Программа 3. Разработка и реализация сложного класса на языке С++.

Условие задачи

1. Разработать класс "стек" в соответствии со следующим заданием.

Состояние класса -

Стек отображается в памяти машины вектором. Память под стек выделяется статически, во время компиляции, и задаётся массивом некоторого предопределённого фиксированного размера. Элементами стека являются целые числа.

Предусмотреть следующие возможности:

- пустой конструктор для инициализации экземпляров и массивов экземпляров класса по умолчанию;
- вывод содержимого стека в выходной поток;
- занесение в стек нового элемента;
- выборка элемента из стека;
- проверка стека на пустоту;
- копирование данных из одного стека в другой;
- получение текущего размера стека;
- получение максимального размера стека.

2. Проектирование класса рекомендуется начать с представления состояния класса, учитывающего заданные операции, а затем реализации конструкторов и метода вывода. Для отладки и исчерпывающего тестирования других методов разработанного класса реализовать диалоговую программу, которая позволяет вводить параметры отлаживаемых методов. Для обработки ошибочных ситуаций использовать механизм исключений.

3. Повторить разработку класса, реализовав отдельные методы (там, где это оправдано), перегруженными операторами.

4. Ещё раз повторить разработку класса при условии, что память под массив необходимой длины выделяется динамически, во время выполнения программы (с помощью оператора new; память задаётся указателем на тип элемента вектора в состоянии класса).

Дополнить интерфейс класса следующими возможностями:

- создание экземпляра класса с его инициализацией другим экземпляром класса (копирующий конструктор);
- переопределение экземпляра класса (с помощью перегруженного оператора присваивания).

Особенности реализации

Указанное задание целесообразно разбить на три этапа; каждый вариант следует разрабатывать, отлаживать и защищать автономно. При переходе от одного варианта к другому целесообразно копировать исходные файлы и вносить в копии файлов необходимые изменения. В представленном примере приведена не диалоговая отладочная программа.

Текст программы для каждого из 3-х этапов программы представлен в виде совокупности следующих файлов:

— файл stack.h— для определения класса,

— файл stack.cpp— для реализации методов класса,

— файл main.cpp— для реализации отладочной программы,

— файл CMakeLists.txt— для сборки проекта.

Для собрания разных этапов в один проект используется один общий файл CMakeLists.txt:

```
# установка версии CMake
cmake_minimum_required(VERSION 3.16)

# название проекта (обязательно)
project(oopprog3)

# установка стандарта языка - C++20
set(CMAKE_CXX_STANDARD 20)

# установка флагов компилятора: CMAKE_CXX_FLAGS += -Wall -Wextra
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wall -Wextra")

# добавление подпроектов
add_subdirectory(static)
add_subdirectory(operators)
add_subdirectory(dynamic)

# для сборки из консоли:
#
# mkdir build   # создание директории для файлов сборки
# cd build      # переход в директорию сборки
# cmake ..      # генерация файлов сборки на основе CMakeLists.txt
# make          # сборка проекта
```

Статическая память

Рассмотрим особенности разработки первого этапа (в соответствии с пунктом 1).

Так как память под стек выделяется статически (на этапе компиляции), при занесении (включении) нового элемента в стек следует проверять ситуацию переполнения вектора.

Исходный текст

Файл stack.h

```
#ifndef OOPPROG3_STACK_H
#define OOPPROG3_STACK_H

#include <iostream>

namespace Prog3 {

    class Stack {
    private:
        static const int SZ = 10; // максимальный размер вектора
        int top; // индекс вершины стека; первый элемент вектора, доступный для записи в стек
        int ar[SZ];
    public:
        Stack() : top(0) {}

        void push(int); // занесение в стек
        int pop(); // выборка из стека
        bool empty() const { return top == 0; } // проверка стека на пустоту
        void copy(const Stack &); // копирование данных из одного стека в другой
        int getSize() const { return top; } // текущий размер стека
        int getMaxSize() const { return SZ; } // максимальный размер стека
        std::ostream &print(std::ostream &) const;
    };

} // Stack

#endif //OOPPROG3_STACK_H
```

Файл stack.cpp

```cpp
#include "stack.h"

namespace Prog3 {
    // выборка из стека; генерируется исключение, если стек пуст
    int Stack::pop() {
        if(top == 0)
            throw std::logic_error("Stack is empty");
        int el = ar[--top];
        return el;
    }

    // запись в стек; генерируется исключение, если стек полон
    void Stack::push(int el) {
        if(top >= SZ) {
            throw std::runtime_error("Stack overflow!");
        }
        ar[top++] = el;
    }

    // вывод содержимого стека в поток (в порядке, соответствующем выборке из стека)
    std::ostream &Stack::print(std::ostream &s) const {
        if(top == 0)
            s << "Stack is empty";
        else
            for(int i = top - 1; i >= 0; --i)
                s << ar[i] << ' ';
        s << std::endl;
        return s;
    }

    // Копирование данных из одного стека в другой.
    // Данные из стека-операнда записываются в стек-адресат
    // Состояние стека-операнда не изменяется.
    void Stack::copy(const Stack &st) {
        int new_top = top + st.top;
        if(new_top >= SZ) {
            throw std::runtime_error("Stack overflow!");
        }
        std::copy(st.ar, st.ar + st.top, ar + top);
        top = new_top;
    }
}
```

Файл main.cpp

```cpp
#include "stack.h"

using namespace Prog3;

int main() {
    Stack st;
    std::cout << "1. Push into stack: max size = " << st.getMaxSize() << std::endl;
    try {
        for(int i = 0; i < 12; ++i)
            st.push(12 + i * 10);
    }
    catch(const std::exception &msg) {
        std::cout << msg.what() << std::endl;
    }
    std::cout << "Current size = " << st.getSize() << "; In stack : \n";
    st.print(std::cout);
    std::cout << std::endl << std::endl;
    {
        Stack newst(st), tmpst;
        std::cout << "2. New block:\n in new stack: \n";
        newst.print(std::cout);
        std::cout << std::endl;
        std::cout << "3. Push into local tmp stack: \n";
        for(int i = 0; i < 8; ++i)
            tmpst.push(25 + i * 10);
        std::cout << "In tmp stack: \n";
        tmpst.print(std::cout);
        std::cout << std::endl << std::endl;
        std::cout << "4. Copying from new stack into tmp stack: \n";
        try {
            tmpst.copy(newst);
        }
        catch(const std::exception &msg) {
            std::cout << msg.what() << std::endl;
        }
        std::cout << "In new stack: \n";
        newst.print(std::cout);
        std::cout << std::endl;
        std::cout << "In tmp stack: \n";
        tmpst.print(std::cout);
        std::cout << std::endl << std::endl;
        std::cout << "5. Remove from new stack: \n";
        while(!newst.empty())
            std::cout << newst.pop() << ' ';
        std::cout << std::endl;
        std::cout << "In new stack: \n";
        newst.print(std::cout);
        std::cout << std::endl << std::endl;
    }
    std::cout << "6. The first stack in the first block: \n";
    st.print(std::cout);
    std::cout << std::endl;
    return 0;
}
```

Файл CMakeLists.txt

```
# создание библиотеки stack_static
add_library(stack_static stack.h stack.cpp)

# создание тестирующей программы
add_executable(main_static main.cpp)

# подключение библиотеки к тестирующей программе
target_link_libraries(main_static stack_static)
```

Перегрузка операторов

Рассмотрим особенности разработки второго варианта (в соответствии с пунктом 3). Здесь, прежде всего, необходимо определить, какие перегруженные операторы целесообразно использовать.

Для операций занесения в стек и выборки из стека использовать перегрузку операторов нецелесообразно, так как в языке нет операторов, реализующих семантику данных операций.

При копировании данных из одного стека в другой, стек, из которого считываются данные, должен сохранить своё исходное состояние. Такую семантику можно реализовать с помощью перегрузки оператора +=, для которого второй операнд также представляет собой стек.

Данный оператор можно было бы использовать и для занесения элемента в стек, если была бы возможность выполнять преобразование типа: тип int в тип Stack. Такое преобразование возможно, если в классе есть соответствующий конструктор с одним аргументом типа int. Однако использование для занесения в стек элемента перегруженного оператора += менее эффективно, так как потребует лишнее преобразование типа (вызов конструктора) и последующее уничтожение созданного временного экземпляра стека (вызов деструктора). Тем не менее, определим дополнительный инициализирующий конструктор, который будет создавать стек, содержащий один элемент (заданный аргументом конструктора).

Для вывода содержимого стека в поток целесообразно использовать перегрузку оператора <<.

Исходный текст

Файл stack.h

```
#ifndef OOPPROG3_STACK_H
#define OOPPROG3_STACK_H

#include <iostream>

namespace Prog3 {

    class Stack {
    private:
        static const int SZ = 10; // максимальный размер вектора
        int top; // индекс вершины стека; первый элемент вектора, доступный для записи в стек
        int ar[SZ];
    public:
        Stack() : top(0) {}

        Stack(int el) : top(1) { ar[0] = el; }

        void push(int); // занесение в стек
        int pop(); // выборка из стека
        bool empty() const { return top == 0; } // проверка стека на пустоту
        // преобразование стека в тип bool - true если стек не пустой, false если пустой
        operator bool() const { return !empty(); }

        Stack &operator+=(const Stack &); // копирование данных из одного стека в другой
        int getSize() const { return top; } // текущий размер стека
        int getMaxSize() const { return SZ; } // максимальный размер стека
        friend std::ostream &operator<<(std::ostream &, const Stack &);
    };

} // Stack
#endif //OOPPROG3_STACK_H
```

Файл stack.cpp

```cpp
#include "stack.h"

namespace Prog3 {
    // выборка из стека; генерируется исключение, если стек пуст
    int Stack::pop() {
        if(top == 0)
            throw std::logic_error("Stack is empty");
        int el = ar[--top];
        return el;
    }

    // запись в стек; генерируется исключение, если стек полон
    void Stack::push(int el) {
        if(top >= SZ) {
            throw std::runtime_error("Stack overflow!");
        }
        ar[top++] = el;
    }


    // Копирование данных из одного стека в другой.
    // Данные из стека-операнда записываются в стек-адресат
    // Состояние стека-операнда не изменяется.
    Stack &Stack::operator+=(const Stack &st) {
        int new_top = top + st.top;
        if(new_top >= SZ) {
            throw std::runtime_error("Stack overflow!");
        }
        std::copy(st.ar, st.ar + st.top, ar + top);
        top = new_top;
        return *this;
    }

    // вывод содержимого стека в поток (в порядке, соответствующем выборке из стека)
    std::ostream &operator<<(std::ostream &s, const Stack &st) {
        if(st.top == 0)
            s << "Stack is empty";
        else
            for(int i = st.top - 1; i >= 0; --i)
                s << st.ar[i] << ' ';
        s << std::endl;
        return s;
    }
}
```

Файл main.cpp

```cpp
#include "stack.h"

using namespace Prog3;

int main() {
    Stack st;
    std::cout << "1. Push into stack: max size = " << st.getMaxSize() << std::endl;
    try {
        for(int i = 0; i < 12; ++i)
            st.push(12 + i * 10);
    }
    catch(const std::exception &msg) {
        std::cout << msg.what() << std::endl;
    }
    std::cout << "Current size = " << st.getSize() << "; In stack : \n";
    {
        Stack newst(st), tmpst;
        std::cout << "2. New block:\n in new stack: \n" << newst << std::endl;
        std::cout << "3. Push into local tmp stack: \n";
        for(int i = 0; i < 8; ++i)
            // перегруженный оператор += и преобразование типа вместо push()
            tmpst += (25 + i * 10);

        std::cout << "In tmp stack: \n" << tmpst << std::endl << std::endl;
        std::cout << "4. Copying from new stack into tmp stack: \n";
        try {
            tmpst += newst;
        }
        catch(const std::exception &msg) {
            std::cout << msg.what() << std::endl;
        }
        std::cout << "In new stack: \n" << newst << std::endl;
        std::cout << "In tmp stack: \n" << tmpst << std::endl << std::endl;
        std::cout << "5. Remove from new stack: \n";
        while(newst)
            std::cout << newst.pop() << ' ';
        std::cout << std::endl;
        std::cout << "In new stack: \n" << newst << std::endl << std::endl;
    }
    std::cout << "6. The first stack in the first block: \n" << st << std::endl;
    return 0;
}
```

Файл CMakeLists.txt

```
# создание библиотеки stack_operators
add_library(stack_operators stack.h stack.cpp)

# создание тестирующей программы
add_executable(main_operators main.cpp)

# подключение библиотеки к тестирующей программе
target_link_libraries(main_operators stack_operators)
```

Динамическая память

Рассмотрим особенности разработки очередного этапа (в соответствии с пунктом 4). Здесь, прежде всего, следует обратить внимание на то, что память под стек должна выделяться автоматически, во время выполнения программы. При этом если при записи элемента в стек обнаруживается, что стек полон, память под стек должна быть переопределена, и старое состояние стека не должно исчезнуть.

В связи с этим, в тексты программы, реализованной для этапа 2, нужно включить необходимые изменения и добавления.

1. Прежде всего, следует иметь в виду, что размер вектора, на который будет отображаться стек, может меняться в процессе работы программы: при занесении новых элементов в стек размер вектора может увеличиваться, но при выборке из стека уменьшаться не будет. Следовательно, в состоянии класса должен быть предусмотрен компонент, хранящий размер вектора (или максимальный, на текущий момент, размер стека).

2. Пустой конструктор выделяет под стек массив, размер которого определяется с помощью некоторой предопределённой константы. В дальнейшем, при необходимости увеличить размер стека, память будет увеличиваться не на 1, а на некоторое предопределённое значение. Увеличивать объем памяти под стек, добавляя каждый раз по одному элементу, неэффективно, так как операции занесения в стек могут выполняться достаточно часто.

3. Необходимо изменить инициализирующие конструкторы, в которых должна выделяться память.

4. Необходимо перепрограммировать метод занесения нового элемента в стек, так как в этом методе вероятно увеличение памяти, отведённой под стек. Проверка на переполнение стека не выполняется — в случае нехватки памяти генерируется стандартное исключение std::bad_alloc, которое можно перехватить (в прикладной программе, но не в методах класса) и обработать.

5. Так как в методах копирования и перемещения элементы в стек добавляются с помощью метода занесения в стек, наличие доступной памяти здесь дополнительно можно не проверять.

6. Необходимо добавить деструктор, освобождающий выделенную память.

7. Необходимо добавить копирующий и перемещающий конструкторы, перегруженный и перемещающий операторы присваивания.

8. Методы, в которых не требуется перераспределение памяти, останутся неизменными.

Исходный текст

Файл stack.h

```
#ifndef OOPPROG3_STACK_H
#define OOPPROG3_STACK_H

#include <iostream>

namespace Prog3 {
    class Stack {
    private:
        static const int QUOTA = 10; // размер квоты для выделения памяти
        int SZ; // максимальный размер вектора
        int top;// индекс вершины стека; первый элемент вектора, доступный для записи в стек
        int *ar;
        void resize(int new_size); // изменение размера стека
    public:
        Stack() : SZ(QUOTA), top(0), ar(new int[QUOTA]) {}
        Stack(int el) : SZ(QUOTA), top(1), ar(new int[QUOTA]) { ar[0] = el; }
        Stack(const Stack &); // копирующий конструктор
        Stack(Stack &&) noexcept; // перемещающий конструктор
        ~Stack() { delete[] ar; } // деструктор
        Stack &operator=(const Stack &); // перегруженный оператор присваивания
        Stack &operator=(Stack &&) noexcept; // перемещающий оператор присваивания
        void push(int); // занесение в стек
        int pop(); // выборка из стека
        bool empty() const { return top == 0; } // проверка стека на пустоту
        // преобразование стека в тип bool - true если стек не пустой, false если пустой
        operator bool() const { return !empty(); }
        Stack &operator+=(const Stack &);// копирование данных из одного стека в другой
        int getSize() const { return top; } // текущий размер стека
        int getMaxSize() const { return SZ; } // максимальный размер стека
        friend std::ostream &operator<<(std::ostream &, const Stack &);
    };

} // Stack
#endif //OOPPROG3_STACK_H
```

Файл stack.cpp

```
#include "stack.h"

namespace Prog3 {

    // копирующий конструктор
    Stack::Stack(const Stack &st) : SZ(st.SZ), top(st.top) {
        ar = new int[SZ];
        std::copy(st.ar, st.ar + st.top, ar);
    }

    // перемещающий конструктор
    Stack::Stack(Stack &&st) noexcept: SZ(st.SZ), top(st.top), ar(st.ar) {
        st.ar = nullptr;
    }

    // изменение размера стека
    void Stack::resize(int new_size) {
        if(new_size < top) {
            throw std::runtime_error("New size is too small");
        }
        int *new_ar = new int[new_size];
        std::move(ar, ar + top, new_ar);
        delete[] ar;
        SZ = new_size;
```

```
        ar = new_ar;
    }

    // перегруженный оператор присваивания
    Stack &Stack::operator=(const Stack &st) {
        if(this != &st) {
            // сначала выделяем память и только потом освобождаем старую,
            // чтобы в случае исключения не нарушить согласованность данных
            // (strong exception safety)
            int *new_ar = new int[SZ];
            top = st.top;
            SZ = st.SZ;
            delete[] ar;
            ar = new_ar;
            std::copy(st.ar, st.ar + st.top, ar);
        }
        return *this;
    }

    // перемещающий оператор присваивания
    Stack &Stack::operator=(Stack &&st) noexcept {
        std::swap(top, st.top);
        std::swap(SZ, st.SZ);
        std::swap(ar, st.ar);
        return *this;
    }

    // выборка из стека; генерируется исключение, если стек пуст
    int Stack::pop() {
        if(top == 0)
            throw std::logic_error("Stack is empty");
        int el = ar[--top];
        return el;
    }

    // запись в стек; генерируется исключение, если стек полон
    void Stack::push(int el) {
        if(top >= SZ) {
            resize(SZ + QUOTA);
        }
        ar[top++] = el;
    }

    // Копирование данных из одного стека в другой.
    // Данные из стека-операнда записываются в стек-адресат
    // Состояние стека-операнда не изменяется.
    Stack &Stack::operator+=(const Stack &st) {
        int new_top = top + st.top;
        if(new_top >= SZ) {
            resize(new_top + QUOTA);
        }
        std::copy(st.ar, st.ar + st.top, ar + top);
        top = new_top;
        return *this;
    }

    // вывод содержимого стека в поток (в порядке, соответствующем выборке из стека)
    std::ostream &operator<<(std::ostream &s, const Stack &st) {
        if(st.top == 0)
            s << "Stack is empty";
        else
            for(int i = st.top - 1; i >= 0; --i)
                s << st.ar[i] << ' ';
        s << std::endl;
        return s;
    }
}
```

Файл main.cpp

```cpp
#include "stack.h"

using namespace Prog3;

int main() {
    Stack st;
    std::cout << "1. Push into stack: max size = " << st.getMaxSize() << std::endl;
    for(int i = 0; i < 12; ++i)
        st.push(12 + i * 10);
    std::cout << "Current size = " << st.getSize() << "; In stack : \n";
    {
        Stack newst(st), tmpst;
        std::cout << "2. New block:\n in new stack: \n" << newst << std::endl;
        std::cout << "3. Push into local tmp stack: \n";
        for(int i = 0; i < 8; ++i)
            tmpst += (25 + i * 10); // перегруженный оператор += и преобразование типа
        std::cout << "In tmp stack: \n" << tmpst << std::endl << std::endl;
        std::cout << "4. Copying from new stack into tmp stack: \n";
        tmpst += newst;
        std::cout << "In new stack: \n" << newst << std::endl;
        std::cout << "In tmp stack: \n" << tmpst << std::endl << std::endl;
        std::cout << "5. Remove from new stack: \n";
        while(newst)
            std::cout << newst.pop() << ' ';
        std::cout << std::endl;
        std::cout << "In new stack: \n" << newst << std::endl << std::endl;
    }
    std::cout << "6. The first stack in the first block: \n" << st << std::endl;
    return 0;
}
```

Файл CMakeLists.txt

```
# создание библиотеки stack_operators
add_library(stack_dynamic stack.h stack.cpp)

# создание тестирующей программы
add_executable(main_dynamic main.cpp)

# подключение библиотеки к тестирующей программе
target_link_libraries(main_dynamic stack_dynamic)
```