|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **ИУК «Информатика и управление»** |
| **КАФЕДРА** | **ИУК4 «Программное обеспечение ЭВМ,** |
| **информационные технологии»** | |

**ЛАБОРАТОРНАЯ РАБОТА №4**

**«Знакомство с контейнерами»**

**ДИСЦИПЛИНА: «Высокоуровневое программирование»**



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Выполнил: студент гр. ИУК4-21Б | |  |  | ( | Суриков Н.С | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |
| Проверил: | |  |  | ( | Пчелинцева Н. И. | ) |
|  |  |  | (подпись) |  | (Ф.И.О.) |  |

|  |  |
| --- | --- |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | |
|  | - Балльная оценка:  - Оценка: |

**Цель:** приобретение практических навыков по созданию и обработке контейнеров данных.

**Задачи:**

1. Изучить понятие контейнера;

2. Познакомиться с итераторами и научиться применять их;

3. Ознакомиться с классами-контейнерами в языке С++.

**Условие задачи:**

*1. Замените C-строки на использование встроенного класса-контейнера*

*string.*

*2. Создайте собственный класс-контейнер для хранения набора персональных сущностей. По аналогии с приведенным примером (vector) он должен включать в себя динамический массив данных, размер массива и базовые функции по взаимодействию с ним: вывод всех элементов, получение, изменение, добавление, удаление элементовпо индексу, сортировку и фильтрацию по одному главному признаку. Последние две функции были разработаны в прошлой лабораторной работе.*

*3. В Вашем проекте замените имеющиеся массив*

**Листинг программы:**

Main.cpp:

1 *#include* *"Menu/CMenu/CMenu.h"*

2 *#include* *"Menu/CMenuItem/CMenuItem.h"*

3 *#include* *"Models/Car/Car.h"*

4 *#include* *"Models/Client/Client.h"*

5 *#include* *"Models/Employee/Employee.h"*

6 *#include* *"Storage/Storage.h"*

7 *#include* *"Tools/Tools.h"*

8 *#include* *<algorithm>*

9 *#include* *<iostream>*

10

11 using namespace std;

12

13 *#pragma region функции-заглушки*

14

15 int testCar(int index)

16 {

17 using namespace SNS;

18 Car car("Toyota", 25000.0, 2022, "Sedan", "Bluetooth, Backup Camera");

19 car.displayInfo();

20 return index;

21 }

22

23 int testEmployee(int index)

24 {

25 using namespace SNS;

26 Employee employee("John", "Doe", 25, "johndoe", "12345", "Manager");

27 employee.displayPublicInfo();

28 employee.displayPrivateInfo();

29 return index;

30 }

31

32 int testClient(int index)

33 {

34 using namespace SNS;

35 Client client("John", "Doe", 25, "johndoe", "12345", "Service");

36 client.displayPublicInfo();

37 client.displayPrivateInfo();

38 return index;

39 }

40 *#pragma endregion*

41

42 void renderMain()

43 {

44 SNS::clearScreen();

45 cout << "Добро пожаловать в главное меню\n"

46 << "============================================\n\n"

47 << endl;

48 }

49 namespace SNS

50 {

51

52 void addCarToStorage()

53 {

54 Car car;

55 std::cin >> car;

56 Storage::getStorage()->cars\_list.push\_back(new Car(\*static\_cast<Car \*>(&car)));

57 }

58

59 void removeCarFromStorage()

60 {

61 std::string brand;

62 std::cout << "Введите марку машины для удаления: ";

63 std::cin >> brand;

64

65 auto &cars = Storage::getStorage()->cars\_list;

66 auto it = std::find\_if(cars.begin(), cars.end(), [&](const Car \*car)

67 { return car->getBrand() == brand; });

68

69 if (it != cars.end())

70 {

71 cars.erase(it);

72 std::cout << "Машина успешно удалена из хранилища." << std::endl;

73 }

74 else

75 {

76 std::cout << "Машина с указанной маркой не найдена в хранилище." << std::endl;

77 }

78 }

79

80 void sortCarsInStorage()

81 {

82 std::string criteria;

83 std::cout << "Выберите критерий сортировки для машин (brand, price, year): ";

84 std::cin >> criteria;

85

86 auto &cars = Storage::getStorage()->cars\_list;

87 std::sort(cars.begin(), cars.end(), [&](const Car \*a, const Car \*b)

88 { return a > b; });

89 }

90

91 void addClientToStorage()

92 {

93 Client client;

94 std::cin >> client;

95 Storage::getStorage()->users\_list.push\_back(new Client(\*static\_cast<Client \*>(&client)));

96 }

97

98 void removeClientFromStorage()

99 {

100 std::string login;

101 std::cout << "Введите логин клиента для удаления: ";

102 std::cin >> login;

103

104 auto &clients = Storage::getStorage()->users\_list;

105 auto it = std::find\_if(clients.begin(), clients.end(), [&](const User \*client)

106 { return client->getLogin() == login; });

107

108 if (it != clients.end())

109 {

110 clients.erase(it);

111 std::cout << "Клиент успешно удален из хранилища." << std::endl;

112 }

113 else

114 {

115 std::cout << "Клиент с указанным логином не найден в хранилище." << std::endl;

116 }

117 }

118

119 void sortClientsInStorage()

120 {

121 std::string criteria;

122 auto &clients = Storage::getStorage()->users\_list;

123 std::sort(clients.begin(), clients.end(), [&](const User \*a, const User \*b)

124 { return a->getName() < b->getName(); });

125 }

126

127 void displayCarsFromStorage()

128 {

129 auto cars = Storage::getStorage()->cars\_list;

130 for (const auto &car : cars)

131 {

132 cout << \*car << endl;

133 }

134

135 if (cars.empty())

136 {

137 std::cout << "Машин нет." << std::endl;

138 }

139 }

140

141 void displayClientsFromStorage()

142 {

143 auto clients = Storage::getStorage()->users\_list;

144 for (const auto &client : clients)

145 {

146 cout << \*(Client \*)client << endl;

147 }

148

149 if (clients.empty())

150 {

151 std::cout << "Клиентов нет." << std::endl;

152 }

153 }

154

155 CMenu \*createMainMenu()

156 {

157 CMenu \*menu = new CMenu("Главное меню",

158 ItemList{

159 CMenuItem("Добавить машину в хранилище", [](int index) -> int

160 {addCarToStorage();

161 return index; }),

162 CMenuItem("Удалить машину из хранилища", [](int index) -> int

163 {removeCarFromStorage(); return index; }),

164 CMenuItem("Сортировать машины в хранилище", [](int index) -> int

165 {sortCarsInStorage(); return index; }),

166 *// CMenuItem("Добавить сотрудника в хранилище", [](int index) -> int*

167 *// {addEmployeeToStorage; return index; }),*

168 *// CMenuItem("Удалить сотрудника из хранилища", [](int index) -> int*

169 *// {removeEmployeeFromStorage; return index; }),*

170 *// CMenuItem("Сортировать сотрудников в хранилище", [](int index) -> int*

171 *// {sortEmployeesInStorage; return index; }),*

172 CMenuItem("Добавить клиента в хранилище", [](int index) -> int

173 {addClientToStorage(); return index; }),

174 CMenuItem("Удалить клиента из хранилища", [](int index) -> int

175 {removeClientFromStorage(); return index; }),

176 CMenuItem("Сортировать клиентов в хранилище", [](int index) -> int

177 {sortClientsInStorage(); return index; }),

178 CMenuItem("Показать все машины в хранилище", [](int index) -> int

179 {displayCarsFromStorage(); return index; }),

180 *// CMenuItem("Показать всех сотрудников в хранилище", [](int index) -> int*

181 *// {displayEmployeesFromStorage; return index; }),*

182 CMenuItem("Показать всех клиентов в хранилище", [](int index) -> int

183 {displayClientsFromStorage(); return index; })});

184 return menu;

185 }

186 }

187

188 int main()

189 {

190 using namespace SNS;

191

192 renderMain();

193 Storage::createStorage("./db.txt");

194 Storage \*storage = Storage::getStorage();

195

196 CMenu &menu = \*createMainMenu();

197

198 do

199 {

200 *// выводим меню*

201 cout << menu;

202

203 *// ожидаем ввод от пользователя*

204 cin >> menu;

205 clearScreen();

206 *// запускаем заданную функцию*

207 } while (menu() != -1);

208

209 *// удаляем меню*

210 delete &menu;

211 return 0;

212 }

CMenuItem.cpp:

1 *#include* *"./CMenuItem.h"*

2

3 namespace SNS

4 {

5 CMenuItem::CMenuItem(std::string name, Func func) : item\_name(name), func(func)

6 {

7 }

8

9 std::string CMenuItem::getName()

10 {

11 return item\_name;

12 }

13

14 void CMenuItem::print()

15 {

16 std::cout << item\_name;

17 }

18

19 int CMenuItem::run()

20 {

21 return func();

22 }

23 } *// namespace SNS*

24

CMenu.h:

1 *#pragma once*

2

3 *#include* *"./CMenuItem.h"*

4 *#include* *<cstddef>*

5

6 namespace SNS

7 {

8 class CMenu

9 {

10 public:

11 CMenu(std::string, CMenuItem \*, std::size\_t);

12 int getSelect() const;

13 bool isRun() const;

14 std::string getTitle();

15 size\_t getCount() const;

16 CMenuItem \*getItems();

17 void print();

18 int runCommand();

19

20 private:

21 int select{-1};

22 size\_t count{};

23 bool running{};

24 std::string title{};

25 CMenuItem \*items{};

26 };

27 } *// namespace SNS*

28

CMenu.cpp:

1 *#include* *"./CMenu.h"*

2

3 namespace SNS

4 {

5 CMenu::CMenu(std::string title, CMenuItem \*items, size\_t count) : title(title), items(items), count(count)

6 {

7 }

8

9 int CMenu::getSelect() const

10 {

11 return select;

12 }

13

14 bool CMenu::isRun() const

15 {

16 return running;

17 }

18

19 size\_t CMenu::getCount() const

20 {

21 return count;

22 }

23

24 std::string CMenu::getTitle()

25 {

26 return title;

27 }

28

29 CMenuItem \*CMenu::getItems()

30 {

31 return items;

32 }

33

34 void CMenu::print()

35 {

36 for (size\_t i{}; i < count; ++i)

37 {

38 std::cout << i << ". ";

39 items[i].print();

40 std::cout << std::endl;

41 }

42 }

43

44 int CMenu::runCommand()

45 {

46 print();

47 std::cout << "\n Select >> ";

48 std::cin >> select;

49 return items[select].run();

50 }

51 } *// namespace SNS*

52

Car.h:

1 *#pragma once*

2 *#include* *<string>*

3

4 namespace SNS

5 {

6 class Car

7 {

8 private:

9 std::string brand;

10 double price;

11 int year;

12 std::string description;

13 std::string features;

14

15 public:

16 Car(const std::string, double, int, const std::string, const std::string);

17

18 void displayInfo();

19

20 std::string getBrand() const;

21 double getPrice() const;

22 int getYear() const;

23 std::string getDescription() const;

24 std::string getFeatures() const;

25

26 void setBrand(const std::string &brand);

27 void setPrice(double price);

28 void setYear(int year);

29 void setDescription(const std::string &description);

30 void setFeatures(const std::string &features);

31 };

32 } *// namespace SNS*

Car.cpp:

1 *#include* *"./Car.h"*

2 *#include* *<iostream>*

3

4 namespace SNS

5 {

6 Car::Car(const std::string brand, double price, int year, const std::string description, const std::string features)

7 : brand(brand), price(price), year(year), description(description), features(features)

8 {

9 }

10

11 void Car::displayInfo()

12 {

13 std::cout << "Brand: " << brand << std::endl;

14 std::cout << "Price: " << price << std::endl;

15 std::cout << "Year: " << year << std::endl;

16 std::cout << "Description: " << description << std::endl;

17 std::cout << "Features: " << features << std::endl;

18 }

19

20 std::string Car::getBrand() const

21 {

22 return brand;

23 }

24

25 double Car::getPrice() const

26 {

27 return price;

28 }

29

30 int Car::getYear() const

31 {

32 return year;

33 }

34

35 std::string Car::getDescription() const

36 {

37 return description;

38 }

39

40 std::string Car::getFeatures() const

41 {

42 return features;

43 }

44

45 void Car::setBrand(const std::string &brand)

46 {

47 this->brand = brand;

48 }

49

50 void Car::setPrice(double price)

51 {

52 this->price = price;

53 }

54

55 void Car::setYear(int year)

56 {

57 this->year = year;

58 }

59

60 void Car::setDescription(const std::string &description)

61 {

62 this->description = description;

63 }

64

65 void Car::setFeatures(const std::string &features)

66 {

67 this->features = features;

68 }

69

70 }

71

Client.h:

1 *#pragma once*

2 *#include* *"../User/User.h"*

3

4 namespace SNS

5 {

6 class Client : public User

7 {

8 public:

9 Client(std::string, std::string, int, std::string,

10 std::string, std::string);

11 std::string getService() const;

12 void setService(const std::string &service);

13 void displayPublicInfo();

14 void displayPrivateInfo();

15

16 protected:

17 std::string service;

18 };

19 }

Client.cpp:

1 *#include* *"Client.h"*

2

3 namespace SNS

4 {

5 Client::Client(std::string name, std::string surname, int age, std::string login,

6 std::string password, std::string service)

7 : User(name, surname, age, login, password), service(service)

8 {

9 }

10

11 void Client::displayPublicInfo(){

12 User::displayPublicInfo();

13 std::cout << "Service: " << service << std::endl;

14 }

15

16 void Client::displayPrivateInfo(){

17 User::displayPrivateInfo();

18 }

19

20 std::string Client::getService() const

21 {

22 return service;

23 }

24

25 void Client::setService(const std::string &service)

26 {

27 this->service = service;

28 }

29 }

User.h:

1 *#pragma once*

2

3 *#include* *<iostream>*

4 namespace SNS

5 {

6 class User

7 {

8 public:

9 User(std::string m\_name, std::string, int, std::string, std::string);

10 std::string m\_name;

11 std::string m\_surname;

12 int m\_age;

13 std::string m\_login;

14 std::string m\_password;

15

16 virtual void displayPublicInfo() = 0;

17 virtual void displayPrivateInfo() = 0;

18 };

19 }

User.cpp:

1 *#include* *"./User.h"*

2 *#include* *<iostream>*

3

4 namespace SNS

5 {

6 User::User(std::string name, std::string surname, int age, std::string login, std::string password) : m\_name(name), m\_surname(surname), m\_age(age), m\_login(login), m\_password(password)

7 {

8 }

9 void User::displayPublicInfo()

10 {

11 std::cout << "Name: " << m\_name << std::endl;

12 std::cout << "Surname: " << m\_surname << std::endl;

13 std::cout << "Age: " << m\_age << std::endl;

14 }

15 void User::displayPrivateInfo()

16 {

17 std::cout << "Login: " << m\_login << std::endl;

18 std::cout << "Password: " << m\_password << std::endl;

19 }

20 }

Employee.h:

1 *#pragma once*

2 *#include* *"../User/User.h"*

3

4 namespace SNS

5 {

6 class Employee : public User

7 {

8 public:

9 Employee(std::string name, std::string surname, int age, std::string login,

10 std::string password, std::string post);

11 std::string getPost() const;

12 void setPost(const std::string &post);

13 void displayPublicInfo();

14 void displayPrivateInfo();

15

16 protected:

17 std::string post;

18 };

19 }

Employee.cpp:

1 *#include* *"Employee.h"*

2

3 namespace SNS

4 {

5

6 Employee::Employee(std::string name, std::string surname, int age, std::string login,

7 std::string password, std::string post)

8 : User(name, surname, age, login, password), post(post)

9 {

10 }

11

12 std::string Employee::getPost() const

13 {

14 return post;

15 }

16

17 void Employee::setPost(const std::string &post)

18 {

19 this->post = post;

20 }

21

22 void Employee::displayPublicInfo()

23 {

24 User::displayPublicInfo();

25 std::cout << "Post: " << post << std::endl;

26 }

27

28 void Employee::displayPrivateInfo()

29 {

30 User::displayPrivateInfo();

31 }

32 }

**Storage.h**

2

3 *#pragma once*

4

5 *#include* *<string>*

6 *#include* *<vector>*

7

8 *#include* *"../Models/Car/Car.h"*

9 *#include* *"../Models/User/User.h"*

10

11 using std::string;

12 using std::vector;

13

14 class Storage

15 {

16 public:

17 Storage() = delete;

18 Storage(const Storage &) = delete; *// удаляем конструктор копирования*

19 Storage &operator=(const Storage &) = delete; *// удаляем оператор присваивания*

20

21 static Storage &createStorage(string root\_path);

22 static Storage \*getStorage();

23

24 ~Storage();

25

26 string app\_name{"Dealship"};

27 SNS::Car::CarsList cars\_list{};

28 SNS::User::UserList users\_list{};

29

30 private:

31 explicit Storage(string root\_path);

32

33 static Storage \*s\_storage;

34

35 };

MyVector.h

1 *#ifndef VECTOR*

2 *#define VECTOR*

3

4 *#include* *<format>*

5

6 template <typename T>

7 class Vector

8 {

9 public:

10 template <typename U>

11 class Iter

12 {

13 public:

14 friend class Vector;

15

16 Iter(const Iter &iter);

17

18 friend bool operator==(const Iter &iter1, const Iter &iter2)

19 {

20 return iter1.\_obj == iter2.\_obj;

21 }

22

23 friend bool operator!=(const Iter &iter1, const Iter &iter2)

24 {

25 return iter1.\_obj != iter2.\_obj;

26 }

27

28 Iter &operator++();

29 Iter operator++(int);

30 Iter &operator--();

31 Iter operator--(int);

32 friend Iter operator+(const Iter &iter, const int n){

33 return Vector<T>::Iter<U>(iter.\_obj + n);

34 }

35

36 friend Iter operator-(const Iter &iter, const int n){

37 return Vector<T>::Iter<U>(iter.\_obj - n);

38 }

39 Iter &operator+=(const int n);

40 Iter &operator-=(const int n);

41 U &operator\*() const;

42

43 private:

44 U \*\_obj{nullptr};

45

46 Iter(U \*obj);

47 Iter(const U \*obj);

48 };

49

50 typedef Iter<T> iterator;

51 typedef Iter<const T> constIterator;

52

53 Vector();

54 Vector(const Vector &vector);

55 ~Vector();

56

57 void pushBack(const T &obj);

58 void insert(const T &obj, const int index = 0);

59 void popBack();

60 void erase(const int index = 0);

61 void clear();

62 void sort(const bool reverse = 0);

63

64 bool empty() const;

65 int size() const;

66 int capacity() const;

67

68 T &at(const int index);

69 const T &at(const int index) const;

70

71 iterator begin();

72 iterator end();

73 constIterator begin() const;

74 constIterator end() const;

75

76 T &operator[](const int index);

77

78 private:

79 T \*\_start{nullptr};

80 int \_length{0};

81 int \_capacity{0};

82

83 void init();

84 void increaseCapacity();

85 void decreaseCapasity();

86 };

87

88 template <typename T>

89 inline Vector<T>::Vector() { init(); }

90

91 template <typename T>

92 inline Vector<T>::Vector(const Vector &vector) : Vector()

93 {

94 for (int i = 0; i < vector.\_length; i++)

95 {

96 pushBack(vector.at(i));

97 }

98 }

99

100 template <typename T>

101 inline Vector<T>::~Vector() { delete[] \_start; }

102

103 template <typename T>

104 inline void Vector<T>::pushBack(const T &obj)

105 {

106 if (\_length == \_capacity)

107 {

108 increaseCapacity();

109 }

110 \_start[\_length++] = obj;

111 }

112

113 template <typename T>

114 inline void Vector<T>::insert(const T &obj, const int index)

115 {

116 if (\_length == \_capacity)

117 {

118 increaseCapacity();

119 }

120 if (index > \_length || index < 0)

121 {

122 throw std::format("OutOfBoundsException: Vector index out of range ({})", index);

123 }

124 for (int i = \_length; i >= 0; i--)

125 {

126 if (i == index)

127 {

128 \_start[i] = obj;

129 \_length++;

130 break;

131 }

132 \_start[i] = \_start[i - 1];

133 }

134 }

135

136 template <typename T>

137 inline void Vector<T>::popBack()

138 {

139 if (--\_length == \_capacity / 2 && \_capacity > 1)

140 {

141 decreaseCapasity();

142 }

143 }

144

145 template <typename T>

146 inline void Vector<T>::erase(const int index)

147 {

148 if (index > \_length || index < 0)

149 {

150 throw std::format("OutOfBoundsException: Vector index out of range ({})", index);

151 }

152 for (int i = index; i < \_length; i++)

153 {

154 \_start[i] = \_start[i + 1];

155 }

156 if (--\_length == \_capacity / 2 && \_capacity > 1)

157 {

158 decreaseCapasity();

159 }

160 }

161

162 template <typename T>

163 inline void Vector<T>::clear()

164 {

165 delete[] \_start;

166 init();

167 }

168

169 template <typename T>

170 inline void Vector<T>::sort(const bool reverse)

171 {

172 for (int i = 0; i < \_length - 1; i++)

173 {

174 for (int j = 0; j < \_length - 1 - i; j++)

175 {

176 if (!reverse && \_start[j] > \_start[j + 1] || reverse && \_start[j] < \_start[j + 1])

177 {

178 std::swap(\_start[j], \_start[j + 1]);

179 }

180 }

181 }

182 }

183

184 template <typename T>

185 inline bool Vector<T>::empty() const { return \_length == 0; }

186

187 template <typename T>

188 inline int Vector<T>::size() const { return \_length; }

189

190 template <typename T>

191 inline int Vector<T>::capacity() const { return \_capacity; }

192

193 template <typename T>

194 inline T &Vector<T>::at(const int index)

195 {

196 if (index >= \_length || index < 0)

197 {

198 throw std::format("OutOfBoundsException: Vector index out of range ({})", index);

199 }

200 return \_start[index];

201 }

202

203 template <typename T>

204 inline const T &Vector<T>::at(const int index) const

205 {

206 if (index >= \_length || index < 0)

207 {

208 throw std::format("OutOfBoundsException: Vector index out of range ({})", index);

209 }

210 return \_start[index];

211 }

212

213 template <typename T>

214 inline Vector<T>::iterator Vector<T>::begin() { return iterator(\_start); }

215

216 template <typename T>

217 inline Vector<T>::iterator Vector<T>::end() { return iterator(\_start + \_length); }

218

219 template <typename T>

220 inline Vector<T>::constIterator Vector<T>::begin() const { return constIterator(\_start); }

221

222 template <typename T>

223 inline Vector<T>::constIterator Vector<T>::end() const { return constIterator(\_start + \_length); }

224

225 template <typename T>

226 inline T &Vector<T>::operator[](const int index) { return \_start[index % \_length]; }

227

228 template <typename T>

229 inline void Vector<T>::init()

230 {

231 \_length = 0;

232 \_capacity = 2;

233 \_start = new T[\_capacity];

234 }

235

236 template <typename T>

237 inline void Vector<T>::increaseCapacity()

238 {

239 T \*newVector = new T[(\_capacity \*= 2)];

240 for (int index = 0; index < \_length; index++)

241 {

242 newVector[index] = \_start[index];

243 }

244 delete[] \_start;

245 \_start = newVector;

246 }

247

248 template <typename T>

249 inline void Vector<T>::decreaseCapasity()

250 {

251 if (\_capacity <= 1)

252 {

253 throw std::format("TooSmallCapacityException: Vector capacity less than 1 ({})", \_capacity);

254 }

255 if (\_length > \_capacity / 2)

256 {

257 throw std::format("DataLossException: Vector capacity less than length (cap: {}; len: {})", \_capacity, \_length);

258 }

259 T \*newVector = new T[(\_capacity /= 2)];

260 for (int index = 0; index < \_length; index++)

261 {

262 newVector[index] = \_start[index];

263 }

264 delete[] \_start;

265 \_start = newVector;

266 }

267

268 template <typename T>

269 template <typename U>

270 inline Vector<T>::Iter<U>::Iter(const Iter& iter) : Iter(iter.\_obj) {}

271

272 template <typename T>

273 template <typename U>

274 inline Vector<T>::Iter<U> &Vector<T>::Iter<U>::operator++()

275 {

276 ++\_obj;

277 return \*this;

278 }

279

280 template <typename T>

281 template <typename U>

282 inline Vector<T>::Iter<U> Vector<T>::Iter<U>::operator++(int) { return iter(\_obj++); }

283

284 template <typename T>

285 template <typename U>

286 inline Vector<T>::Iter<U> &Vector<T>::Iter<U>::operator--()

287 {

288 ++\_obj;

289 return \*this;

290 }

291

292 template <typename T>

293 template <typename U>

294 inline Vector<T>::Iter<U> Vector<T>::Iter<U>::operator--(int) { return iter(\_obj--); }

295

296 template <typename T>

297 template <typename U>

298 inline Vector<T>::Iter<U> &Vector<T>::Iter<U>::operator+=(const int n)

299 {

300 \_obj += n;

301 return \*this;

302 }

303

304 template <typename T>

305 template <typename U>

306 inline Vector<T>::Iter<U> &Vector<T>::Iter<U>::operator-=(const int n)

307 {

308 \_obj -= n;

309 return \*this;

310 }

311

312 template <typename T>

313 template <typename U>

314 inline U &Vector<T>::Iter<U>::operator\*() const { return (\*\_obj); }

315

316 template <typename T>

317 template <typename U>

318 inline Vector<T>::Iter<U>::Iter(U \*obj) : \_obj(obj) {}

319

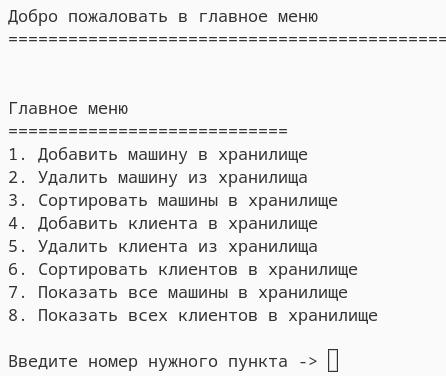
320 template <typename T>

321 template <typename U>

322 inline Vector<T>::Iter<U>::Iter(const U \*obj) : \_obj(obj) {}

323 *#endif // !VECTOR*

**Результат работы:**

****

**Вывод:** в ходе выполнения лабораторной работы были получены практические навыки работы с контейнерами средствами языка C++.