

# Методические указания для СРС по выполнению лабораторных работ по дисциплине Объектно-ориентированное программирование

Тема: **Шаблоны, исключительные ситуации**

1. Определить класс-шаблон с использованием динамического распределения памяти согласно варианту и необходимые конструкторы и операции, включая конструктор копий, операция присваивания и если указано операцию индексации. При выходе за границу, переполнении и т.п. вызвать исключительную ситуацию (определить собственные классы) для информирования программы, вызвавшей метод.

## **Варианты заданий**

### Вариант 1

класс вектор (одномерный массив элементов заданного типа и размера, указанного в аргументах конструктора), получение i-го элемента с помощью операции индексации

### Вариант 2

класс матрица (двумерный массив элементов заданного типа, размеры задаются в аргументах конструктора), получение (i,j)-го элемента с помощью операции (), получение строки как матрицы

Остальные варианты <https://ipc.susu.ru/20768-5.html>

2. Реализовать main с тестами

(создание объектов и выполнение действий с ними, в т.ч. действие, приводящее к возникновению исключительной ситуации, которую необходимо перехватить)

3. Написать отчет

- Постановка задачи
- Описание интерфейса класса (class { } и комментарии ко всем полям, методам и функциям)
- Описание тестов для проверки классов (main с комментариями, какие действия выполнялись, полученные результаты)
- Листинг реализации классов (реализация методов и функций)

4. Отправить отчет

## **Пример решения задания 4**

Определить класс-шаблон с использованием динамического распределения памяти согласно варианту и необходимые конструкторы и операции, включая конструктор копий, операция присваивания. При выходе за границу, переполнении и т.п. вызвать исключительную ситуацию (определить собственные классы) для информирования программы, вызвавшей метод.

Шаблонный класс стек элементов заданного типа, размером не более указанного в параметрах конструктора, добавление << и извлечение >> элемента.

Классы для исключений - наследование

```

struct stackerror { // базовый класс для ошибок
    virtual ~stackerror() {} // деструктор
    virtual const char *what() const=0; // сообщение для печати
};
struct stackempty: stackerror {
    const char *what() const {return "Стек пуст";} // сообщение для печати
};
struct stackfull: stackerror {
    const char *what() const {return "Стек полон";} // сообщение для печати
};
// пример порождения
throw stackempty();

```

Класс для исключений - номер ошибки

```

struct stackerror {
public:
    enum Error { empty, full }; // виды ошибок
    stackerror(Error e):e(e){} // конструктор
    const char *what() const; // сообщение для печати
private:
    Error e;
};
const char *stackerror::what() const {
    switch(e) {
        case empty: return "Стек пуст";
        case full: return "Стек полон";
    }
};
// пример порождения
throw stackerror(stackerror::empty);

```

Класс стек

```

template <typename T>
class Stack {
    T *a; // указатель на данные в стеке
    int col, // текущее количество
        size; // максимальный размер
public:
    // конструктор
    Stack(int size): col(0), size(size),a(new T[size]) {}
    // конструктор копий
    Stack(const Stack<T> &c);
    // деструктор
    ~Stack(){delete []a;}
    // операция присваивания
    Stack<T> &operator=(const Stack<T> &c);
    // операция добавления в стек
    Stack<T> &operator<<(const T &x);
    // операция извлечения из стека
    Stack<T> &operator>>(T &x);
};
template <typename T>
Stack<T>::Stack(const Stack<T> &c):a(new T[c.size]),col(c.col),size(c.size) {
    for (int i=0; i<col; i++)
        a[i]=c.a[i];
}
template <typename T>
Stack<T> &Stack<T>::operator<<(const T &x) {
    if (col==size) throw stackfull();
    a[col++]=x;
    return *this;
}

```

```

}
template <typename T>
Stack<T> &Stack<T>::operator>>(T &x) {
    if (col==0) throw stackempty();
    x=a[--col];
    return *this;
}
template <typename T>
Stack<T> & Stack<T>::operator=(const Stack<T> &c) {
    if(c.col>size) throw stackfull();
    col=c.col;
    for (int i=0; i<col; i++)
        a[i]=c.a[i];
    return *this;
}

```

Пример проверки, задача - вызвать все методы и продемонстрировать работу исключений

```

int main() {
    Stack<int> obj(10);
    try {
        for (int i=0;; i++) {
            obj<<i;
        }
    }
    catch (stackerror &e) {
        cout<<e.what()<<"\n";
    }
    try {
        while (1) {
            int temp;
            obj>>temp;
            cout<<temp;
        }
    }
    catch (stackerror &e) {
        cout<<"\n";
        cout<<e.what()<<"\n";
    }
    return 0;
}

```

Южно-Уральский государственный университет (НИУ)  
Институт естественных и точных наук  
Кафедра «Прикладная математика и программирование»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4  
по дисциплине «Объектно-ориентированное программирование»

Автор работы  
студент группы ЕТ-212  
\_\_\_\_\_ А.А.Александрова  
\_\_\_\_\_ 2019 г.

Работа зачтена с оценкой  
\_\_\_\_\_  
\_\_\_\_\_ А.К.Демидов  
\_\_\_\_\_ 2019 г.

Челябинск, 2019

## 1 Постановка задачи

I. Определить класс-шаблон с использованием динамического распределения памяти согласно варианту и необходимые конструкторы и операции, включая конструктор копий, операция присваивания и если указано операцию индексации. При выходе за границу, переполнении и т.п. вызвать исключительную ситуацию (определить собственные классы) для информирования программы, вызвавшей метод.

13. Класс стек элементов заданного типа, размером не более указанного в параметрах конструктора,  
добавление << и извлечение >> элемента

II. Реализовать main с тестами (создание объектов и выполнение действий с ними, в т.ч. действие, приводящее к возникновению исключительной ситуации, которую необходимо перехватить)

## 2 Описание интерфейса класса

```
struct stackerror { // базовый класс для ошибок
    virtual ~stackerror() {} // деструктор
    virtual const char *what() const=0; // сообщение для печати
};
struct stackempty: stackerror {
    const char *what() const {return "Стек пуст";} // сообщение для печати
};
struct stackfull: stackerror {
    const char *what() const {return "Стек полон";} // сообщение для печати
};
template <typename T>
class Stack {
    T *a; // указатель на данные в стеке
    int col, // текущее количество
        size; // максимальный размер
public:
    // конструктор
    Stack(int size): col(0), size(size),a(new T[size]) {}
    // конструктор копий
    Stack(const Stack<T> &);
    // деструктор
    ~Stack() throw() {delete []a;}
    // операция присваивания
    Stack<T> &operator=(const Stack<T> &);
    // операция добавления в стек
    Stack<T> &operator<<(const T &);
    // операция извлечения из стека
    Stack<T> &operator>>(T &);
};
```

### 3 Описание тестов для проверки классов

```
int main()
{
    Stack<int> obj(10);
    cout<<"Тест 1. Добавление\n";
    try {
        for (int i=0;; i++) {
            obj<<i;
        }
    }
    catch (stackerror &e) {
        cout<<e.what();
    }
    cout<<"\nТест 2. Извлечение\nИзвлекаем из стека:\n";
    try {
        while (1) {
            int temp;
            obj>>temp;
            cout<<temp;
        }
    }
    catch (stackerror &e) {
        cout<<"\n"<<e.what();
    }
    cout<<"\nТест 3. Присваивание";
    ...
    return 0;
}
```

#### Полученные результаты:

```
Тест 1. Добавление
Стек полон
Тест 2. Извлечение
Извлекаем из стека:
987654321
Стек пуст
Тест 3. Присваивание
...
```

## 4 Листинг реализации класса

```
template <typename T>
Stack<T>:: Stack(const Stack <T> &c):a(new T[c.size]),col(c.col),size(c.size) {
    for (int i=0; i<col; i++)
        a[i]=c.a[i];
}
template <typename T>
Stack<T> &Stack<T> ::operator<<(const T &x) {
    if (col==size) throw stackfull();
    a[col++]=x;
    return *this;
}
template <typename T>
Stack <T> &Stack<T>::operator>>(T &x) {
    if (col==0) throw stackempty();
    x=a[--col];
    return *this;
}
...
```