** — тестирование правильности вывода элементов стека.

Приложение:

main.cpp

```
#include "Complex.h"
#include "Array.h"
#include <iostream>
using namespace std;

int main() {
    cout << "=== Test Complex<T> ===" << endl;
    Complex<int> ci1(3, 4), ci2(1, -2);
    Complex<int> ci_sum = ci1 + ci2;
    cout << "ci1 + ci2 = "; ci_sum.print(); cout << endl;

    Complex<double> cd1(2.5, 1.5), cd2(1.0, -0.5);
    Complex<double> cd_mul = cd1 * cd2;
    cout << "cd1 * cd2 = "; cd_mul.print(); cout << endl;

    cout << "\n=== Test Array<T> ===" << endl;
    Array<int> arr(5);
    for (size_t i = 0; i < arr.getSize(); i++) {
        arr[i] = i * 10;
    }
    cout << "Array int: ";
    arr.print();
    cout << endl;

    arr.resize(7);
    cout << "After resize(7): ";
    arr.print();
    cout << endl;

    try {
        cout << "arr[10] = " << arr[10] << endl;
    } catch (const exception& e) {
        cout << "Exception: " << e.what() << endl;
    }

    return 0;
}
```

Complex.h

```cpp
#ifndef COMPLEX_H
#define COMPLEX_H

#include <iostream>
#include <stdexcept>

template <typename T>
class Complex {
private:
    T real;
    T imag;
public:
    Complex(T r = 0, T i = 0) : real(r), imag(i) {}

    Complex<T> operator+(const Complex<T>& other) const {
        return Complex<T>(real + other.real, imag + other.imag);
    }

    Complex<T> operator-(const Complex<T>& other) const {
        return Complex<T>(real - other.real, imag - other.imag);
    }

    Complex<T> operator*(const Complex<T>& other) const {
        return Complex<T>(real * other.real - imag * other.imag,
                          real * other.imag + imag * other.real);
    }

    Complex<T> operator/(const Complex<T>& other) const {
        T denom = other.real * other.real + other.imag * other.imag;
        if (denom == 0) throw std::runtime_error("Division by zero in Complex");
        return Complex<T>((real * other.real + imag * other.imag) / denom,
                          (imag * other.real - real * other.imag) / denom);
    }

    void print() const {
        std::cout << "(" << real << ", " << imag << ")";
    }
};

#endif
```

Array.h:

```
#ifndef ARRAY_H
#define ARRAY_H

#include <iostream>
#include <stdexcept>

template <typename T>
class Array {
private:
    T* data;
    size_t size;
public:
    Array(size_t n = 0) : size(n) {
        if (n == 0) {
            data = nullptr;
        } else {
            data = new T[n];
        }
    }

    ~Array() {
        delete[] data;
    }

    Array(const Array<T>& other) {
        size = other.size;
        data = new T[size];
        for (size_t i = 0; i < size; i++) {
            data[i] = other.data[i];
        }
    }

    Array<T>& operator=(const Array<T>& other) {
        if (this == &other) return *this;
        delete[] data;
        size = other.size;
        data = new T[size];
        for (size_t i = 0; i < size; i++) {
            data[i] = other.data[i];
        }
        return *this;
    }

    T& operator<a href="size_t index" target="_blank" rel="noopener noreferrer nofollow"></a> {
        if (index >= size) throw std::out_of_range("Index out of range");
        return data[index];
    }

    const T& operator<a href="size_t index" target="_blank" rel="noopener noreferrer nofollow"></a> const {
        if (index >= size) throw std::out_of_range("Index out of range");
        return data[index];
    }
```

```
    size_t getSize() const {
        return size;
    }

    void resize(size_t newSize) {
        T* newData = new T[newSize];
        size_t minSize = (newSize < size) ? newSize : size;
        for (size_t i = 0; i < minSize; i++) {
            newData[i] = data[i];
        }
        delete[] data;
        data = newData;
        size = newSize;
    }

    void print() const {
        std::cout << "[ ";
        for (size_t i = 0; i < size; i++) {
            std::cout << data[i] << " ";
        }
        std::cout << "]";
    }
};

#endif
```