МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

Национальный исследовательский ядерный университет «МИФИ»

Факультет Кибернетики и Информационной безопасности

КАФЕДРА КИБЕРНЕТИКИ (№ 22)

Направление подготовки 09.03.04 Программная инженерия

Объектно-ориентированное программирование

Лабораторная работа 3.
Обобщённое программирование в C++

Группа _____ M20-504

Студент _____ Авраменко А. Д.

Преподователь _____ Шапкин П.А.

Москва 2021

Задание 1

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

Программа 1 компилируется и исполняется, как ожидается. Видно, что функция *min* вызываются и выполняется без ошибок, что очевидно и соответствует ожиданиям.

```
#include <cstring>
#include <iostream>


template<class T>
T min(T a, T b)
{
  if (a < b)
    return a;
  return b;
}


#define MIN(a, b) \
  (a < b ? a : b)


int main()
{

  std::cout << min(1, 2) << " " << min(2, 1) << std::endl;
  std::cout << MIN(1, 2) << " " << MIN(2, 1) << std::endl;

  return 0;
};
```

Листинг 1 – Шаблонная функция *min* и макрос *MIN*

Задание 2

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

Программа 1 компилируется и исполняется, как ожидается. Видно, что макрос *MIN* используется и отрабатывает без ошибок, что очевидно и соответствует ожиданиям.

Задание 3

Задание 3. *С использованием шаблонов разработать класс Stack, представляющий из себя реализацию структуры данных — стек. Проверить программу на предмет отсутствия утечек памяти.*

Программа 2 компилируется и исполняется, как ожидается. Шаблонный класс соответствует требованиям. Утечки памяти отсутствуют, как и ожидается, что подтвердается утилитой *Valgrind* 3 .

```
#include <vector>
#include <iostream>

template<class T>
class Stack
{
public:
  Stack() {}

  Stack(const Stack& ) = default;
  Stack(Stack&& ) = default;

  bool empty()
  {
    return _vec.size() == 0;
  }

  void push(const T& item)
  {
    _vec.push_back(item);
  }

  void push(T&& item)
  {
    _vec.push_back(item);
  }

  T& top()
  {
    return _vec.beck();
  }

  void pop()
  {
    _vec.pop_back();
  }


  friend std::ostream& operator<<(std::ostream& s, Stack<T> b)
  {
```

```
    for (auto i = b._vec.begin(); i < b._vec.end(); ++i)
      s << *i << " ";
    return s;
  }

  friend Stack operator+(Stack<T> a, Stack<T> b)
  {
    Stack result(a);
    for (auto i = b._vec.begin(); i < b._vec.end(); ++i)
      result.push(*i);
    return result;
  }
protected:
  std::vector<T> _vec;
};


int main()
{
  Stack<int> a;
  a.push(1);
  a.push(2);
  a.push(3);
  a.push(4);
  std::cout << a << std::endl;
  a.pop();
  a.pop();
  std::cout << a << std::endl;
  std::cout << (a + a) << std::endl;
}
```

Листинг 2 – Шаблонный класс *Stack*

```
==5425== Memcheck, a memory error detector
==5425== Copyright (C) 2002–2017, and GNU GPL'd, by Julian Seward et al.
==5425== Using Valgrind–3.14.0 and LibVEX; rerun with –h for copyright info
==5425== Command: ./a.out
==5425==
==5425==
==5425== HEAP SUMMARY:
==5425==     in use at exit: 0 bytes in 0 blocks
==5425==   total heap usage: 11 allocs, 11 frees, 73,820 bytes allocated
==5425==
==5425== All heap blocks were freed -- no leaks are possible
==5425==
==5425== For counts of detected and suppressed errors, rerun with: –v
==5425== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Листинг 3 – Анализ *Valgrind*

Задание 4

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.*

Программа 2 компилируется и исполняется, как ожидается. Оператор *operator+* успешно перегружен и соответсвует требованиям. Утечки памяти отсутствуют, как и ожидается, что подтвердается утилитой *Valgrind* 3 .

Задание 5

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.Все методы и конструкторы должны выбрасывать исключение invalid_argument в случае передачи в параметрах некорректных данных.*

Программа 4 компилируется и исполняется, как ожидается. Класс соответствует требованиям. Утечки памяти отсутствуют, как и ожидается, что подтвердается утилитой *Valgrind* 5 .

```
#include <iostream>
#include <vector>
#include <map>
#include <utility>
#include <stdexcept>
#include <set>

class Graph
{
  using vec = std::vector<int>;
public:
  Graph(const vec& a, const vec& b)
  {
    if (a.size() != b.size())
      throw std::invalid_argument("a.size() != b.size()");   //        ,

    std::map<int, std::set<int>> tmp;
    for (auto i = a.begin(), j = b.begin(); i < a.end() && j < b.end(); ++i, ++j) {
      auto val = _m.find(*i);
      if (val != _m.end())
        val->second.push_back(*j);
      else
        _m[*i] = vec({*j});

      val = _m.find(*j);
      if (val == _m.end())
        _m[*j] = vec();
    }

    for (auto i = _m.begin(); i != _m.end(); ++i) {
      std::set<int> tmp_set(i->second.begin(), i->second.end());
      std::set<int> res;
      //std::cout << "out " << i->first << std::endl;
      outgoing(res, tmp_set);
      tmp[i->first] = res;
    }

    _m.clear();
    for (auto i = tmp.begin(); i != tmp.end(); ++i) {
      _m[i->first] = vec(i->second.begin(), i->second.end());
```

```
      _nodes.insert(i->first);
    }
  }

  int numOutgoing(const int node_id) const
  {
    return adjacent(node_id).size();
  }

  const vec& adjacent(const int node_id) const
  {
    if (_m.find(node_id) == _m.end())
      throw std::invalid_argument("node doesnt exist");

    return _m.at(node_id);
  }

  const std::set<int>& nodes() const
  {
    return _nodes;
  }
protected:

  void outgoing(std::set<int>& checked, std::set<int> next)
  {
    for (auto i = next.begin(); i != next.end(); ++i) {
      if (checked.find(*i) == checked.end()) {
        checked.insert(*i);
        std::set<int> tmp_next(_m.at(*i).begin(), _m.at(*i).end());
        //std::cout << "in " << checked.size() << " out " << *i << std::endl;
        outgoing(checked, tmp_next);
      }
    }
  }

  std::set<int> _nodes;
  std::map<int, vec> _m;
};


int main()
{
  std::vector<int> a = {1, 2, 3, 4, 1}, b = {2, 3, 4, 1, 5};

  Graph g(a, b);
  for (auto i = g.nodes().begin(); i != g.nodes().end(); ++i) {
    std::cout << *i << " : ";
    const std::vector<int>& tmp = g.adjacent(*i);
    for (auto j = tmp.begin(); j < tmp.end(); ++j)
      std::cout << *j << " ";
    std::cout << std::endl;
  }

  return 0;
};
```

Листинг 4 – Класс Graph

```
==5416== Memcheck, a memory error detector
==5416== Copyright (C) 2002—2017, and GNU GPL'd, by Julian Seward et al.
==5416== Using Valgrind—3.14.0 and LibVEX; rerun with —h for copyright info
==5416== Command: ./a.out
==5416==
==5416==
==5416== HEAP SUMMARY:
==5416==     in use at exit: 0 bytes in 0 blocks
==5416==   total heap usage: 123 allocs, 123 frees, 78,752 bytes allocated
==5416==
==5416== All heap blocks were freed —— no leaks are possible
==5416==
==5416== For counts of detected and suppressed errors, rerun with: —v
==5416== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Листинг 5 – Анализ *Valgrid*