|  |  |
| --- | --- |
| 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 №6**

**«Исключения и обработка исключений»**

**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 и знаний по обработке исключительных ситуаций в логике и синтаксисе программы.

**Задачи:**

1. Познакомиться с типами ошибок;

2. Научиться обрабатывать ошибки при компиляции;

3. Познакомиться с концепцией исключений;

4. Научиться вызывать и обрабатывать исключения.

**Условие задачи:**

*Проанализируйте код и найдите места в программе, где возможны исключения (как минимум 3 вида ошибок), например: некорректный ввод пользователя, логически недопустимые значения полей или выход за границы допустимых значений типов, несовпадение типов, передаваемых в параметрах функции или математические ошибки, как возможное деление на ноль и т. д.*

*Используя механизм обработки исключений ликвидируйте такие места сделав код безопасным.*

*Протестируйте свою программу с различными категориями данными. Исправьте встречающиеся ошибки.*

**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

9 *#include* *<algorithm>*

10 *#include* *<exception>*

11 *#include* *<iostream>*

12

13 using namespace std;

14

15 *#pragma region функции-заглушки*

16

17 int testCar(int index)

18 {

19 using namespace SNS;

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

21 car.displayInfo();

22 return index;

23 }

24

25 int testEmployee(int index)

26 {

27 using namespace SNS;

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

29 employee.displayPublicInfo();

30 employee.displayPrivateInfo();

31 return index;

32 }

33

34 int testClient(int index)

35 {

36 using namespace SNS;

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

38 client.displayPublicInfo();

39 client.displayPrivateInfo();

40 return index;

41 }

42 *#pragma endregion*

43

44 void renderMain()

45 {

46 SNS::clearScreen();

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

48 << "============================================\n\n"

49 << endl;

50 }

51

52 namespace SNS

53 {

54 void displayCarsFromStorage()

55 {

56 try

57 {

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

59 for (const auto &car : cars)

60 {

61 cout << \*car << endl;

62 }

63

64 if (cars.empty())

65 {

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

67 }

68 }

69 catch (const std::exception &e)

70 {

71 std::cerr << "Ошибка при отображении машин из хранилища: " << e.what() << std::endl;

72 }

73 }

74

75 void displayClientsFromStorage()

76 {

77 try

78 {

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

80 for (const auto &client : clients)

81 {

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

83 }

84

85 if (clients.empty())

86 {

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

88 }

89 }

90 catch (const std::exception &e)

91 {

92 std::cerr << "Ошибка при отображении клиентов из хранилища: " << e.what() << std::endl;

93 }

94 }

95

96 void addCarToStorage()

97 {

98 try

99 {

100 Car car;

101 std::cin >> car;

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

103 }

104 catch (const std::exception &e)

105 {

106 std::cerr << "Ошибка при добавлении машины в хранилище: " << e.what() << std::endl;

107 }

108 }

109

110 void removeCarFromStorage()

111 {

112 displayCarsFromStorage();

113 if (!Storage::getStorage()->cars\_list.empty())

114 {

115 try

116 {

117 std::string brand;

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

119 std::cin >> brand;

120

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

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

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

124

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

126 {

127 cars.erase(it);

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

129 }

130 else

131 {

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

133 }

134 }

135 catch (const std::exception &e)

136 {

137 std::cerr << "Ошибка при удалении машины из хранилища: " << e.what() << std::endl;

138 }

139 }

140 }

141

142 void sortCarsInStorage()

143 {

144 try

145 {

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

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

148 { return a > b; });

149 }

150 catch (const std::exception &e)

151 {

152 std::cerr << "Ошибка при сортировке машин в хранилище: " << e.what() << std::endl;

153 }

154 }

155

156 void addClientToStorage()

157 {

158 try

159 {

160 Client client;

161 std::cin >> client;

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

163 }

164 catch (const std::exception &e)

165 {

166 std::cerr << "Ошибка при добавлении клиента в хранилище: " << e.what() << std::endl;

167 }

168 }

169

170 void removeClientFromStorage()

171 {

172 displayClientsFromStorage();

173 if (!Storage::getStorage()->users\_list.empty())

174 {

175 try

176 {

177 std::string login;

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

179 std::cin >> login;

180

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

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

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

184

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

186 {

187 clients.erase(it);

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

189 }

190 else

191 {

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

193 }

194 }

195 catch (const std::exception &e)

196 {

197 std::cerr << "Ошибка при удалении клиента из хранилища: " << e.what() << std::endl;

198 }

199 }

200 }

201

202 void sortClientsInStorage()

203 {

204 try

205 {

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

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

208 { return a < b; });

209 }

210 catch (const std::exception &e)

211 {

212 std::cerr << "Ошибка при сортировке клиентов в хранилище: " << e.what() << std::endl;

213 }

214 }

215

216 CMenu \*createMainMenu()

217 {

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

219 ItemList{

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

221 {addCarToStorage(); return index; }),

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

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

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

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

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

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

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

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

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

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

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

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

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

235 {displayClientsFromStorage(); return index; }),

236 CMenuItem("Выйти", [](int index) -> int

237 {return -1; })});

238 return menu;

239 }

240 }

241

242 int main()

243 {

244 using namespace SNS;

245

246 renderMain();

247

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

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

250

251 CMenu &menu = \*createMainMenu();

252

253 do

254 {

255 cout << menu;

256

257 try

258 {

259 cin >> menu;

260 clearScreen();

261 }

262 catch (const std::exception &e)

263 {

264 std::cerr << "Ошибка при взаимодействии с меню: " << e.what() << std::endl;

265 }

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

267

268 delete &menu;

269 return 0;

270 }

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* *<exception>*

3

4 namespace SNS

5 {

6 Car::Car()

7 {

8 }

9

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

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

12 {

13 if (brand.empty() || description.empty() || features.empty() || price <= 0 || year <= 0)

14 {

15 throw CarException("Некорректные данные автомобиля");

16 }

17 }

18

19 void Car::displayInfo()

20 {

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

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

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

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

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

26 }

27

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

29 {

30 return brand;

31 }

32

33 double Car::getPrice() const

34 {

35 return price;

36 }

37

38 int Car::getYear() const

39 {

40 return year;

41 }

42

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

44 {

45 return description;

46 }

47

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

49 {

50 return features;

51 }

52

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

54 {

55 if (brand.empty())

56 {

57 throw CarException("Недопустимое значение марки автомобиля");

58 }

59 this->brand = brand;

60 }

61

62 void Car::setPrice(double price)

63 {

64 if (price <= 0)

65 {

66 throw CarException("Недопустимое значение цены автомобиля");

67 }

68 this->price = price;

69 }

70

71 void Car::setYear(int year)

72 {

73 if (year <= 0)

74 {

75 throw CarException("Недопустимое значение года автомобиля");

76 }

77 this->year = year;

78 }

79

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

81 {

82 if (description.empty())

83 {

84 throw CarException("Недопустимое значение описания автомобиля");

85 }

86 this->description = description;

87 }

88

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

90 {

91 if (features.empty())

92 {

93 throw CarException("Недопустимое значение характеристик автомобиля");

94 }

95 this->features = features;

96 }

97

98 bool Car::operator<(const Car &other) const

99 {

100 return year < other.year;

101 }

102

103 bool Car::operator>(const Car &other) const

104 {

105 return year > other.year;

106 }

107

108 std::ostream &operator<<(std::ostream &out, const Car &car)

109 {

110 out << "Brand: " << car.brand << std::endl;

111 out << "Price: " << car.price << std::endl;

112 out << "Year: " << car.year << std::endl;

113 out << "Description: " << car.description << std::endl;

114 out << "Features: " << car.features << std::endl;

115 return out;

116 }

117

118 std::istream &operator>>(std::istream &in, Car &car)

119 {

120 std::string brand, description, features;

121 double price;

122 int year;

123

124 cout << "Enter brand: ";

125 in >> brand;

126 cout << "Enter price: ";

127 in >> price;

128 cout << "Enter year: ";

129 in >> year;

130 cout << "Enter description: ";

131 in >> description;

132 cout << "Enter features: ";

133 in >> features;

134

135 if (brand.empty() || description.empty() || features.empty() || price <= 0 || year <= 0)

136 {

137 throw CarException("Некорректные данные автомобиля");

138 }

139

140 car.setBrand(brand);

141 car.setPrice(price);

142 car.setYear(year);

143 car.setDescription(description);

144 car.setFeatures(features);

145

146 return in;

147 }

148 }

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() : User()

6 {

7 }

8

9 Client::~Client() {}

10

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

12 std::string password, std::string service)

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

14 {

15 if (name.empty() || surname.empty() || login.empty() || password.empty())

16 {

17 throw ClientException("Некорректные данные клиента");

18 }

19 }

20

21 void Client::displayPublicInfo() const

22 {

23 User::displayPublicInfo();

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

25 }

26

27 void Client::displayPrivateInfo() const

28 {

29 User::displayPrivateInfo();

30 }

31

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

33 {

34 return service;

35 }

36

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

38 {

39 this->service = service;

40 }

41

42 bool Client::operator<(const Client &other) const

43 {

44 return m\_name < other.getName();

45 }

46

47 bool Client::operator>(const Client &other) const

48 {

49 return m\_name > other.getName();

50 }

51

52 std::ostream &operator<<(std::ostream &out, const Client &client)

53 {

54 out << "Name: " << client.getName() << std::endl;

55 out << "Surname: " << client.getSurname() << std::endl;

56 out << "Age: " << client.getAge() << std::endl;

57 out << "Login: " << client.getLogin() << std::endl;

58 out << "Password: " << client.getPassword() << std::endl;

59 out << "Service: " << client.service << std::endl;

60 return out;

61 }

62

63 std::istream &operator>>(std::istream &in, Client &client)

64 {

65 std::string name, surname, login, password, service;

66 int age;

67

68 cout << "Enter name: ";

69 in >> name;

70 cout << "Enter surname: ";

71 in >> surname;

72 cout << "Enter age: ";

73 in >> age;

74 cout << "Enter login: ";

75 in >> login;

76 cout << "Enter password: ";

77 in >> password;

78 cout << "Enter service: ";

79 in >> service;

80

81 if (name.empty() || surname.empty() || login.empty() || password.empty())

82 {

83 throw ClientException("Некорректные данные клиента");

84 }

85

86 client.setName(name);

87 client.setSurname(surname);

88 client.setAge(age);

89 client.setLogin(login);

90 client.setPassword(password);

91 client.setService(service);

92

93 return in;

94 }

95 }

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 Employee::Employee() : User()

6 {

7 }

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

9 std::string password, std::string post)

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

11 {

12 }

13 Employee::~Employee(){}

14

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

16 {

17 return post;

18 }

19

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

21 {

22 this->post = post;

23 }

24

25 void Employee::displayPublicInfo() const

26 {

27 User::displayPublicInfo();

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

29 }

30

31 void Employee::displayPrivateInfo() const

32 {

33 User::displayPrivateInfo();

34 }

35

36 bool Employee::operator<(const Employee &employee){

37 return post < employee.post;

38 }

39 bool Employee::operator>(const Employee &employee){

40 return post > employee.post;

41 }

42

43 std::ostream &operator<<(std::ostream &out, const Employee &employee)

44 {

45 out << "Name: " << employee.getName() << std::endl;

46 out << "Surname: " << employee.getSurname() << std::endl;

47 out << "Age: " << employee.getAge() << std::endl;

48 out << "Login: " << employee.getLogin() << std::endl;

49 out << "Password: " << employee.getPassword() << std::endl;

50 out << "Post: " << employee.post << std::endl;

51 return out;

52 }

53

54 std::istream &operator>>(std::istream &in, Employee &employee)

55 {

56 std::string name, surname, login, password, post;

57 int age;

58

59 cout << "Enter name: ";

60 in >> name;

61 cout << "Enter surname: ";

62 in >> surname;

63 cout << "Enter age: ";

64 in >> age;

65 cout << "Enter login: ";

66 in >> login;

67 cout << "Enter password: ";

68 in >> password;

69 cout << "Enter post: ";

70 in >> post;

71

72 employee.setName(name);

73 employee.setSurname(surname);

74 employee.setAge(age);

75 employee.setLogin(login);

76 employee.setPassword(password);

77 employee.setPost(post);

78

79 return in;

80 }

81

82 }

**Storage.h**

1 *#include* *"Storage.h"*

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

3

4 Storage \*Storage::s\_storage{};

5 Storage::Storage(string root\_path){};

6

7 Storage &Storage::createStorage(string root\_path)

8 {

9 static Storage storage{root\_path};

10

11 s\_storage = &storage;

12

13 return storage;

14 }

15

16 Storage::~Storage()

17 {

18

19 for (auto \*pItem : users\_list)

20 {

21 delete pItem;

22 }

23

24 for (auto \*pItem : cars\_list)

25 {

26 delete pItem;

27 }

28 }

29

30 Storage \*Storage::getStorage()

31 {

32 try {

33 if (s\_storage == nullptr) throw SNS::StorageException("\nВызов getStore раньше чем createStore! Получен нулевой указатель!\n");

34 }

35 catch (SNS::StorageException & invalid\_store) {

36 std::cerr << invalid\_store.what();

37 }

38

39 return s\_storage;

40 }

**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*

**Tools.cpp**

1 *#include* *"Tools.h"*

2

3 namespace SNS

4 {

5

6 void clearScreen(){

7 system("clear");

8 }

9

10 string getEnteredString(string text, ValidateString validate, std::istream &in)

11 {

12 string console\_enter{};

13

14 while (true)

15 {

16 cout << text;

17 in >> console\_enter;

18

19 if (in.fail() || (validate && !validate(console\_enter)))

20 {

21 console\_enter.clear();

22 in.clear();

23 in.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

24 cout << "\n\nYou entered an incorrect value, please try again!\n\n";

25 continue;

26 }

27

28 break;

29 }

30

31 return console\_enter;

32 }

33

34 int getEnteredNum(string text, ValidateNum validate, std::istream &in)

35 {

36 int console\_enter{};

37

38 while (true)

39 {

40 cout << text;

41 in >> console\_enter;

42

43 if (cin.fail() || (validate && !validate(console\_enter)))

44 {

45 in.clear();

46 in.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

47 cout << "\n\nYou entered an incorrect value, please try again!\n\n";

48 continue;

49 }

50

51 break;

52 }

53

54 return console\_enter;

55 }

56

57 char getEnteredChar(string text, ValidateChar validate, std::istream &in)

58 {

59 char console\_enter{};

60

61 while (true)

62 {

63 cout << text;

64 in >> console\_enter;

65

66 if (cin.fail() || (validate && !validate(console\_enter)))

67 {

68 in.clear();

69 in.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

70 cout << "\n\nYou entered an incorrect value, please try again!\n\n";

71 continue;

72 }

73

74 break;

75 }

76

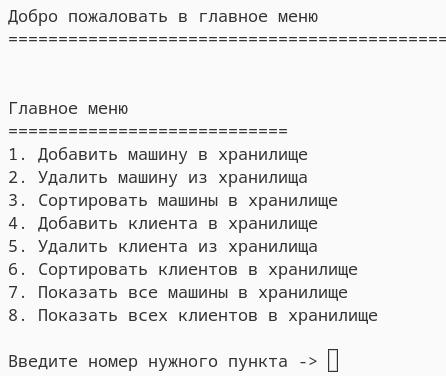
77 return console\_enter;

78 }

79

80 } *// namespace SNS*

**Результат работы:**

****

**Вывод:** в ходе выполнения лабораторной работы были получены практические навыки работы по обработке исключительных ситуаций в логике и синтаксиса программ средствами языка C++.