Институт информационных технологий

Кафедра: Математическое и программное обеспечение ЭВМ

Дисциплина: Объектно-ориентированное программирование

ЛАБОРАТОРНАЯ РАБОТА № 4

Тема: Программирование шаблонных файлов и функций

Выполнил:

студент гр. 1ПИб-02-2оп-23

Кринкин Олег Алексеевич

Проверил:

ассистент Матевосян Ремик Артурович

ЗАДАНИЕ

1. Ознакомьтесь с УМП по ООП часть 2 раздел VII.
2. Модифицируйте контейнерный класс, реализованный по заданию ЛР2, используя шаблоны определения класса (обязательно) и шаблоны определения функции (по желанию).
3. Проверьте работоспособность АТД на тестовом наборе данных.

Примечания:

* Обязательно сделать шаблонным класс контейнер.
* Шаблоны определения функции на свое усмотрение.

Контрольные вопросы:

1. Для чего используется ключевое слово template?
2. В чем заключаются особенности параметров по умолчанию для шаблонов?
3. Для чего используются шаблоны функций?
4. Назовите разновидности дружественных функций шаблонного класса.
5. Дайте определение специализации шаблона.

ХОД РАБОТЫ

1. В объявлении класса ServerRoom в заголовок класса (net\_room.h) добавлено ключевое слово template с указанием типа T. У всех аргументов или возвращаемых значений компонентных функций тип Switch был заменен на T.

|  |
| --- |
| template <typename T>  class ServerRoom {  ...  const T\* search(int\*);  const T\*\* get\_array();  ...  }; |

1. Создан новый класс Router, который наследует класс Switch. К нему добавлено несколько новых полей (ssid, passwd, wpa) и компонентных функций. Этот класс будет использоваться для дальнейшего тестирования шаблона наряду со Switch.

|  |
| --- |
| class Router: public Switch {  private:  ...  char\* ssid;  char\* passwd;  bool wpa;  public:  ...  char\* get\_ssid() const;  bool wpa\_enabled() const;  void set\_ssid(char\*);  void set\_passwd(char\*);  void set\_wpa(bool);  void reset();  }; |

1. В заголовок определения компонентных функций добавлено ключевое слово template с указанием типа T. Все типы параметров и возвращаемых значений также заменены на T.

|  |
| --- |
| template <typename T>  ServerRoom<T>::ServerRoom(T& device) {  ...  }  template <typename T>  typename ServerRoom<T>::Shelf\* ServerRoom<T>::get\_shelf(ServerRoom<T>\* room, int shelf\_id) {  ...  } |

1. Для функций print() и search() определены инстанцирования для конкретных двух типов – Switch и Router.

|  |
| --- |
| template <>  void ServerRoom<Switch>::print(int shelf\_id) {  ...  }  template <>  void ServerRoom<Router>::print(int shelf\_id) {  ...  } |

1. В начале файла объявлены инстанцирования для каждого класса и их компонентных для двух типов данных – Switch и Router.

|  |
| --- |
| template class ServerRoom<Switch>;  template ServerRoom<Switch>::ServerRoom(Switch&);  template ServerRoom<Switch>::~ServerRoom<Switch>();  template ServerRoom<Switch>& ServerRoom<Switch>::operator+(Switch&);  template ServerRoom<Switch>& ServerRoom<Switch>::operator--(int);  template const Switch\* ServerRoom<Switch>::operator[](int);  template void ServerRoom<Switch>::sort();  template int ServerRoom<Switch>::get\_room\_size();  ... // Также для Router |

1. В файл Source.cpp добавлено тестирование контейнерного класса с новым типом данных.

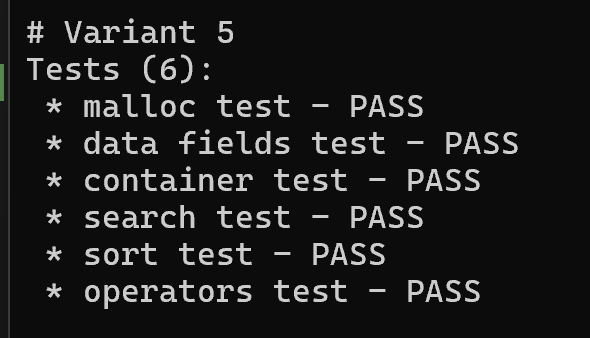
РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ РАБОТЫ

Результат исполнения программы

Используя некоторую систему автоматических «юнит» тестов, которая выполняет несколько тестов:

* выделение и освобождение памяти классами;
* заполнение и хранение данных в полях классов;
* хранение объектов классов в классе-контейнере;
* поиск объекта в контейнере;
* сортировка объектов;
* работа операторов,

получен следующий результат:



Текст программы:

|  |  |  |
| --- | --- | --- |
| net\_room.h | net\_room.cpp | Source.cpp |
| // Variant 5  #pragma once  #include <ostream>  template <typename T>  class ServerRoom {  private:  // Структура-"полка" (динамический список)  struct Shelf {  const T\* device;  Shelf\* next;  };  // Указатели на первую и последнюю полку  Shelf\* first, \* last;  Shelf\* get\_shelf(ServerRoom<T>\*, int);  public:  ServerRoom();  ServerRoom(T&);  ~ServerRoom();  ServerRoom<T>& operator+(T&); // Оператор присваивания ServerRoom и Switch  ServerRoom<T>& operator--(int); // Оператор декремента  const T\* operator[](int); // Оператор взятия по индексу  friend bool operator==(ServerRoom<T>&, ServerRoom<T>&); // Оператор сравнения  friend std::ostream& operator<<(std::ostream& stream, ServerRoom<T>&); // Оператор потокового вывода  void print(int);  void sort();  int get\_room\_size();  const T\* search(int\*);  const T\*\* get\_array();  }; | #include "net\_room.h"  #include "network.h"  #include <iostream>  // Инстанцирование для Switch  template class ServerRoom<Switch>;  template ServerRoom<Switch>::ServerRoom(Switch&);  template ServerRoom<Switch>::~ServerRoom<Switch>();  template ServerRoom<Switch>& ServerRoom<Switch>::operator+(Switch&);  template ServerRoom<Switch>& ServerRoom<Switch>::operator--(int);  template const Switch\* ServerRoom<Switch>::operator[](int);  template void ServerRoom<Switch>::sort();  template int ServerRoom<Switch>::get\_room\_size();  // Инстанцирование для Router  template class ServerRoom<Router>;  template ServerRoom<Router>::ServerRoom(Router&);  template ServerRoom<Router>::~ServerRoom();  template ServerRoom<Router>& ServerRoom<Router>::operator+(Router&);  template ServerRoom<Router>& ServerRoom<Router>::operator--(int);  template const Router\* ServerRoom<Router>::operator[](int);  template void ServerRoom<Router>::sort();  template int ServerRoom<Router>::get\_room\_size();  template <typename T>  ServerRoom<T>::ServerRoom() {  /\*  \* Конструктор по умолчанию - создаёт пустую первую "полку".  \*/  this->first = new Shelf;  this->first->device = nullptr;  this->first->next = nullptr;  this->last = this->first;  }  template <typename T>  ServerRoom<T>::ServerRoom(T& device) {  /\*  \* Конструктор по умолчанию - создаёт первую "полку" с объектом T.  \*/  this->first = new Shelf;  this->first->device = &device;  this->first->next = nullptr;  this->last = this->first;  }  template <typename T>  ServerRoom<T>::~ServerRoom() {  /\*  \* Декоструктор - удаляет все структуры-"полки", не затронув хранящиеся  \* в них объекты типа Switch.  \*/  Shelf\* current = this->first;  Shelf\* to\_delete = nullptr;  while (current->next != nullptr) {  to\_delete = current;  current = current->next;  delete to\_delete;  }  delete current;  }  template <typename T>  typename ServerRoom<T>::Shelf\* ServerRoom<T>::get\_shelf(ServerRoom<T>\* room, int shelf\_id) {  /\*  \* Функция возвращает указатель на "полку" под некоторым номером.  \* (приватна)  \*/  Shelf\* shelf = room->first;  for (int i = 0; i < shelf\_id; i++) {  if (shelf != nullptr)  shelf = shelf->next;  }  return shelf;  }  template <typename T>  ServerRoom<T>& ServerRoom<T>::operator+(T& device) {  /\*  \* Оператор добавляет "полку" в конец комнаты.  \*/  this->last->next = new Shelf;  this->last->next->device = &device;  this->last = this->last->next;  this->last->next = nullptr;  return \*this;  }  template <typename T>  ServerRoom<T>& ServerRoom<T>::operator--(int count) {  /\*  \* Оператор удаляет последнее количество "полок".  \*/  for (int i = 0; i < count; i++) {  Shelf\* prev = get\_shelf(this, get\_room\_size() - 1);  prev->next = nullptr;  delete this->last;  this->last = prev;  }  return \*this;  }  template <typename T>  const T\* ServerRoom<T>::operator[](int index) {  /\*  \* Оператор возвращает указатель на объект T указанной "полки".  \*/  Shelf\* shelf = get\_shelf(this, index);  return shelf->device;  }  template <typename T>  bool operator==(ServerRoom<T>& first, ServerRoom<T>& second) {  /\*  \* Оператор проверяет равны ли две комнаты по размеру  \*/  return first.get\_room\_size() == second.get\_room\_size();  }  std::ostream& operator<<(std::ostream& stream, ServerRoom<Switch>& room) {  /\*  \* Оператор печатает в консоль информацию обо всех устройствах в "комнате".  \*/  ServerRoom<Switch>::Shelf\* shelf = room.first;  int size = room.get\_room\_size();  for (int i = 0; i < size; i++) {  ServerRoom<Switch>::Shelf\* shelf = room.get\_shelf(&room, i);  if (shelf != nullptr && shelf->device != nullptr) {  stream << " \* Shelf " << i << " device info:" << std::endl;  shelf->device->print\_info();  } else  stream << " \* Shelf " << i << " is empty!" << std::endl;  shelf = shelf->next;  }  return stream;  }  std::ostream& operator<<(std::ostream& stream, ServerRoom<Router>& room) {  /\*  \* Оператор печатает в консоль информацию обо всех устройствах в "комнате".  \*/  ServerRoom<Router>::Shelf\* shelf = room.first;  int size = room.get\_room\_size();  for (int i = 0; i < size; i++) {  ServerRoom<Router>::Shelf\* shelf = room.get\_shelf(&room, i);  if (shelf != nullptr && shelf->device != nullptr) {  stream << " \* Shelf " << i << " device info:" << std::endl;  shelf->device->print\_info();  } else  stream << " \* Shelf " << i << " is empty!" << std::endl;  shelf = shelf->next;  }  return stream;  }  // Инстанцирование для Switch  template <>  void ServerRoom<Switch>::print(int shelf\_id) {  /\*  \* Функция печатает в консоли информацию о "полке" и её содержимом.  \*/  Shelf\* shelf = get\_shelf(this, shelf\_id);  if (shelf != nullptr && shelf->device != nullptr) {  std::cout << " \* Shelf " << shelf\_id << " device info:" << std::endl;  shelf->device->print\_info();  } else  std::cout << " \* Shelf " << shelf\_id << " is empty!" << std::endl;  }  // Инстанцирование для Router  template <>  void ServerRoom<Router>::print(int shelf\_id) {  /\*  \* Функция печатает в консоли информацию о "полке" и её содержимом.  \*/  Shelf\* shelf = get\_shelf(this, shelf\_id);  if (shelf != nullptr && shelf->device != nullptr) {  std::cout << " \* Shelf " << shelf\_id << " device info:" << std::endl;  shelf->device->print\_info();  } else  std::cout << " \* Shelf " << shelf\_id << " is empty!" << std::endl;  }  template <typename T>  void ServerRoom<T>::sort() {  /\*  \* Функция сортирует объекты T по количеству поключенных устройств  \* по возрастанию.  \*/  int room\_size = get\_room\_size();  for (int i = 0; i < room\_size; i++)  for (int j = 0; j < room\_size - 1 - i; j++) {  Shelf\* current = get\_shelf(this, j);  if (current->device > current->next->device) { // Использован оператор меньше  const T\* tmp;  tmp = current->next->device;  current->next->device = current->device;  current->device = tmp;  }  }  }  // Инстанцирование для Switch  template<>  const Switch\* ServerRoom<Switch>::search(int\* address) {  /\*  \* Функция поиска объекта T по адресу. Возвращает адрес объекта  \* или nullptr, если объект не найден.  \*/  Shelf\* current = this->first;  while (current->next != nullptr) {  for (int i = 0; i < 5; i++) {  if (current->device->get\_address()[i] != address[i]) {  if (current->next != nullptr)  current = current->next;  continue;  }  }  return current->device;  }  return nullptr;  }  // Инстанцирование для Router  template<>  const Router\* ServerRoom<Router>::search(int\* address) {  /\*  \* Функция поиска объекта T по адресу. Возвращает адрес объекта  \* или nullptr, если объект не найден.  \*/  Shelf\* current = this->first;  while (current->next != nullptr) {  for (int i = 0; i < 5; i++) {  if (current->device->get\_address()[i] != address[i]) {  if (current->next != nullptr)  current = current->next;  continue;  }  }  return current->device;  }  return nullptr;  }  template <typename T>  int ServerRoom<T>::get\_room\_size() {  /\*  \* Функция возвращает количество "полок" в комнате.  \*/  int count = 0;  Shelf\* current = this->first;  while (current->next != nullptr) {  if (current != nullptr)  current = current->next;  count++;  }  return count;  }  // Инстанцирование для Switch  template <>  const Switch\*\* ServerRoom<Switch>::get\_array() {  /\*  \* Функция возвращает массив указателей на объекты T.  \*/  int size = get\_room\_size();  const Switch\*\* array = new const Switch\*[size];  for (int i = 0; i < size; i++)  array[i] = (\*this)[i];  return array;  }  // Инстанцирование для Router  template <>  const Router\*\* ServerRoom<Router>::get\_array() {  /\*  \* Функция возвращает массив указателей на объекты T.  \*/  int size = get\_room\_size();  const Router\*\* array = new const Router\*[size];  for (int i = 0; i < size; i++)  array[i] = (\*this)[i];  return array;  } | #include "network.h"  #include "net\_room.h"  #include <iostream>  int main() {  Switch sw1 = Switch(new int[11]{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}, 11);  Switch sw2 = Switch(new int[2]{1, 2}, 2);  Switch sw3 = Switch(new int[8]{1, 2, 3, 4, 5, 6, 7, 8}, 8);  Switch sw4 = Switch(new int[5]{65, 53, 52, 24, 78});  ServerRoom<Switch> cab1 = ServerRoom<Switch>(sw1);  cab1 = cab1 + sw2;  cab1 = cab1 + sw3;  cab1 = cab1 + sw4;  std::cout << "# SWITCH INITIAL" << std::endl;  std::cout << cab1 << std::endl; // Использование потокового вывода  std::cout << "# SWITCH SEARCH TEST" << std::endl;  cab1.search(new int[5]{65, 53, 52, 24, 78})->print\_info();  std::cout << std::endl;  std::cout << "# SWITCH DELETE AND SORT" << std::endl;  cab1--;  cab1.sort();  std::cout << cab1 << std::endl; // Использование потокового вывода  std::cout << "# SWITCH ARRAY TEST" << std::endl;  const Switch\*\* s\_array = cab1.get\_array();  for (int i = 0; i < cab1.get\_room\_size(); i++)  s\_array[i]->print\_info();  const double\* packets = new double[6]{1, 45, 3243, 56, 65, 12};  Router rt1 = Router();  Router rt2 = Router(sw1);  Router rt3 = Router("CoolWLAN", "qwerty123");  Router rt4 = Router(packets, new int[4] {23, 32, 54, 65}, 4, new int[5] {30, 22, 15, 8, 7}, "CHSU Work", "VBhsuVYH9");  ServerRoom<Router> cab2 = ServerRoom<Router>(rt1);  cab2 = cab2 + rt1;  cab2 = cab2 + rt2;  cab2 = cab2 + rt3;  cab2 = cab2 + rt4;  std::cout << "# ROUTER INITIAL" << std::endl;  std::cout << cab2 << std::endl; // Использование потокового вывода  std::cout << "# ROUTER SEARCH TEST" << std::endl;  cab2.search(new int[5]{65, 53, 52, 24, 78})->print\_info();  std::cout << std::endl;  std::cout << "# ROUTER DELETE AND SORT" << std::endl;  cab2--;  cab2.sort();  std::cout << cab2 << std::endl; // Использование потокового вывода  std::cout << "# ROUTER ARRAY TEST" << std::endl;  const Router\*\* r\_array = cab2.get\_array();  for (int i = 0; i < cab2.get\_room\_size(); i++)  r\_array[i]->print\_info();  return 0;  } |

Текст тестирования:

|  |
| --- |
| #ifndef TESTS\_H  #define TESTS\_H  #endif // TESTS\_H  #include "network.h"  #include "net\_room.h"  class Tests {  private:  const double\* packets = new double[1]{15.6};  const int\* clients = new int[3]{1, 2, 3};  const int\* address = new int[5]{10, 11, 12, 13, 14};  const int connected = 3;  const char\* ssid = "CoolWLAN";  const char\* passwd = "qwerty123";  const bool wpa = false;  Switch\* s\_def;  Switch\*\* s\_units;  Router\* r\_def;  Router\*\* r\_units;  constexpr static const char\* errors[100] = {  "PASS", //0  "FAIL (Switch->connected is not equal)", //1  "FAIL (Switch->clients is not equal)", //2  "FAIL (Switch->address is not equal)", //3  "FAIL (Switch->packets is not equal)", //4  "FAIL (Router->connected is not equal)", //5  "FAIL (Router->clients is not equal)", //6  "FAIL (Router->address is not equal)", //7  "FAIL (Router->packets is not equal)", //8  "FAIL (Router->ssid is not equal)", //9  "FAIL (Router->passwd is not equal)", //10  "FAIL (Router->wpa is not equal)", //11  "FAIL (Switch container failed to save object)", //12  "FAIL (Router container failed to save object)", //13  "FAIL (Switch container failed to find object)", //14  "FAIL (Router container failed to find object)", //15  "FAIL (Switch container sorting is wrong)", //16  "FAIL (Router container sorting is wrong)", //17  "FAIL (Operator + is wrong for Switch)", //18  "FAIL (Operator -- is wrong for Switch)", //19  "FAIL (Operator [] is wrong for Switch)", //20  "FAIL (Operator == is wrong for Switch)", //21  "FAIL (Operator + is wrong for Router)", //22  "FAIL (Operator -- is wrong for Router)", //23  "FAIL (Operator [] is wrong for Router)", //24  "FAIL (Operator == is wrong for Router)", //25  };  public:  Tests() {  this->s\_def = new Switch(this->packets, this->clients, this->connected, this->address);  this->s\_units = new Switch\*[5] {  new Switch(this->clients, this->connected),  new Switch(this->address),  new Switch(this->clients, this->connected, this->address),  new Switch(this->packets, this->clients, this->connected, this->address),  new Switch(\*s\_def)  };  this->r\_def = new Router(this->packets, this->clients, this->connected, this->address, this->ssid, this->passwd);  this->r\_units = new Router\*[3] {  new Router(this->ssid, this->passwd),  new Router(\*r\_def),  new Router(this->packets, this->clients, this->connected, this->address, this->ssid, this->passwd)  };  }  ~Tests() {  delete[] this->s\_units;  delete this->s\_def;  delete[] this->r\_units;  delete this->r\_def;  }  const char\* memory() {  return errors[0];  }  const char\* data() {  if (s\_units[0]->connected != connected) return errors[1];  if (\*(s\_units[0]->clients) != \*clients) return errors[2];  if (\*(s\_units[1]->address) != \*address) return errors[3];  if (s\_units[2]->connected != connected) return errors[1];  if (\*(s\_units[2]->clients) != \*clients) return errors[2];  if (\*(s\_units[2]->address) != \*address) return errors[3];  if (s\_units[3]->connected != connected) return errors[1];  if (\*(s\_units[3]->clients) != \*clients) return errors[2];  if (\*(s\_units[3]->address) != \*address) return errors[3];  if (\*(s\_units[3]->packets) != \*packets) return errors[4];  if (\*(s\_units[4]->packets) != \*(s\_def->packets)) return errors[4];  if (\*(s\_units[4]->clients) != \*(s\_def->clients)) return errors[2];  if (\*(s\_units[4]->address) != \*(s\_def->address)) return errors[3];  if (s\_units[4]->connected != s\_def->connected) return errors[1];  if (\*(r\_units[0]->ssid) != \*(ssid)) return errors[9];  if (\*(r\_units[0]->passwd) != \*(passwd)) return errors[10];  if (\*(r\_units[1]->packets) != \*packets) return errors[8];  if (\*(r\_units[1]->clients) != \*clients) return errors[6];  if (\*(r\_units[1]->address) != \*address) return errors[7];  if (r\_units[1]->connected != connected) return errors[5];  if (\*(r\_units[1]->ssid) != \*ssid) return errors[8];  if (\*(r\_units[1]->passwd) != \*passwd) return errors[10];  if (r\_units[1]->wpa != wpa) return errors[11];  if (\*(r\_units[2]->packets) != \*(r\_def->packets)) return errors[8];  if (\*(r\_units[2]->clients) != \*(r\_def->clients)) return errors[6];  if (\*(r\_units[2]->address) != \*(r\_def->address)) return errors[7];  if (r\_units[2]->connected != r\_def->connected) return errors[5];  if (\*(r\_units[2]->ssid) != \*(r\_def->ssid)) return errors[9];  if (\*(r\_units[2]->passwd) != \*(r\_def->passwd)) return errors[10];  if (r\_units[2]->wpa != r\_def->wpa) return errors[11];  return errors[0];  }  const char\* container() {  ServerRoom<Switch> s\_cont = ServerRoom<Switch>();  for (int i=0; i<5; i++)  s\_cont = s\_cont + \*s\_units[i];  for (int i=0; i<5; i++)  if (s\_cont.get\_shelf(&s\_cont, i)->device != s\_units[i])  return errors[12];  ServerRoom<Router> r\_cont = ServerRoom<Router>();  for (int i=0; i<5; i++)  r\_cont = r\_cont + \*r\_units[i];  for (int i=0; i<5; i++)  if (r\_cont.get\_shelf(&r\_cont, i)->device != r\_units[i])  return errors[13];  return errors[0];  }  const char\* search() {  ServerRoom<Switch> s\_cont = ServerRoom<Switch>();  for (int i=0; i<5; i++)  s\_cont = s\_cont + \*s\_units[i];  const Switch\* sw = s\_cont.search(this->address);  if (sw != s\_units[1]) return errors[14];  ServerRoom<Router> r\_cont = ServerRoom<Router>();  for (int i=0; i<5; i++)  r\_cont = r\_cont + \*r\_units[i];  const Router\* rt = r\_cont.search(this->address);  if (rt != r\_units[1]) return errors[15];  return errors[0];  }  const char\* sort() {  Switch\*\* ls\_units = new Switch\*[3] {  new Switch(new int[3] {1, 2, 3}, 3),  new Switch(new int[4] {1, 2, 3, 4}, 4),  new Switch(new int[5] {1, 2, 3, 4, 5}, 5)  };  ServerRoom<Switch> s\_cont = ServerRoom<Switch>();  for (int i=0; i<3; i++)  s\_cont = s\_cont + \*ls\_units[i];  s\_cont.sort();  if (s\_cont.first->device->connected != 3 || s\_cont.last->device->connected != 5)  return errors[16];  Router\*\* lr\_units = new Router\*[3] {  new Router(this->packets, new int[3] {1, 2, 3}, 3, this->address, this->ssid, this->passwd),  new Router(this->packets, new int[4] {1, 2, 3, 4}, 4, this->address, this->ssid, this->passwd),  new Router(this->packets, new int[5] {1, 2, 3, 4, 5}, 5, this->address, this->ssid, this->passwd)  };  ServerRoom<Router> r\_cont = ServerRoom<Router>();  for (int i=0; i<3; i++)  r\_cont = r\_cont + \*lr\_units[i];  r\_cont.sort();  if (r\_cont.first->device->connected != 3 || r\_cont.last->device->connected != 5)  return errors[17];  return errors[0];  };  const char\* opers() {  Switch\* sw = new Switch();  ServerRoom<Switch> c\_sw = ServerRoom<Switch>();  c\_sw = c\_sw + \*sw;  if (c\_sw.first->device != sw) return errors[18];  c\_sw--;  if (c\_sw.first != nullptr) return errors[19];  ServerRoom<Switch> s\_cont = ServerRoom<Switch>();  for (int i = 0; i < 3; i++)  s\_cont = s\_cont + \*s\_units[i];  if (s\_cont[2] != s\_units[2]) return errors[20];  ServerRoom<Switch> s\_cont\_1 = ServerRoom<Switch>();  for (int i = 0; i < 3; i++)  s\_cont\_1 = s\_cont\_1 + \*s\_units[i];  ServerRoom<Switch> s\_cont\_2 = ServerRoom<Switch>();  for (int i = 0; i < 3; i++)  s\_cont\_2 = s\_cont\_2 + \*s\_units[i];  if (!(s\_cont\_1 == s\_cont\_2)) return errors[21];  Router\* rt = new Router();  ServerRoom<Router> c\_rt = ServerRoom<Router>();  c\_rt = c\_rt + \*rt;  if (c\_rt.first->device != rt) return errors[18];  c\_rt--;  if (c\_rt.first != nullptr) return errors[19];  ServerRoom<Router> r\_cont = ServerRoom<Router>();  for (int i = 0; i < 3; i++)  r\_cont = r\_cont + \*r\_units[i];  if (r\_cont[2] != r\_units[2]) return errors[20];  ServerRoom<Router> r\_cont\_1 = ServerRoom<Router>();  for (int i = 0; i < 3; i++)  r\_cont\_1 = r\_cont\_1 + \*r\_units[i];  ServerRoom<Router> r\_cont\_2 = ServerRoom<Router>();  for (int i = 0; i < 3; i++)  r\_cont\_2 = r\_cont\_2 + \*r\_units[i];  if (!(r\_cont\_2 == r\_cont\_2)) return errors[21];  return errors[0];  }  }; |

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были освоены навыки перегрузки операторов как унарных, так и бинарных, как компонентными функциями, так и внешними дружественными. Также объявленные для класса контейнера операторы были применены на практике при алгоритме сортировки и вывода на экран.