МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

Национальный исследовательский ядерный университет «МИФИ»

**Институт интеллектуальных кибернетических систем**

**кафедра “Кибернетика”(22)**

“Объектно-Ориентированное Программирование”

**Лабораторная работа по ООП:**

«Лабораторная работа № 3. Обобщённое программирование в C++»

*Выполнил: Локтионов А.А.. гр. М19-514*

*Преподаватель: Шапкин П.А.*

Москва

2020 г.

**Задание 1.** Используя шаблоны реализовать функцию min, которая должна возвращать минимальный из двух элементов для любого сравнимого типа:

template <class T>

T min(T value1, T value2){

return value1 > value2 ? value2 : value1;

}

**Задание 2.** Реализовать функцию min из предыдущего задания, воспользовавшись макросом:

#define min(a,b) ((a)<(b)?(a):(b))

**Задание 3.** С использованием шаблонов разработать класс Stack, представляющий из себя реализацию структуры данных — стек:

Реализация класса должна содержать следующие методы (где тип T является параметризованным типом для класса Stack):

• bool Stack::empty() — показывает пустой стек или нет.

• void Stack::push(const T &item) — добавляет элемент в стек.

• T &Stack::top() — возвращает ссылку на последний добавленный элемент.

• void Stack::pop() — извлекает последний добавленный элемент из стека.

template <class T>

class Elem {

public:

Elem(T item, Elem<T>\* next) : item(item), next(next) {};

T item;

Elem\* next;

};

template <class T>

class Stack {

public:

Stack() : tail(nullptr), length(0) {}

bool empty() {

return length == 0;

}

void push(const T& item) {

Elem<T>\* newElem = new Elem<T>(item, tail);

tail = newElem;

length++;

}

T& top() {

if (!empty()) {

return tail->item;

}

}

void pop() {

if (!empty()) {

Elem<T>\* result = tail;

tail = result->next;

delete result;

length--;

}

}

friend Stack<T> operator+ <> (const Stack<T>& s1, const Stack<T>& s2);

private:

Elem<T>\* tail;

int length;

};

**Задание 4.** Для класса Stack добавить дружественную (friend) функцию, которая бы реализовывала операцию сложения с использованием оператора + для стека. Поведение оператора должно быть таким, что при записи a + b получался бы новый стек, содержащий сначала элементы стека a со следующими за ними элементами стека b.

template <class T>

Stack<T> operator+ (const Stack<T>& s1, const Stack<T>& s2) { //a+b as s1+s2

Stack<T> st;

Elem<T>\* head = nullptr;

Elem<T>\* pointer = s2.tail;

while (pointer!=nullptr){ //Copy b to new stack

if (head == nullptr) {

st.tail = new Elem<T>(pointer->item, nullptr);

head = st.tail;

pointer = pointer->next;

continue;

}

head->next = new Elem<T>(pointer->item, nullptr);

head = head->next;

pointer = pointer->next;

}

pointer = s1.tail;

while (pointer != nullptr){ //Copy a to new stack

if (head == nullptr) {

st.tail = new Elem<T>(pointer->item, nullptr);

head = st.tail;

pointer = pointer->next;

continue;

}

head->next = new Elem<T>(pointer->item, nullptr);

head = head->next;

pointer = pointer->next;

}

if (head == nullptr) { //both a and b empty

st.tail = nullptr;

}

st.length = s1.length + s2.length;

return st;

}

**Задание 5.** Разработать класс Graph с использованием контейнеров из стандартной библиотеки шаблонов C++, который бы представлял из себя реализацию ориентированного графа. Каждый узел должен быть представлен уникальным числом. Разработанный класс должен содержать реализации следующих методов:

• Graph::Graph(const vector &starts, const vector &ends) — строит ориентированный граф для переданных множеств вершин. Элементы с одинаковым индексом в starts и ends представляют из себя направленное ребро графа.

• int Graph::numOutgoing(const int nodeId) const — возвращает количество достижимых из вершины с nodeId вершин графа.

• const vector &Graph::adjacent(const int nodeId) const — возвращает ссылку на список достижимых из вершины с nodeId вершин графа.

Все методы и конструкторы должны выбрасывать исключение invalid\_argument в случае передачи в параметрах некорректных данных.

class Node {

public:

explicit Node(int id) : id(id) {};

int id;

vector<int> edges;

};

bool operator==(const Node& n1, const Node& n2) {

return n1.id == n2.id ? true : false;

}

class Graph {

public:

Graph(const vector<int>& starts, const vector<int>& ends) {

if (starts.size() != ends.size()) {

throw invalid\_argument("array size differ");

}

for (int i = 0; i < starts.size();i++) {

auto node = Node(starts[i]);

auto it = find(map.begin(), map.end(),node);

if (it == map.end()) {

node.edges.push\_back(ends[i]);

map.push\_back(node);

}

else {

(\*it).edges.push\_back(ends[i]);

}

}

}

int NumOutgoing(const int nodeId) const {

auto node = Node(nodeId);

auto it = find(map.begin(), map.end(), node);

if (it == map.end()) {

throw invalid\_argument("unknown node");

}

else {

return Adjacent(nodeId).size();

}

}

const vector<int>& Adjacent(const int nodeId) const {

static vector<int> result; //костыль из-за интерфейса

result.clear();

auto node = Node(nodeId);

auto it = find(map.begin(), map.end(), node);

if (it == map.end()) {

return result;

}

else {

queue<int> q;

for (auto edge : (\*it).edges) {

q.push(edge);

result.push\_back(edge);

}

while (!q.empty()) {

auto curNode = Node(q.front());

q.pop();

it = find(map.begin(), map.end(), curNode);

if (it != map.end()) {

for (auto edge : (\*it).edges) {

if (find(result.begin(), result.end(), edge) == result.end()) {

q.push(edge);

result.push\_back(edge);

}

}

}

}

return result;

}

}

private:

vector<Node> map;

};