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 2.
Изучение конструкций ООП в C++

Группа _____ M20-504

Студент _____ Авраменко А. Д.

Преподователь _____ Шапкин П.А.

Москва 2021

Задание 1

Задание 1. *Провести исследование порядка вызова конструкторов/деструкторов при наследовании: составить программу, отображающую порядок выполнения операций при создании и удалении объекта.*

Программа 1 компилируется и исполняется, как ожидается. Результат работы программы приведен в листинге 2 . Видно, что конструкторы вызываются в порядке убывания старшинства, а деструкторы наоборот, что очевидно и соответствует ожиданиям.

```
#include <iostream>


class A
{
protected:
  enum func_type {constructor, destructor, other};
public:
  explicit A(int id = 0) : _id(id)
  {
    print(constructor, "A");
  }


  void print(func_type t, std::string name) const
  {
    std::string tmp;
    switch (t) {
      case constructor:
        tmp = "created";
        break;
      case destructor:
        tmp = "deleted";
        break;
      case other:
        tmp = "something done";
    }
    std::cout << tmp << " " << name << "(" << _id << ")" << std::endl;
  }

  virtual ~A()
  {
    print(destructor, "A");
  }
protected:
  int _id;
};


#define NEW_CLASS(cls, parent) \
```

```
  class cls : public parent \
  { \
  public: \
    explicit cls(int id = 0) : parent(id) \
    { \
      print(constructor, #cls); \
    } \
    virtual ~cls() override \
    { \
      print(destructor, #cls); \
    } \
  };

NEW_CLASS(B, A)
NEW_CLASS(C, B)
NEW_CLASS(D, C)

NEW_CLASS(E, A)
NEW_CLASS(F, E)



#undef NEW_CLASS


int main()
{
  A(1);
  std::cout << std::endl;

  B(2);
  std::cout << std::endl;

  C(3);
  std::cout << std::endl;

  D(4);
  std::cout << std::endl;

  E(5);
  std::cout << std::endl;

  F(6);
  std::cout << std::endl;
}
```

Листинг 1 – Программа для исследования порядка вызова конструкторов и деструкторов

```
created A(1)
deleted A(1)

created A(2)
created B(2)
deleted B(2)
deleted A(2)

created A(3)
created B(3)
created C(3)
deleted C(3)
deleted B(3)
deleted A(3)

created A(4)
created B(4)
created C(4)
created D(4)
deleted D(4)
deleted C(4)
deleted B(4)
deleted A(4)

created A(5)
created E(5)
deleted E(5)
deleted A(5)

created A(6)
created E(6)
created F(6)
deleted F(6)
deleted E(6)
deleted A(6)
```

Листинг 2 – Результат работы программы

Задание 2

Задание 2. *Разработать систему классов для представления структуры синтаксического дерева арифметических выражений и вычисления его значения: Expr, Num, Add, Substract*

Программа 3 компилируется и исполняется, как ожидается. Система классов соответствует требованиям. Для контроля памяти был создан вспомогательный класс *BOperation* , который наследуется от *Expr* и представляет собой контейнер двух экземпляров классов *Expr* . Во избежании утечек памяти деструктор *Expr* делается виртуальным. Для удобного использования написанна функция *create_expr* , которая получая на вход строку создает синтаксическое дерево.

```
#include <iostream>
#include <string>
#include <algorithm>



class Expr
{
public:
  virtual int eval() = 0;
  virtual ~Expr() {}
};

class Num : public Expr
{
public:
  explicit Num(int n = 0) : _n(n)
  {}

  virtual int eval() override
  {
    return _n;
  }
protected:
  int _n;
};

class BOperation : public Expr
{
public:
  explicit BOperation(Expr* left, Expr* right) : _left(left), _right(right)
  {}

  virtual ~BOperation()
  {
    delete _left;
```

```
    delete _right;
  }
protected:
  Expr* _left;
  Expr* _right;
};

class Add : public BOperation
{
public:
  explicit Add(Expr* left, Expr* right) : BOperation(left, right)
  {}

  virtual int eval() override
  {
    return _left->eval() + _right->eval();
  }
};



class Substract : public BOperation
{
public:
  explicit Substract(Expr* left, Expr* right) : BOperation(left, right)
  {}

  virtual int eval() override
  {
    return _left->eval() - _right->eval();
  }
};


Expr* create_expr(std::string str)
{
  str.erase(std::remove(str.begin(), str.end(), ' '), str.end());

  const std::string::size_type not_found = std::string::npos;
  if (str.find('+') != not_found && str.find('+') != str.size()-1) {
    return new Add(
        create_expr(str.substr(0, str.find('+'))),
        create_expr(str.substr(str.find('+')+1))
        );
  } else if (str.find('-') != not_found && str.find('-') != str.size()-1) {
    return new Substract(
        create_expr(str.substr(0, str.find('-'))),
        create_expr(str.substr(str.find('-')+1))
        );
  } else
    return new Num(std::stoi(str));
}


int main()
{
  std::string tmp;
```

```
  //std::cin >> tmp;
  std::getline(std::cin, tmp);
  Expr* expr = create_expr(tmp);
  std::cout << tmp << " = " << expr->eval() << std::endl;
  delete expr;
};
```

Листинг 3 – Система классов

Задание 3

Задание 3. *Разработать систему классов, реализующих структуры структуры данных: Collection, Stack, Queue. Написать определения классов на языке C++. Написать следующую программу:*

- *Создать массив коллекций.*
- *Заполнить массив коллекциями различных типов (стэками и очередями).*
- *В цикле: используя методы put и take провести запись и чтение значений.*
- *В цикле: распечатать элементы коллекций массива.*

Проверить программу на предмет отсутствия утечек памяти.

Программа 4 компилируется и исполняется, как ожидается. Система классов соответствует требованиям. Представлен результат работы программы 5 . Утечки памяти отсутствуют, как и ожидается, что подтвердается утилитой *Valgrind* 6 .

```
#include <iostream>


class Collection
{
public:
  explicit Collection(std::size_t i = 1) : _buf(new int[i]), _size(i)
  {}

  virtual void put(int ) = 0;
  virtual int take() = 0;

  virtual ~Collection()
  {
    delete[] _buf;
  }

  int get(std::size_t i) const
  {
    if (i >= _size)
      return _buf[_size-1];
    return _buf[i];
  }

protected:
  int* _buf;
  std::size_t _size;
};


class Stack : public Collection
{
```

```
public:
  explicit Stack(std::size_t i) : Collection(i), _pos(0)
  {}

  virtual void put(int el) override
  {
    if (_pos >= _size)
      return ;
    _buf[_pos++] = el;
  }

  virtual int take() override
  {
    if (_pos > 0)
      return _buf[_pos--];
    return _buf[0];
  }
protected:
  std::size_t _pos;
};


class Queue : public Collection
{
public:
  explicit Queue(std::size_t i) : Collection(i), _pos(0), _queue_size(0)
  {}

  virtual void put(int el) override
  {
    if (_queue_size+1 >= _size)
      return ;
    _buf[(_pos+_queue_size++)%_size] = el;
  }

  virtual int take() override
  {
    if (_queue_size-- > 0)
      return _buf[_pos++];
    return _buf[0];
  }
protected:
  std::size_t _pos;
  std::size_t _queue_size;
};



int main(int argc, char** argv)
{
  const std::size_t N = 2;
  const std::size_t M = 6;
  Collection** array = new Collection*[N];
  for (int i = 0; i < N; ++i) {
    if (i < N/2)
      array[i] = new Stack(M);
    else
```

```
    array[i] = new Queue(M);
  }

  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < 2*M; ++j) {
      if (j < 2*M/3 || j > 4*M/3 || j % 3 == 1)
        array[i]->put(j);
      else
        array[i]->take();
    }
  }


  for (int i = 0; i < N; ++i) {
    std::cout << "Collection " << i << std::endl;
    for (int j = 0; j < M; ++j)
      std::cout << array[i]->get(j) << " ";
    std::cout << std::endl;
    delete array[i];
  }
  delete[] array;

}
```

Листинг 4 – Классы *Collection, Stack, Queue*

```
Collection 0
0 1 2 9 10 11
Collection 1
9 10 2 3 4 7
```

Листинг 5 – Вызов программы

```
==9006== Memcheck, a memory error detector
==9006== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==9006== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==9006== Command: ./a.out
==9006==
==9006==
==9006== HEAP SUMMARY:
==9006==     in use at exit: 0 bytes in 0 blocks
==9006==   total heap usage: 7 allocs, 7 frees, 73,864 bytes allocated
==9006==
==9006== All heap blocks were freed -- no leaks are possible
==9006==
==9006== For counts of detected and suppressed errors, rerun with: -v
==9006== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Листинг 6 – Проверка на наличие утечек памяти

Задание 4

Задание 4. *Используя модификации реализованных в предыдущих заданиях программ, исследовать зависимость объема памяти, необходимой для выполнения программы, от используемого способа передачи параметров объектного типа:*

- *по значению;*
- *по ссылке;*
- *использование указателя;*

Программа 7 компилируется и исполняется, как ожидается. Для удобства выполнения задания задаются макросы *VALUE, REFERENCE, POINTER* . Каждый макрос ответсвеннен за вызов соответсвующей ему функции, аргумент которой передается по методу соотносящемуся с названием макроса. Представленны результаты работы программы с активными макросами 8 , 9 , 10 . Больше всего памяти выделяется при передаче экземпляра класса *A* по значению, как и ожидается, потому что вызывается конструктор копирования.

```
#define VALUE

#define REFERENCE

#define POINTER


class A
{
public:
  A() : _a(new int(0))
  {}

  A(const A& a) : _a(new int(*(a._a)))
  {}

  A(A&& a) : _a(new int(*(a._a)))
  {}

  ~A()
  {
    delete _a;
  }
protected:
  int* _a;
};
```

```
#ifdef VALUE
void foo0(A a)
{}
#endif

#ifdef REFERENCE
void foo(A& a)
{}
#endif

#ifdef POINTER
void foo(A* a)
{}
#endif


int main()
{
  A a;
#ifdef VALUE
  foo0(a);
#endif

#ifdef REFERENCE
  foo(a);
#endif

#ifdef POINTER
  foo(&a);
#endif


  return 0;
}
```

Листинг 7 – Вызов программы

```
==6984== Memcheck, a memory error detector
==6984== Copyright (C) 2002–2017, and GNU GPL'd, by Julian Seward et al.
==6984== Using Valgrind–3.14.0 and LibVEX; rerun with –h for copyright info
==6984== Command: ./a.out
==6984==
==6984==
==6984== HEAP SUMMARY:
==6984==     in use at exit: 0 bytes in 0 blocks
==6984==   total heap usage: 3 allocs, 3 frees, 72,712 bytes allocated
==6984==
==6984== All heap blocks were freed -- no leaks are possible
==6984==
==6984== For counts of detected and suppressed errors, rerun with: –v
==6984== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Листинг 8 – Вызов программы с макором *VALUE*

```
==7017== Memcheck, a memory error detector
==7017== Copyright (C) 2002−2017, and GNU GPL'd, by Julian Seward et al.
==7017== Using Valgrind−3.14.0 and LibVEX; rerun with −h for copyright info
==7017== Command: ./a.out
==7017==
==7017==
==7017== HEAP SUMMARY:
==7017==     in use at exit: 0 bytes in 0 blocks
==7017==   total heap usage: 2 allocs, 2 frees, 72,708 bytes allocated
==7017==
==7017== All heap blocks were freed −− no leaks are possible
==7017==
==7017== For counts of detected and suppressed errors, rerun with: −v
==7017== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Листинг 9 – Вызов программы с макором *REFERENCE*

```
==7028== Memcheck, a memory error detector
==7028== Copyright (C) 2002−2017, and GNU GPL'd, by Julian Seward et al.
==7028== Using Valgrind−3.14.0 and LibVEX; rerun with −h for copyright info
==7028== Command: ./a.out
==7028==
==7028==
==7028== HEAP SUMMARY:
==7028==     in use at exit: 0 bytes in 0 blocks
==7028==   total heap usage: 2 allocs, 2 frees, 72,708 bytes allocated
==7028==
==7028== All heap blocks were freed −− no leaks are possible
==7028==
==7028== For counts of detected and suppressed errors, rerun with: −v
==7028== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Листинг 10 – Вызов программы с макором *POINTER*