ПРИЛОЖЕНИЕ Г

(обязательное)

Исходный текст программы (с комментариями)

**TestingSystem.cpp**

1. #include "Menu.h"
2. #include <windows.h>
3. int main()
4. {
5. SetConsoleOutputCP(1251);
6. SetConsoleCP(1251);
7. Menu menu;
8. menu.displayRoleSelectionMenu();
9. }

**Menu.h**

1. #pragma once
2. #include <iostream>
3. #include "Test.h"
4. #include "SingleChoiceQuestion.h"
5. #include "OpenAnswerQuestion.h"
6. #include "MultipleChoiceQuestion.h"
7. #define MIN\_FULL\_NAME\_LENGTH 7
8. #define MAX\_FULL\_NAME\_LENGTH 25
9. #define MIN\_LAST\_NAME\_LENGTH 2
10. #define INITIALS\_LENGTH 4
11. #define MIN\_TEST\_QUESTIONS 3
12. using namespace std;
13. class Menu {
14. public:
15. Menu();
16. ~Menu();
17. void displayRoleSelectionMenu();
18. static int getValidatedNumber(const std::string& input, int min, int max);
19. static char getInput(const string& validOptions);
20. private:
21. void displayAdminMenu();
22. void displayStudentMenu(const string& studentName);
23. void checkStudentName(const string& studentName);
24. void startTesting(const string& studentName);
25. void displayTestList();
26. void addTest();
27. void deleteTest();
28. void editTest();
29. void editTestDetails(Test\* test);
30. bool saveTest(Test\* test);
31. void addQuestionsToTest(Test\* newTest);
32. void addSingleChoiceQuestion(Test\* test);
33. void addMultipleChoiceQuestion(Test\* test);
34. void addOpenAnswerQuestion(Test\* test);
35. void changeQuestion(SingleChoiceQuestion\* question);
36. void changeQuestion(MultipleChoiceQuestion\* question);
37. void changeQuestion(OpenAnswerQuestion\* question);
38. bool setQuestionTitle(Question\* question);
39. bool setQuestionOptions(ChoiceQuestion\* question);
40. bool handleExistingOption(ChoiceQuestion\* question, char& option, unsigned
41. int index);
42. bool replaceOption(ChoiceQuestion\* question, char option, unsigned int index);
43. bool handleNewOption(ChoiceQuestion\* question, char& option);
44. bool setAnswer(SingleChoiceQuestion\* question);
45. bool setAnswer(MultipleChoiceQuestion\* question);
46. bool setAnswer(OpenAnswerQuestion\* question);
47. void changeTestTitle(Test\* test);
48. void deleteQuestionFromTest(Test\* test);
49. void editQuestionInTest(Test\* test);
50. int printTestList();
51. void printQuestion(Question\* question);
52. void sortTests();
53. List<Test\*> tests;
54. };

**Menu.cpp**

1. #include "Menu.h"
2. #include <limits>
3. #include <string>
4. #include <cstdlib>
5. #include <functional>
6. #include "InputError.h"
7. #include "FileError.h"
8. #include "KeyVerifier.h"
9. #include "TestSession.h"
10. #include "StatisticMenu.h"
11. Menu::Menu()
12. {
13. tests = Test::loadAllTests();
14. }
15. Menu::~Menu()
16. {
17. for (auto test : tests) {
18. delete test;
19. }
20. tests.clear();
21. }
22. void Menu::displayRoleSelectionMenu()
23. {
24. char choice = 0;
25. cout << "Добро пожаловать!\n";
26. do {
27. cout << "Выберите вашу роль:\n";
28. cout << "1. Учитель/Администратор\n";
29. cout << "2. Ученик\n";
30. cout << "0. Выход\n";
31. try {
32. choice = getInput("012");
33. }
34. catch (const InputError& e) {
35. cout << e.what() << endl;
36. continue;
37. }
38. switch (choice) {
39. case '1':
40. for (;;) {
41. if (KeyVerifier::verify()) {
42. displayAdminMenu();
43. break;
44. }
45. cout << "Верификация не пройдена. Проверьте наличие ключа и

попробуйте снова.\n";

1. cout << "Введите 0 для выхода или любую другую клавишу для

повторной попытки: ";

1. string adminChoice;
2. getline(cin, adminChoice);
3. system("cls");
4. if (adminChoice == "0") {
5. break;
6. }
7. }
8. break;
9. case '2':
10. for (;;) {
11. cout << "Введите вашу фамилию и инициалы (Фамилия И.О.) или '0'

для возврата: ";

1. string studentName;
2. getline(cin, studentName);
3. system("cls");
4. if (studentName == "0") break;
5. try {
6. checkStudentName(studentName);
7. cout << "Добро пожаловать, " << studentName << "!\n";
8. displayStudentMenu(studentName);
9. break;
10. }
11. catch (InputError& e) {
12. cout << "Ошибка: " << e.what() << endl;
13. }
14. }
15. break;
16. }
17. } while (choice != '0');
18. cout << "Выход из программы...\n";
19. }
20. void Menu::displayAdminMenu()
21. {
22. char choice = 0;
23. do {
24. cout << "Меню учителя/администратора:\n";
25. cout << "1. Список тестов\n";
26. cout << "2. Добавить тест\n";
27. cout << "3. Удалить тест\n";
28. cout << "4. Редактировать тест\n";
29. cout << "5. Просмотр статистики\n";
30. cout << "0. Вернуться в меню выбора роли\n";
31. try {
32. choice = getInput("012345");
33. }
34. catch (const InputError& e) {
35. cout << e.what() << endl;
36. continue;
37. }
38. switch (choice) {
39. case '1':
40. displayTestList();
41. break;
42. case '2':
43. addTest();
44. break;
45. case '3':
46. deleteTest();
47. break;
48. case '4':
49. editTest();
50. break;
51. case '5':
52. {
53. auto statMenu = new StatisticMenu();
54. statMenu->start();
55. delete statMenu;
56. break;
57. }
58. }
59. } while (choice != '0');
60. }
61. void Menu::displayStudentMenu(const string& studentName)
62. {
63. char choice = 0;
64. do {
65. cout << "Меню ученика:\n";
66. cout << "1. Начать тестирование\n";
67. cout << "0. Вернуться в меню выбора роли\n";
68. try {
69. choice = getInput("01");
70. }
71. catch (const InputError& e) {
72. cout << e.what() << endl;
73. continue;
74. }
75. switch (choice) {
76. case '1':
77. startTesting(studentName);
78. break;
79. }
80. } while (choice != '0');
81. }
82. void Menu::startTesting(const string& studentName)
83. {
84. do {
85. int maxIndex = printTestList();
86. if (maxIndex == 0) {
87. cout << "Список тестов пуст.\n\nВведите любой символ для возврата...";
88. string input;
89. getline(cin, input);
90. system("cls");
91. return;
92. }
93. cout << "\nВведите номер теста для продолжения, либо 0 для возврата: ";
94. string input;
95. getline(cin, input);
96. system("cls");
97. if (input == "0") return;
98. try {
99. int selectedIndex = getValidatedNumber(input, 1, maxIndex);
100. auto session = new TestSession(tests[selectedIndex - 1], studentName);
101. if (!session->prepare()) {
102. delete session;
103. continue;
104. }
105. session->start();
106. delete session;
107. return;
108. }
109. catch (const InputError& e)
110. {
111. cout << "Ошибка: " << e.what() << "\n";
112. }
113. } while (true);
114. }
115. void Menu::displayTestList()
116. {
117. while (true) {
118. int maxIndex = printTestList();
119. if (maxIndex == 0) {
120. cout << "Список тестов пуст.\n\nВведите любой символ для возврата...";
121. string input;
122. getline(cin, input);
123. system("cls");
124. return;
125. }
126. cout << "\nВведите номер теста для просмотра, либо 0 для возврата: ";
127. string input;
128. getline(cin, input);
129. system("cls");
130. if (input == "0") return;
131. try {
132. int selectedIndex = getValidatedNumber(input, 1, maxIndex);
133. tests[selectedIndex - 1]->print();
134. cout << "\nВведите любой символ для возврата...";
135. string input;
136. getline(cin, input);
137. system("cls");
138. }
139. catch (const InputError& e)
140. {
141. cout << "Ошибка: " << e.what() << "\n";
142. }
143. }
144. }
145. void Menu::addTest()
146. {
147. cout << "Добавление нового теста\n";
148. do {
149. cout << "Введите название теста (или 0 для отмены): ";
150. string testTitle;
151. getline(cin, testTitle);
152. system("cls");
153. if (testTitle == "0") return;
154. try {
155. Test\* newTest = new Test(testTitle);
156. do {
157. addQuestionsToTest(newTest);
158. if (newTest->getQuestionCount() >= MIN\_TEST\_QUESTIONS) {
159. if (saveTest(newTest)) {
160. tests.pushBack(newTest);
161. sortTests();
162. cout << "Тест успешно сохранен.\n";
163. }
164. else {
165. delete newTest;
166. cout << "Тест не был сохранен.\n";
167. }
168. return;
169. }
170. else {
171. cout << "Тест содержит недостаточно вопросов.\n";
172. cout << "Введите любой символ для продолжения (или 0 для

выхода без сохранения):\n";

1. string userInput;
2. getline(cin, userInput);
3. system("cls");
4. if (userInput == "0") {
5. return;
6. }
7. }
8. } while (true);
9. return;
10. }
11. catch (const InputError& e) {
12. cout << "Ошибка: " << e.what() << "\n";
13. }
14. } while (true);
15. }
16. bool Menu::saveTest(Test\* test)
17. {
18. do {
19. try {
20. test->saveToFile();
21. return true;
22. }
23. catch (const FileError& e) {
24. cout << "Ошибка: " << e.what() << endl;
25. cout << "Введите 0 для выхода без сохранения или любой другой символ

для повторной попытки: ";

1. string input;
2. getline(cin, input);
3. system("cls");
4. if (input == "0") {
5. return false;
6. }
7. }
8. } while (true);
9. }
10. void Menu::deleteTest()
11. {
12. while (true) {
13. int maxIndex = printTestList();
14. if (maxIndex == 0) {
15. cout << "Нет тестов для удаления.\n\nВведите любой символ для

возврата...";

1. string input;
2. getline(cin, input);
3. system("cls");
4. return;
5. }
6. cout << "\nВведите номер теста для удаления (или 0 для отмены): ";
7. string input;
8. getline(cin, input);
9. if (input == "0") {
10. system("cls");
11. return;
12. }
13. try {
14. int selectedIndex = getValidatedNumber(input, 1, maxIndex);
15. cout << "Вы действительно хотите удалить тест \"" <<
16. tests[selectedIndex - 1]->getTitle()
17. << "\"?\n(Введите Y для подтверждения или любой другой символ для

отмены): ";

1. getline(cin, input);
2. system("cls");
3. if (input != "Y" && input != "y") {
4. cout << "Тест не был удалён.\n";
5. continue;
6. }
7. tests[selectedIndex - 1]->deleteTestFile();
8. tests.removeAt(selectedIndex - 1);
9. cout << "Тест успешно удалён!\n";
10. }
11. catch (const InputError& e)
12. {
13. system("cls");
14. cout << "Ошибка: " << e.what() << "\n";
15. }
16. catch (const FileError& e) {
17. cout << "Ошибка: " << e.what() << "\n";
18. return;
19. }
20. }
21. }
22. void Menu::editTest()
23. {
24. while (true) {
25. int maxIndex = printTestList();
26. if (maxIndex == 0) {
27. cout << "Нет тестов для изменения.\n\nВведите любой символ для

возврата...";

1. string input;
2. getline(cin, input);
3. system("cls");
4. return;
5. }
6. cout << "\nВведите номер теста для изменения (или 0 для отмены): ";
7. string input;
8. getline(cin, input);
9. system("cls");
10. if (input == "0") return;
11. try {
12. int selectedIndex = getValidatedNumber(input, 1, maxIndex);
13. auto test = tests[selectedIndex - 1];
14. editTestDetails(test);
15. break;
16. }
17. catch (const InputError& e)
18. {
19. cout << "Ошибка: " << e.what() << "\n";
20. }
21. }
22. }
23. void Menu::editTestDetails(Test\* test)
24. {
25. auto changedTest = test;
26. char choice = 0;
27. while (true) {
28. cout << "Редактирование теста \"" << test->getTitle() << "\":\n";
29. cout << "1. Изменить название теста\n";
30. cout << "2. Добавить новые вопросы\n";
31. cout << "3. Удалить вопрос\n";
32. cout << "4. Изменить существующий вопрос\n";
33. cout << "0. Вернуться в меню администратора\n";
34. try {
35. choice = getInput("01234");
36. switch (choice) {
37. case '1': {
38. changeTestTitle(test);
39. break;
40. }
41. case '2':
42. addQuestionsToTest(test);
43. break;
44. case '3':
45. deleteQuestionFromTest(test);
46. break;
47. case '4':
48. editQuestionInTest(test);
49. break;
50. case '0':
51. {
52. if (changedTest->getQuestionCount() >= MIN\_TEST\_QUESTIONS) {
53. if (saveTest(changedTest)) {
54. test = changedTest;
55. sortTests();
56. cout << "Тест успешно сохранен.\n";
57. }
58. else {
59. delete changedTest;
60. cout << "Тест не был сохранен.\n";
61. }
62. return;
63. }
64. else {
65. cout << "Тест содержит недостаточно вопросов.\n";
66. cout << "Введите любой символ для продолжения (или 0 для

выхода без сохранения):\n";

1. string userInput;
2. getline(cin, userInput);
3. system("cls");
4. if (userInput == "0") {
5. return;
6. }
7. }
8. break;
9. }
10. }
11. }
12. catch (const InputError& e) {
13. cout << "Ошибка: " << e.what() << "\n";
14. }
15. }
16. }
17. void Menu::addQuestionsToTest(Test\* newTest)
18. {
19. char choice;
20. do {
21. cout << "Добавление нового вопроса:\n";
22. cout << "1. Вопрос с одним правильным ответом\n";
23. cout << "2. Вопрос с несколькими правильными ответами\n";
24. cout << "3. Вопрос со свободным ответом\n";
25. cout << "0. Завершить добавление вопросов\n";
26. try {
27. choice = getInput("0123");
28. }
29. catch (const InputError& e) {
30. cout << "Ошибка: " << e.what() << endl;
31. continue;
32. }
33. switch (choice) {
34. case '1': {
35. addSingleChoiceQuestion(newTest);
36. break;
37. }
38. case '2': {
39. addMultipleChoiceQuestion(newTest);
40. break;
41. }
42. case '3': {
43. addOpenAnswerQuestion(newTest);
44. break;
45. }
46. case '0':
47. return;
48. }
49. } while (true);
50. }
51. void Menu::addSingleChoiceQuestion(Test\* test)
52. {
53. SingleChoiceQuestion\* question = new SingleChoiceQuestion();
54. cout << "Добавление вопроса с одним правильным ответом\n";
55. if (!setQuestionTitle(question) || !setQuestionOptions(question)

|| !setAnswer(question)) {

1. delete question;
2. cout << "Вопрос не был добавлен!\n";
3. return;
4. }
5. test->addQuestion(question);
6. cout << "Вопрос успешно добавлен!\n";
7. }
8. void Menu::addMultipleChoiceQuestion(Test\* test)
9. {
10. MultipleChoiceQuestion\* question = new MultipleChoiceQuestion();
11. cout << "Добавление вопроса с несколькими правильным ответом\n";
12. if (!setQuestionTitle(question) || !setQuestionOptions(question) || !setAnswer(question)) {
13. delete question;
14. cout << "Вопрос не был добавлен!\n";
15. return;
16. }
17. test->addQuestion(question);
18. cout << "Вопрос успешно добавлен!\n";
19. }
20. void Menu::addOpenAnswerQuestion(Test\* test)
21. {
22. OpenAnswerQuestion\* question = new OpenAnswerQuestion();
23. cout << "Добавление вопроса со свободным ответом\n";
24. if (!setQuestionTitle(question) || !setAnswer(question)) {
25. delete question;
26. cout << "Вопрос не был добавлен!\n";
27. return;
28. }
29. test->addQuestion(question);
30. cout << "Вопрос успешно добавлен!\n";
31. }
32. void Menu::changeTestTitle(Test\* test)
33. {
34. while (true) {
35. cout << "Текущее название теста: \"" << test->getTitle() << "\"\n";
36. cout << "Введите новое название теста (или 0 для отмены): ";
37. string newTitle;
38. getline(cin, newTitle);
39. system("cls");
40. if (newTitle == "0") return;
41. try {
42. test->setTitle(newTitle);
43. cout << "Название теста успешно изменено.\n";
44. return;
45. }
46. catch (const InputError& e) {
47. cout << "Ошибка: " << e.what() << "\n";
48. }
49. }
50. }
51. void Menu::deleteQuestionFromTest(Test\* test)
52. {
53. while (true) {
54. if (test->getQuestionCount() == 0) {
55. cout << "В данном тесте нет вопросов для удаления.\n\nВведите любой

символ для возврата...";

1. string input;
2. getline(cin, input);
3. system("cls");
4. return;
5. }
6. cout << "Список вопросов теста \"" << test->getTitle() << "\":\n";
7. for (int i = 0; i < test->getQuestionCount(); ++i) {
8. cout << i + 1 << ". " << test->getQuestion(i)->getTitle() << "\n";
9. }
10. cout << "\nВведите номер вопроса для удаления (или 0 для отмены): ";
11. string input;
12. getline(cin, input);
13. if (input == "0") {
14. system("cls");
15. return;
16. }
17. try {
18. int selectedIndex = getValidatedNumber(input, 1, test-

>getQuestionCount());

1. cout << "Вы действительно хотите удалить вопрос \"" << test-

>getQuestion(selectedIndex - 1)->getTitle()

1. << "\"?\n(Введите Y для подтверждения или любой другой символ для
2. отмены): ";
3. getline(cin, input);
4. system("cls");
5. if (input != "Y" && input != "y") {
6. cout << "Вопрос не был удалён.\n";
7. continue;
8. }
9. test->removeQuestion(selectedIndex - 1);
10. cout << "Вопрос успешно удалён!\n";
11. }
12. catch (const InputError& e) {
13. system("cls");
14. cout << "Ошибка: " << e.what() << "\n";
15. }
16. }
17. }
18. void Menu::editQuestionInTest(Test\* test)
19. {
20. while (true) {
21. if (test->getQuestionCount() == 0) {
22. cout << "В данном тесте нет вопросов для редактирования.\n\nВведите

любой символ для возврата...";

1. string input;
2. getline(cin, input);
3. system("cls");
4. return;
5. }
6. cout << "Список вопросов теста \"" << test->getTitle() << "\":\n";
7. for (int i = 0; i < test->getQuestionCount(); ++i) {
8. cout << i + 1 << ". " << test->getQuestion(i)->getTitle() << "\n";
9. }
10. cout << "\nВведите номер вопроса для редактирования (или 0 для отмены): ";
11. string input;
12. getline(cin, input);
13. system("cls");
14. if (input == "0") {
15. return;
16. }
17. try {
18. int selectedIndex = getValidatedNumber(input, 1, test-

>getQuestionCount());

1. Question\* question = test->getQuestion(selectedIndex - 1);
2. if (auto singleChoiceQuestion =

dynamic\_cast<SingleChoiceQuestion\*>(question)) {

1. changeQuestion(singleChoiceQuestion);
2. }
3. else if (auto multipleChoiceQuestion =

dynamic\_cast<MultipleChoiceQuestion\*>(question)) {

1. changeQuestion(multipleChoiceQuestion);
2. }
3. else if (auto openAnswerQuestion =

dynamic\_cast<OpenAnswerQuestion\*>(question)) {

1. changeQuestion(openAnswerQuestion);
2. }
3. else {
4. cout << "Ошибка: тип вопроса не поддерживается.\n";
5. }
6. }
7. catch (const InputError& e) {
8. cout << "Ошибка: " << e.what() << "\n";
9. }
10. }
11. }
12. void Menu::changeQuestion(SingleChoiceQuestion\* question)
13. {
14. while (true) {
15. cout << "Редактирование вопроса \"" << question->getTitle() << "\":\n";
16. cout << "1. Изменить заголовок\n";
17. cout << "2. Изменить варианты ответа\n";
18. cout << "3. Изменить правильный ответ\n";
19. cout << "4. Показать вопрос\n";
20. cout << "0. Вернуться\n";
21. char choice = getInput("01234");
22. try {
23. switch (choice) {
24. case '1':
25. setQuestionTitle(question);
26. break;
27. case '2':
28. {
29. auto changedQuestion = new SingleChoiceQuestion(\*question);
30. setQuestionOptions(changedQuestion);
31. if (changedQuestion->getOptionCount() != question-

>getOptionCount()) {

1. if (!setAnswer(changedQuestion)) {
2. delete changedQuestion;
3. break;
4. }
5. }
6. \*question = \*changedQuestion;
7. delete changedQuestion;
8. break;
9. }
10. case '3':
11. setAnswer(question);
12. break;
13. case '4':
14. printQuestion(question);
15. break;
16. case '0':
17. return;
18. }
19. }
20. catch (const InputError& e) {
21. cout << "Ошибка: " << e.what() << "\n";
22. }
23. }
24. }
25. void Menu::changeQuestion(MultipleChoiceQuestion\* question)
26. {
27. while (true) {
28. cout << "Редактирование вопроса \"" << question->getTitle() << "\":\n";
29. cout << "1. Изменить заголовок\n";
30. cout << "2. Изменить варианты ответа\n";
31. cout << "3. Изменить правильные ответы\n";
32. cout << "4. Показать вопрос\n";
33. cout << "0. Вернуться\n";
34. char choice = getInput("01234");
35. try {
36. switch (choice) {
37. case '1':
38. setQuestionTitle(question);
39. break;
40. case '2':
41. {
42. auto changedQuestion = new MultipleChoiceQuestion(\*question);
43. setQuestionOptions(changedQuestion);
44. if (changedQuestion->getOptionCount() != question-

>getOptionCount()) {

1. if (!setAnswer(changedQuestion)) {
2. delete changedQuestion;
3. break;
4. }
5. }
6. \*question = \*changedQuestion;
7. delete changedQuestion;
8. break;
9. }
10. case '3':
11. setAnswer(question);
12. break;
13. case '4':
14. printQuestion(question);
15. break;
16. case '0':
17. return;
18. }
19. }
20. catch (const InputError& e) {
21. cout << "Ошибка: " << e.what() << "\n";
22. }
23. }
24. }
25. void Menu::changeQuestion(OpenAnswerQuestion\* question)
26. {
27. while (true) {
28. cout << "Редактирование вопроса \"" << question->getTitle() << "\":\n";
29. cout << "1. Изменить заголовок\n";
30. cout << "2. Изменить ответ\n";
31. cout << "3. Показать вопрос\n";
32. cout << "0. Вернуться\n";
33. char choice = getInput("0123");
34. try {
35. switch (choice) {
36. case '1':
37. setQuestionTitle(question);
38. break;
39. case '2':
40. setAnswer(question);
41. break;
42. case '3':
43. printQuestion(question);
44. break;
45. case '0':
46. return;
47. }
48. }
49. catch (const InputError& e) {
50. cout << "Ошибка: " << e.what() << "\n";
51. }
52. }
53. }
54. char Menu::getInput(const string& validOptions) {
55. string input;
56. cout << "Введите выбор: ";
57. getline(cin, input);
58. system("cls");
59. if (input.length() == 1) {
60. if (validOptions.find(tolower(input[0])) == string::npos) throw

InputError("Некорректный ввод. Операции (" + input + ") не существует.");

1. return input[0];
2. }
3. throw InputError("Некорректный ввод. Введите один символ.");
4. }
5. void Menu::checkStudentName(const string& studentName) {
6. if (studentName.size() < MIN\_FULL\_NAME\_LENGTH || studentName.size() >

MAX\_FULL\_NAME\_LENGTH) {

1. throw InputError("Некорректный формат ФИО. Длина должна быть от "
2. + std::to\_string(MIN\_FULL\_NAME\_LENGTH) + " до "
3. + std::to\_string(MAX\_FULL\_NAME\_LENGTH)
4. + " символов. Пример: Иванов И.И.");
5. }
6. size\_t spacePos = studentName.find(' ');
7. if (spacePos == string::npos || spacePos == 0 || spacePos ==

studentName.size() - 1) {

1. throw InputError("Некорректный формат ФИО. Пример: Иванов И.И.");
2. }
3. string lastName = studentName.substr(0, spacePos);
4. string initials = studentName.substr(spacePos + 1);
5. if (lastName.empty() || ((lastName[0] < 'А' || lastName[0] > 'Я') &&

lastName[0] != 'Ё')) {

1. throw InputError("Фамилия должна начинаться с заглавной буквы.");
2. }
3. if (lastName.size() < MIN\_LAST\_NAME\_LENGTH) {
4. throw InputError("Фамилия слишком короткая.");
5. }
6. for (size\_t i = 1; i < lastName.size(); ++i) {
7. if ((lastName[i] < 'а' || lastName[i] > 'я') && lastName[i] != 'ё') {
8. throw InputError("Фамилия должна содержать только русские буквы и

соответствовать формату \"Фамилия\"");

1. }
2. }
3. if (initials.size() != INITIALS\_LENGTH || initials[1] != '.' ||

initials[3] != '.' ||

1. ((initials[0] < 'А' || initials[0] > 'Я') && initials[0] != 'Ё') ||
2. ((initials[2] < 'А' || initials[2] > 'Я') && initials[2] != 'Ё')) {
3. throw InputError("Инициалы должны быть в формате И.О. с заглавными

русскими буквами.");

1. }
2. }
3. bool Menu::setQuestionTitle(Question\* question) {
4. auto currentTitle = question->getTitle();
5. do {
6. if (!currentTitle.empty()) {
7. cout << "Текущий заголовок: \"" << question->getTitle() << "\"\n";
8. }
9. cout << "Введите " << (!currentTitle.empty() ? "новый " : "") <<

"заголовок вопроса (или 0 для отмены): ";

1. string questionText;
2. getline(cin, questionText);
3. system("cls");
4. if (questionText == "0") {
5. return false;
6. }
7. try {
8. question->setTitle(questionText);
9. return true;
10. }
11. catch (const InputError& e) {
12. cout << "Ошибка: " << e.what() << "\n";
13. }
14. } while (true);
15. }
16. bool Menu::setQuestionOptions(ChoiceQuestion\* question) {
17. char option = 'A';
18. while (option <= 'Z') {
19. int index = option - 'A';
20. if (index < question->getOptionCount()) {
21. if (!handleExistingOption(question, option, index)) {
22. return true;
23. }
24. }
25. else {
26. int startCount = question->getOptionCount();
27. if (!handleNewOption(question, option)) {
28. return false;
29. }
30. if (startCount == question->getOptionCount()) {
31. return true;
32. }
33. }
34. }
35. return true;
36. }
37. bool Menu::handleExistingOption(ChoiceQuestion\* question, char& option, unsigned

int index) {

1. while (true) {
2. cout << "Текущий вариант ответа " << option << ": " << question-

>getOption(index) << "\n";

1. cout << "Выберите действие:\n";
2. cout << "1. Заменить\n";
3. cout << "2. Удалить\n";
4. cout << "3. Пропустить\n";
5. cout << "0. Завершить изменение вариантов\n";
6. char choice;
7. try {
8. choice = getInput("0123");
9. }
10. catch (const InputError& e) {
11. cout << "Ошибка: " << e.what() << "\n";
12. continue;
13. }
14. switch (choice) {
15. case '1':
16. if (replaceOption(question, option, index)) {
17. cout << "Вариант ответа " << option << " успешно изменен.\n";
18. option++;
19. return true;
20. }
21. continue;
22. case '2':
23. question->removeOption(index);
24. cout << "Вариант ответа " << option << " успешно удалён.\n";
25. return true;
26. case '3':
27. cout << "Вариант ответа " << option << " пропущен.\n";
28. option++;
29. return true;
30. case '0':
31. return false;
32. }
33. }
34. }
35. bool Menu::replaceOption(ChoiceQuestion\* question, char option, unsigned int

index) {

1. while (true) {
2. cout << "Введите новый текст для варианта " << option << " (или оставьте

пустым для отмены): ";

1. string newOptionText;
2. getline(cin, newOptionText);
3. system("cls");
4. if (newOptionText.empty()) {
5. return false;
6. }
7. try {
8. question->addOption(newOptionText, index);
9. question->removeOption(index + 1);
10. return true;
11. }
12. catch (const InputError& e) {
13. cout << "Ошибка: " << e.what() << "\n";
14. }
15. }
16. }
17. bool Menu::handleNewOption(ChoiceQuestion\* question, char& option) {
18. while (true) {
19. cout << "Введите вариант ответа " << option << " (или оставьте пустым для

завершения): ";

1. string optionText;
2. getline(cin, optionText);
3. system("cls");
4. if (optionText.empty()) {
5. if (question->getOptionCount() < 2) {
6. cout << "Ошибка: Минимум два варианта ответа обязательно.\n";
7. cout << "Введите любой символ для продолжения (или 0 для выхода

без сохранения): ";

1. string userInput;
2. getline(cin, userInput);
3. system("cls");
4. if (userInput == "0") {
5. return false;
6. }
7. continue;
8. }
9. else {
10. return true;
11. }
12. }
13. try {
14. question->addOption(optionText);
15. option++;
16. return true;
17. }
18. catch (const InputError& e) {
19. cout << "Ошибка: " << e.what() << "\n";
20. }
21. }
22. }
23. bool Menu::setAnswer(SingleChoiceQuestion\* question) {
24. while (true) {
25. cout << "Вопрос: ";
26. question->printQuestion();
27. cout << "Введите правильный вариант ответа (A-" << static\_cast<char>('A'

+ question->getOptionCount() - 1)

1. << ") или 0 для отмены: ";
2. string input;
3. getline(cin, input);
4. system("cls");
5. if (input.size() != 1) {
6. cout << "Ошибка: Ответ должен состоять из одного символа\n";
7. }
8. if (input == "0") {
9. return false;
10. }
11. try {
12. char correctOption = input[0];
13. question->setCorrectIndex(correctOption);
14. break;
15. }
16. catch (const InputError& e) {
17. cout << "Ошибка: " << e.what() << "\n";
18. }
19. }
20. return true;
21. }
22. bool Menu::setAnswer(MultipleChoiceQuestion\* question) {
23. while (true) {
24. cout << "Вопрос: ";
25. question->printQuestion();
26. cout << "Введите правильные варианты ответа (A-" << static\_cast<char>('A'

+ question->getOptionCount() - 1)

1. << "), например \"AB\", или 0 для отмены: ";
2. string input;
3. getline(cin, input);
4. system("cls");
5. if (input == "0") {
6. return false;
7. }
8. List<unsigned char> correctOptions;
9. try {
10. for (unsigned char ch : input) {
11. correctOptions.pushBack(toupper(ch));
12. }
13. question->setCorrectIndices(correctOptions);
14. break;
15. }
16. catch (const InputError& e) {
17. cout << "Ошибка: " << e.what() << "\n";
18. }
19. }
20. return true;
21. }
22. bool Menu::setAnswer(OpenAnswerQuestion\* question) {
23. while (true) {
24. cout << "Вопрос: ";
25. question->printQuestion();
26. cout << "Введите правильный ответ, или оставьте пустым для отмены: ";
27. string input;
28. getline(cin, input);
29. system("cls");
30. if (input.empty()) {
31. return false;
32. }
33. try {
34. question->setAnswer(input);
35. break;
36. }
37. catch (const InputError& e) {
38. cout << "Ошибка: " << e.what() << "\n";
39. }
40. }
41. return true;
42. }
43. int Menu::getValidatedNumber(const std::string& input, int min = 0, int max =
44. 100) {
45. try {
46. int number = std::stoi(input);
47. if (number < min || number > max) {
48. throw InputError("Число должно быть в диапазоне от " +
49. std::to\_string(min) + " до " + std::to\_string(max) + ".");
50. }
51. return number;
52. }
53. catch (const std::invalid\_argument&) {
54. throw InputError("Некорректный формат числа.");
55. }
56. catch (const std::out\_of\_range&) {
57. throw InputError("Введённое число слишком велико.");
58. }
59. }
60. int Menu::printTestList() {
61. if (tests.isEmpty()) {
62. return 0;
63. }
64. cout << "Список тестов:\n";
65. int maxIndex = 0;
66. for (auto test : tests) {
67. cout << ++maxIndex << ". " << test->getTitle() << "\n";
68. }
69. return maxIndex;
70. }
71. void Menu::printQuestion(Question\* question) {
72. cout << "Вопрос: ";
73. question->printQuestion();;
74. question->printAnswer();
75. cout << "\nВведите любой символ для возврата...";
76. string input;
77. getline(cin, input);
78. system("cls");
79. }
80. void Menu::sortTests() {
81. int n = tests.getSize();
82. std::function<void(int, int)> heapify;
83. heapify = [&](int n, int i) {
84. int largest = i;
85. int left = 2 \* i + 1;
86. int right = 2 \* i + 2;
87. if (left < n && tests[left]->getTitle() > tests[largest]->getTitle())
88. {
89. largest = left;
90. }
91. if (right < n && tests[right]->getTitle() > tests[largest]->getTitle())
92. {
93. largest = right;
94. }
95. if (largest != i) {
96. std::swap(tests[i], tests[largest]);
97. heapify(n, largest);
98. }
99. };
100. for (int i = n / 2 - 1; i >= 0; --i) {
101. heapify(n, i);
102. }
103. for (int i = n - 1; i > 0; --i) {
104. std::swap(tests[0], tests[i]);
105. heapify(i, 0);
106. }
107. }

**TestSession.h**

1. #pragma once
2. #include "Test.h"
3. #include "InputError.h"
4. #include <iostream>
5. #include <string>
6. #include <cstdlib>
7. class TestSession {
8. public:
9. TestSession(Test\* test, std::string name);
10. void start();
11. bool prepare();
12. private:
13. void askQuestion(Question\* question, int number);
14. void showResult();
15. Test\* test;
16. std::string name;
17. List<std::string> userAnswers;
18. int correctAnswers;
19. int score;
20. };

**TestSession.cpp**

1. #include "TestSession.h"
2. #include "Statistic.h"
3. TestSession::TestSession(Test\* test, std::string name): name(name)
4. {
5. this->test = new Test(\*test);
6. userAnswers = {};
7. correctAnswers = 0;
8. score = 0;
9. }
10. bool TestSession::prepare()
11. {
12. if (!test || name.empty()) {
13. std::cout << "Необходимо указать тест и имя тестируемого.\n";
14. return false;
15. }
16. std::cout << "Тест: " << test->getTitle();
17. std::cout << "\nКоличество вопросов: " << test->getQuestionCount() << "\n";
18. std::cout << "\nЖелаете начать тест?\n";
19. std::cout << "(Введите Y для подтверждения или любой другой символ для

отмены): ";

1. std::string input;
2. getline(std::cin, input);
3. system("cls");
4. if (input != "Y" && input != "y") {
5. return false;
6. }
7. test->shuffleQuestions();
8. correctAnswers = 0;
9. userAnswers.clear();
10. return true;
11. }
12. void TestSession::start()
13. {
14. auto stat = new Statistic(name, test);
15. int max = test->getQuestionCount();
16. for (int i = 0; i < max; ++i) {
17. Question\* question = test->getQuestion(i);
18. askQuestion(question, i + 1);
19. }
20. stat->setAnswers(userAnswers);
21. stat->setCorrectAnswersCount(correctAnswers);
22. stat->saveToFile();
23. delete stat;
24. showResult();
25. }
26. void TestSession::askQuestion(Question\* question, int number)
27. {
28. do {
29. std::cout << "Вопрос " << number << ": ";
30. question->printQuestion();
31. switch (question->getType()) {
32. case QuestionType::SingleChoice:
33. std::cout << "Введите правильный вариант ответа ";
34. break;
35. case QuestionType::MultipleChoice:
36. std::cout << "Введите правильные варианты ответа ";
37. break;
38. case QuestionType::OpenAnswer:
39. std::cout << "Введите ответ ";
40. break;
41. default:
42. throw InputError("Вопрос имеет неизвестный тип");
43. }
44. std::cout << "(либо введите пустым для пропуска): ";
45. std::string input;
46. getline(std::cin, input);
47. if (input.empty()) {
48. std::cout << "\nВы уверены, что хотите пропустить вопрос?\n";
49. std::cout << "!!! Вы не сможете вернуться к этому вопросу !!!\n";
50. std::cout << "(Введите Y для подтверждения или любой другой символ

для отмены): ";

1. getline(std::cin, input);
2. system("cls");
3. if (input != "Y" && input != "y") continue;
4. userAnswers.pushBack(SKIP);
5. return;
6. }
7. system("cls");
8. try {
9. bool result = question->checkAnswer(input);
10. if (result) correctAnswers++;
11. userAnswers.pushBack(input);
12. return;
13. }
14. catch (const InputError& e) {
15. std::cout << "Ошибка: " << e.what() << std::endl;
16. }
17. } while (true);
19. }
20. void TestSession::showResult()
21. {
22. score = (int)std::round(((double)(correctAnswers) / test->getQuestionCount())

\* 100.0);

1. std::cout << "Тест завершён! Ваш результат: " << correctAnswers << " из "
2. << test->getQuestionCount() << " (" << score << "%).\n";
3. std::cout << "\nВведите любой символ для возврата...";
4. std::string input;
5. getline(std::cin, input);
6. system("cls");
7. }

**Statistic.h**

1. #pragma once
2. #include "Test.h"
3. #include <string>
4. #include <vector>
5. #include <chrono>
6. #define SKIP "Пропущен"
7. #define STAT\_FILE "stat.dat"
8. class Statistic {
9. public:
10. Statistic(const std::string name, Test\* test);
11. void setCurrentTime();
12. void setAnswers(const List<std::string>& answers);
13. void setTime(const std::string time);
14. void setCorrectAnswersCount(int count);
15. std::string getTestTitle();
16. std::string getUserName();
17. std::string getTime();
18. void print();
19. void saveToFile();
20. static Statistic\* loadFromStream(std::ifstream& in);
21. static List<Statistic\*> loadAllFromFile();
22. private:
23. std::string name;
24. std::string time;
25. List<std::string> answers;
26. int correctAnswersCount;
27. int score;
28. Test\* test;
29. };

**Statistic.сpp**

1. #include "Statistic.h"
2. #include "FileError.h"
3. #include <sstream>
4. #include <iomanip>
5. #include <chrono>
6. #include <ctime>
7. #include <fstream>
8. Statistic::Statistic(const std::string name, Test\* test)
9. : name(name) {
10. this->test = new Test(\*test);
11. setCurrentTime();
12. answers = {};
13. score = 0;
14. correctAnswersCount = 0;
15. }
16. void Statistic::setAnswers(const List<std::string>& answers)
17. {
18. this->answers = answers;
19. }
20. void Statistic::setCorrectAnswersCount(int count)
21. {
22. correctAnswersCount = count;
23. score = (int)std::round(((double)(correctAnswersCount) / test-
24. >getQuestionCount()) \* 100.0);
25. }
26. void Statistic::setTime(const std::string time)
27. {
28. this->time = time;
29. }
30. void Statistic::setCurrentTime()
31. {
32. auto now = std::chrono::system\_clock::now();
33. auto nowTimeT = std::chrono::system\_clock::to\_time\_t(now);
34. std::tm localTime;
35. localtime\_s(&localTime, &nowTimeT);
36. std::ostringstream oss;
37. oss << std::put\_time(&localTime, "%d.%m.%y %H:%M");
38. this->time = oss.str();
39. }
40. std::string Statistic::getTestTitle()
41. {
42. return test->getTitle();
43. }
44. std::string Statistic::getUserName()
45. {
46. return name;
47. }
48. std::string Statistic::getTime()
49. {
50. return time;
51. }
52. void Statistic::print()
53. {
54. std::cout << "Имя тестируемого: " << name << "\n";
55. std::cout << "Название теста: " << test->getTitle() << "\n";
56. std::cout << "Дата и время прохождения: "<< time << "\n";
57. std::cout << "===============================";
58. for (int i = 0; i < answers.getSize(); ++i) {
59. std::cout << "\nВопрос " << (i + 1) << ": ";
60. test->getQuestion(i)->printQuestion();
61. if (answers[i] == SKIP) {
62. std::cout << "Вопрос был пропущен\n";
63. test->getQuestion(i)->printAnswer();
64. }
65. else {
66. std::cout << "Ответ: " << answers[i];
67. if (test->getQuestion(i)->checkAnswer(answers[i])) {
68. std::cout << " [V]\n";
69. }
70. else {
71. std::cout << " [X]\n";
72. test->getQuestion(i)->printAnswer();
73. }
74. }
76. }
77. std::cout << "===============================\n";
78. std::cout << "Результат: " << correctAnswersCount << " из "
79. << test->getQuestionCount() << " (" << score << "%).\n";
80. std::cout << "\nВведите любой символ для возврата...";
81. std::string input;
82. getline(std::cin, input);
83. system("cls");
84. }
85. void Statistic::saveToFile()
86. {
87. std::ofstream out(STAT\_FILE, std::ios::binary | std::ios::app);
88. if (!out.is\_open()) {
89. std::string errorTest = "Не удалось открыть файл для записи: ";
90. throw FileError(errorTest + STAT\_FILE);
91. }
92. test->saveToStream(out);
93. size\_t nameSize = name.size();
94. out.write(reinterpret\_cast<const char\*>(&nameSize), sizeof(nameSize));
95. out.write(name.data(), nameSize);
96. size\_t timeSize = time.size();
97. out.write(reinterpret\_cast<const char\*>(&timeSize), sizeof(timeSize));
98. out.write(time.data(), timeSize);
99. size\_t answersCount = answers.getSize();
100. out.write(reinterpret\_cast<const char\*>(&answersCount), sizeof(answersCount));
101. for (int i = 0; i < answersCount; ++i) {
102. size\_t answerSize = answers[i].size();
103. out.write(reinterpret\_cast<const char\*>(&answerSize), sizeof(answerSize));
104. out.write(answers[i].data(), answerSize);
105. }
106. out.write(reinterpret\_cast<const char\*>(&correctAnswersCount),
107. sizeof(correctAnswersCount));
108. out.close();
109. }
110. List<Statistic\*> Statistic::loadAllFromFile()
111. {
112. List<Statistic\*> statistics = {};
113. std::ifstream in(STAT\_FILE, std::ios::binary);
114. if (!in.is\_open()) return statistics;
115. while (in.peek() != EOF) {
116. try {
117. Statistic\* stat = Statistic::loadFromStream(in);
118. statistics.pushBack(stat);
119. }
120. catch (const std::exception&) {
121. throw FileError("Ошибка: Не удалось прочитать файл статистики");
122. }
123. }
124. in.close();
125. return statistics;
126. }
127. Statistic\* Statistic::loadFromStream(std::ifstream& in)
128. {
129. Test\* test = Test::loadFromStream(in);
130. Statistic\* stat = new Statistic("", test);
131. size\_t nameSize;
132. in.read(reinterpret\_cast<char\*>(&nameSize), sizeof(nameSize));
133. stat->name.resize(nameSize);
134. in.read(&stat->name[0], nameSize);
135. size\_t timeSize;
136. in.read(reinterpret\_cast<char\*>(&timeSize), sizeof(timeSize));
137. stat->time.resize(timeSize);
138. in.read(&stat->time[0], timeSize);
139. size\_t answersCount;
140. in.read(reinterpret\_cast<char\*>(&answersCount), sizeof(answersCount));
141. stat->answers.clear();
142. for (size\_t i = 0; i < answersCount; ++i) {
143. size\_t answerSize;
144. in.read(reinterpret\_cast<char\*>(&answerSize), sizeof(answerSize));
145. std::string answer(answerSize, '\0');
146. in.read(&answer[0], answerSize);
147. stat->answers.pushBack(answer);
148. }
149. in.read(reinterpret\_cast<char\*>(&stat->correctAnswersCount), sizeof(stat-

>correctAnswersCount));

1. return stat;
2. }

**StatisticMenu.h**

1. #pragma once
2. #include "Statistic.h"
3. #include "InputError.h"
4. #include <iostream>
5. #include <string>
6. class StatisticMenu {
7. public:
8. StatisticMenu();
9. ~StatisticMenu();
10. void start();
11. private:
12. void displayStatisticList();
13. void deleteStatistic();
14. int printStatisticList();
15. void saveStatistics();
16. void sortStatistics();
17. List<Statistic\*> statistics;
18. };

**StatisticMenu.cpp**

1. #include "StatisticMenu.h"
2. #include "Menu.h"
3. #include "FileError.h"
4. #include <fstream>
5. #include <iomanip>
6. #include <functional>
7. StatisticMenu::StatisticMenu() {
8. statistics = {};
9. statistics = Statistic::loadAllFromFile();
10. sortStatistics();
11. }
12. StatisticMenu::~StatisticMenu()
13. {
14. for (auto statistic : statistics) {
15. delete statistic;
16. }
17. statistics.clear();
18. }
19. void StatisticMenu::start()
20. {
21. char choice;
22. do {
23. std::cout << "Меню статистики:\n";
24. std::cout << "1. Просмотреть статистику\n";
25. std::cout << "2. Удалить статистику\n";
26. std::cout << "0. Вернуться в режим администратора\n";
27. try {
28. choice = Menu::getInput("012");
29. }
30. catch (const InputError& e) {
31. cout << e.what() << endl;
32. continue;
33. }
34. switch (choice) {
35. case '1':
36. displayStatisticList();
37. break;
38. case '2':
39. deleteStatistic();
40. saveStatistics();
41. break;
42. }
43. } while (choice != '0');
44. }
45. void StatisticMenu::displayStatisticList()
46. {
47. while (true) {
48. int maxIndex = printStatisticList();
49. if (maxIndex == 0) {
50. std::cout << "Список статистики пуст.\n\nВведите любой символ для

возврата...";

1. std::string input;
2. getline(std::cin, input);
3. system("cls");
4. return;
5. }
6. std::cout << "\nВведите номер статистики для просмотра, либо 0 для

возврата: ";

1. std::string input;
2. getline(std::cin, input);
3. system("cls");
4. if (input == "0") {
5. return;
6. }
7. try {
8. int selectedIndex = Menu::getValidatedNumber(input, 1, maxIndex);
9. statistics[selectedIndex - 1]->print();
10. }
11. catch (const InputError& e) {
12. std::cout << "Ошибка: " << e.what() << "\n";
13. }
14. }
15. }
16. int StatisticMenu::printStatisticList()
17. {
18. if (statistics.getSize() == 0) {
19. return 0;
20. }
21. const int colWidthIndex = 4;
22. const int colWidthTestTitle = MAX\_TEST\_TITLE\_LENGTH + 1;
23. const int colWidthUserName = MAX\_FULL\_NAME\_LENGTH + 1;
24. const int colWidthTime = 15;
25. std::cout << "Список статистики:\n";
26. std::cout << std::left << std::setw(colWidthIndex) << "№"
27. << std::left << std::setw(colWidthTestTitle) << "Название теста"
28. << std::left << std::setw(colWidthUserName) << "Имя тестируемого"
29. << std::left << std::setw(colWidthTime) << "Дата и время" << "\n";
30. std::cout << std::string(colWidthIndex + colWidthTestTitle + colWidthUserName

+ colWidthTime, '-') << "\n";

1. for (int i = 0; i < statistics.getSize(); ++i) {
2. const auto& stat = statistics[i];
3. std::cout << std::left << std::setw(colWidthIndex) << (i + 1)
4. << std::left << std::setw(colWidthTestTitle) << stat->getTestTitle()
5. << std::left << std::setw(colWidthUserName) << stat->getUserName()
6. << std::left << std::setw(colWidthTime) << stat->getTime() << "\n";
7. }
8. return statistics.getSize();
9. }
10. void StatisticMenu::deleteStatistic()
11. {
12. while (true) {
13. int maxIndex = printStatisticList();
14. if (maxIndex == 0) {
15. std::cout << "Список статистики пуст.\n\nВведите любой символ для

возврата...";

1. std::string input;
2. getline(std::cin, input);
3. system("cls");
4. return;
5. }
6. std::cout << "\nВведите номер статистики для удаления (или 0 для отмены):

";

1. std::string input;
2. getline(std::cin, input);
3. if (input == "0") {
4. system("cls");
5. return;
6. }
7. try {
8. int selectedIndex = Menu::getValidatedNumber(input, 1, maxIndex);
9. auto stat = statistics[selectedIndex - 1];
10. std::cout << "Вы действительно хотите удалить статистику для теста

\""

1. << stat->getTestTitle()
2. << "\" от пользователя \""
3. << stat->getUserName()
4. << "\"?\n(Введите Y для подтверждения или любой другой символ для
5. отмены): ";
6. getline(std::cin, input);
7. system("cls");
8. if (input != "Y" && input != "y") {
9. std::cout << "Удаление отменено.\n";
10. continue;
11. }
12. delete statistics[selectedIndex - 1];
13. statistics.removeAt(selectedIndex - 1);
14. std::cout << "Статистика успешно удалена!\n";
15. }
16. catch (const InputError& e) {
17. system("cls");
18. std::cout << "Ошибка: " << e.what() << "\n";
19. }
20. }
21. }
22. void StatisticMenu::saveStatistics()
23. {
24. do {
25. try {
26. std::ofstream out(STAT\_FILE, std::ios::binary);
27. if (!out.is\_open()) {
28. std::string errorTest = "Не удалось открыть файл для записи: ";
29. throw FileError(errorTest + STAT\_FILE);
30. }
31. out.close();
32. for (const auto& stat : statistics) {
33. stat->saveToFile();
34. }
35. break;
36. }
37. catch (const FileError& e) {
38. cout << "Ошибка: " << e.what() << endl;
39. cout << "Введите 0 для выхода без сохранения или любой другой символ

для повторной попытки: ";

1. string input;
2. getline(cin, input);
3. system("cls");
4. if (input == "0") return;
5. }
6. } while (true);
7. }
8. void StatisticMenu::sortStatistics() {
9. int n = statistics.getSize();
10. std::function<void(int, int)> heapify = [&](int n, int i) {
11. int largest = i;
12. int left = 2 \* i + 1;
13. int right = 2 \* i + 2;
14. if (left < n && statistics[left]->getTestTitle() > statistics[largest]-

>getTestTitle()) {

1. largest = left;
2. }
3. if (right < n && statistics[right]->getTestTitle() > statistics[largest]-

>getTestTitle()) {

1. largest = right;
2. }
3. if (largest != i) {
4. std::swap(statistics[i], statistics[largest]);
5. heapify(n, largest);
6. }
7. };
8. for (int i = n / 2 - 1; i >= 0; --i) {
9. heapify(n, i);
10. }
11. for (int i = n - 1; i > 0; --i) {
12. std::swap(statistics[0], statistics[i]);
13. heapify(i, 0);
14. }
15. }

**Test.h**

1. #pragma once
2. #include "Question.h"
3. #define PATH "tests/"
4. class Test
5. {
6. public:
7. Test(const Test& other);
8. Test(const std::string& title);
9. ~Test();
10. Test& operator=(const Test& other);
11. const std::string getTitle() const;
12. int getQuestionCount() const;
13. void setFileName(const std::string& fileName);
14. void setTitle(const std::string& newTitle);
15. void addQuestion(Question\* question);
16. void removeQuestion(int index);
17. Question\* getQuestion(int index);
18. void saveToFile();
19. void deleteTestFile();
20. static Test\* loadFromFile(const std::string& fileName);
21. static List<Test\*> loadAllTests();
22. private:
23. std::string sanitizeFileName(const std::string& input) const;
24. std::string generateFileName() const;
25. std::string title;
26. List<Question\*> questions;
27. std::string fileName = "";
28. };

**Test.cpp**

1. #include "Test.h"
2. #include "InputError.h"
3. #include "FileError.h"
4. #include <algorithm>
5. #include <string>
6. #include <cctype>
7. #include <fstream>
8. #include <sstream>
9. #include <iostream>
10. #include <unordered\_map>
11. #include "dirent.h"
12. #include <direct.h>
13. Test::Test(const Test& other)
14. {
15. title = other.title;
16. questions = other.questions;
17. fileName = other.fileName;
18. }
19. Test::Test(const std::string& title)
20. {
21. setTitle(title);
22. }
23. Test::~Test()
24. {
25. for (Question\* question : questions) {
26. delete question;
27. }
28. questions.clear();
29. }
30. Test& Test::operator=(const Test& other) {
31. if (this == &other) {
32. return \*this;
33. }
34. title = other.title;
35. questions = other.questions;
36. fileName = other.fileName;
37. return \*this;
38. }
39. const std::string Test::getTitle() const
40. {
41. return title;
42. }
43. int Test::getQuestionCount() const
44. {
45. return questions.getSize();
46. }
47. void Test::setTitle(const std::string& newTitle)
48. {
49. if (newTitle.size() > MAX\_TEST\_TITLE\_LENGTH) {
50. throw InputError("Название превышает максимально допустимую длину (" +
51. std::to\_string(MIN\_TITLE\_LENGTH) + " символов).");
52. }
53. if (newTitle.size() < MIN\_TITLE\_LENGTH) {
54. throw InputError("Название должно содерать минимум " +
55. std::to\_string(MIN\_TITLE\_LENGTH) + " символов.");
56. }
57. if (newTitle.empty()) {
58. throw InputError("Название не может быть пустым.");
59. }
60. if (newTitle.front() == ' ' || newTitle.back() == ' ') {
61. throw InputError("Название не может начинаться или заканчиваться

пробелом.");

1. }
2. if (!Question::isAlpha(newTitle.front())) {
3. throw InputError("Название должно начинаться с буквы");
4. }
5. for (size\_t i = 0; i < newTitle.size(); ++i) {
6. unsigned char ch = newTitle[i];
7. if (!Question::isAlpha(ch) && !std::isdigit(ch) && !(ch == ' ' || ch ==

'+' || ch == '-' || ch == '\_')) {

1. throw InputError("Название может содержать только русские/английские

буквы, пробелы и символы \"+-\_\"");

1. }
2. if (i > 0 && newTitle[i - 1] == ' ' && ch == ' ') {
3. throw InputError("Название не может содержать два и более пробела

подряд.");

1. }
2. }
3. this->title = newTitle;
4. }
5. // Методы для управления вопросами
6. void Test::addQuestion(Question\* question)
7. {
8. questions.pushBack(question);
9. }
10. void Test::removeQuestion(int index) {
11. auto question = getQuestion(index);
12. questions.removeAt(index);
13. delete question;
14. }
15. Question\* Test::getQuestion(int index) {
16. return questions[(unsigned int)index];
17. }
18. std::string Test::generateFileName() const {
19. std::string baseName = sanitizeFileName(title);
20. std::string extension = ".ktst";
21. std::string fileName = baseName + extension;
22. int counter = 1;
23. while (true) {
24. bool fileExists = false;
25. DIR\* dir = opendir(PATH);
26. if (!dir) {
27. throw FileError("Не удалось открыть текущий каталог для проверки.");
28. }
29. struct dirent\* entry;
30. while ((entry = readdir(dir)) != nullptr) {
31. if (fileName == entry->d\_name) {
32. fileExists = true;
33. break;
34. }
35. }
36. closedir(dir);
37. if (!fileExists) {
38. break;
39. }
40. std::ostringstream newFileName;
41. newFileName << baseName << counter << extension;
42. fileName = newFileName.str();
43. ++counter;
44. }
45. return fileName;
46. }
47. std::string Test::sanitizeFileName(const std::string& input) const
48. {
49. static const std::unordered\_map<unsigned char, std::string> transliteration =
50. {
51. {'А', "a"}, {'а', "a"}, {'Б', "b"}, {'б', "b"}, {'В', "v"}, {'в', "v"},
52. {'Г', "g"}, {'г', "g"}, {'Д', "d"}, {'д', "d"}, {'Е', "e"}, {'е', "e"},
53. {'Ё', "e"}, {'ё', "e"}, {'Ж', "zh"}, {'ж', "zh"}, {'З', "z"}, {'з', "z"},
54. {'И', "i"}, {'и', "i"}, {'Й', "y"}, {'й', "y"}, {'К', "k"}, {'к', "k"},
55. {'Л', "l"}, {'л', "l"}, {'М', "m"}, {'м', "m"}, {'Н', "n"}, {'н', "n"},
56. {'О', "o"}, {'о', "o"}, {'П', "p"}, {'п', "p"}, {'Р', "r"}, {'р', "r"},
57. {'С', "s"}, {'с', "s"}, {'Т', "t"}, {'т', "t"}, {'У', "u"}, {'у', "u"},
58. {'Ф', "f"}, {'ф', "f"}, {'Х', "kh"}, {'х', "kh"}, {'Ц', "ts"}, {'ц',

"ts"},

1. {'Ч', "ch"}, {'ч', "ch"}, {'Ш', "sh"}, {'ш', "sh"}, {'Щ', "shch"}, {'щ',

"shch"},

1. {'Ъ', ""}, {'ъ', ""}, {'Ы', "y"}, {'ы', "y"}, {'Ь', ""}, {'ь', ""},
2. {'Э', "e"}, {'э', "e"}, {'Ю', "yu"}, {'ю', "yu"}, {'Я', "ya"}, {'я', "ya"}
3. };
4. std::string fileName;
5. for (unsigned char ch : input)
6. {
7. auto it = transliteration.find(ch);
8. if (it != transliteration.end() || std::isalnum(ch) || ch == '-' || ch

== '\_')

1. {
2. if (it != transliteration.end())
3. {
4. fileName += it->second;
5. }
6. else if (std::isalpha(ch))
7. {
8. fileName += std::tolower(ch);
9. }
10. else
11. {
12. fileName += ch;
13. }
14. }
15. }
16. return fileName;
17. }
18. void Test::saveToFile()
19. {
20. std::string finalFileName = fileName.empty() ? generateFileName() : fileName;
21. std::ofstream outFile(PATH + finalFileName, std::ios::binary);
22. if (!outFile.is\_open())
23. {
24. throw FileError("Невозможно открыть файл для записи: " + finalFileName);
25. }
26. size\_t titleSize = title.size();
27. outFile.write(reinterpret\_cast<const char\*>(&titleSize), sizeof(titleSize));
28. outFile.write(title.data(), titleSize);
29. int questionCount = questions.getSize();
30. outFile.write(reinterpret\_cast<const char\*>(&questionCount),

sizeof(questionCount));

1. for (auto question : questions)
2. {
3. question->saveToStream(outFile);
4. }
6. outFile.close();
7. fileName = finalFileName;
8. }
9. void Test::deleteTestFile() {
10. if (fileName.empty())
11. {
12. throw FileError("Невозможно удалить файл, так как он ещё не создан");
13. }
14. if (std::remove((PATH + fileName).c\_str()) != 0) {
15. throw FileError("Не удалось удалить файл теста \"" + fileName + "\".");
16. }
17. fileName.clear();
18. }
19. Test\* Test::loadFromFile(const std::string& fileName)
20. {
21. std::ifstream inFile(PATH + fileName, std::ios::binary);
22. if (!inFile.is\_open())
23. {
24. throw FileError("Невозможно открыть файл для чтения: " + fileName);
25. }
26. size\_t titleSize;
27. inFile.read(reinterpret\_cast<char\*>(&titleSize), sizeof(titleSize));
28. std::string title(titleSize, '\0');
29. inFile.read(&title[0], titleSize);
30. Test\* test = new Test(title);
31. int questionCount;
32. inFile.read(reinterpret\_cast<char\*>(&questionCount), sizeof(questionCount));
33. for (int i = 0; i < questionCount; ++i)
34. {
35. Question\* question = Question::loadFromStream(inFile);
36. test->addQuestion(question);
37. }
38. inFile.close();
39. test->setFileName(fileName);
40. return test;
41. }
42. List<Test\*> Test::loadAllTests() {
43. List<Test\*> tests = {};
44. const std::string extension = ".ktst";
45. DIR\* dir;
46. struct dirent\* entry;
47. dir = opendir(PATH);
48. if (!dir) {
49. if (\_mkdir(PATH) != 0) {
50. throw FileError("Ошибка: Не удалось создать каталог для тестов");
51. }
52. return tests;
53. }
54. while ((entry = readdir(dir)) != nullptr) {
55. std::string fileName = entry->d\_name;
56. if (fileName.size() > extension.size() &&
57. fileName.substr(fileName.size() - extension.size()) == extension) {
58. try {
59. tests.pushBack(Test::loadFromFile(fileName));
60. } catch (std::exception&) {}
61. }
62. }
63. closedir(dir);
64. return tests;
65. }
66. void Test::setFileName(const std::string& fileName) {
67. this->fileName = fileName;
68. }

**Question.h**

1. #pragma once
2. #include "List.h"
3. #define MIN\_TITLE\_LENGTH 6
4. #define MAX\_TEST\_TITLE\_LENGTH 32
5. #define MAX\_QUESTION\_TITLE\_LENGTH 128
6. #define MAX\_QUESTION\_OPTION\_LENGTH 64
7. #define MAX\_QUESTION\_ANSWER\_LENGTH 20
8. #define MAX\_ANSWER\_LENGTH 32
9. enum class QuestionType
10. {
11. SingleChoice = 1,
12. MultipleChoice = 2,
13. OpenAnswer = 3
14. };
15. class Question
16. {
17. public:
18. Question(const Question& other);
19. Question(const std::string& title);
20. Question() {}
21. Question& operator=(const Question& other);
22. virtual ~Question() = default;
23. virtual QuestionType getType() const = 0;
24. virtual void printQuestion() = 0;
25. virtual bool checkAnswer(const std::string answer) const = 0;
26. virtual void saveToStream(std::ofstream& out) = 0;
27. std::string getTitle() const;
28. void setTitle(const std::string& title);
29. static Question\* loadFromStream(std::ifstream& in);
30. static bool isAlpha(unsigned char ch);
31. static bool isSymbol(unsigned char ch);
32. protected:
33. std::string title = "";
34. };

**Question.cpp**

1. #include "Question.h"
2. #include "SingleChoiceQuestion.h"
3. #include "MultipleChoiceQuestion.h"
4. #include "OpenAnswerQuestion.h"
5. #include "InputError.h"
6. #include "FileError.h"
7. #include <string>
8. #include <memory>
9. #include <fstream>
10. Question::Question(const Question& other) { setTitle(other.title); }
11. Question::Question(const std::string& title) { setTitle(title); }
12. Question& Question::operator=(const Question& other)
13. {
14. if (this == &other) {
15. return \*this;
16. }
17. setTitle(other.title);
18. return \*this;
19. }
20. std::string Question::getTitle() const
21. {
22. return title;
23. }
24. bool Question::isAlpha(unsigned char ch)
25. {
26. return std::isalpha(ch) ||
27. (ch >= 0xC0 && ch <= 0xFF) ||
28. (ch == 0xB8 || ch == 0xA8);
29. }
30. bool Question::isSymbol(unsigned char ch) {
31. static const std::string validSymbols = "-+=?!;%№ $()\*&|#@:/.,><~\_[]{}^'\"\\";
32. return validSymbols.find(ch) != std::string::npos;
33. }
34. void Question::setTitle(const std::string& title)
35. {
36. if (title.size() > MAX\_QUESTION\_TITLE\_LENGTH) {
37. throw InputError("Заголовок превышает максимально допустимую длину (" +
38. std::to\_string(MAX\_QUESTION\_TITLE\_LENGTH) + " символов).");
39. }
40. if (title.size() < MIN\_TITLE\_LENGTH) {
41. throw InputError("Заголовок должен содерать минимум " +
42. std::to\_string(MIN\_TITLE\_LENGTH) + " символов.");
43. }
44. if (title.empty()) {
45. throw InputError("Заголовок не может быть пустым.");
46. }
47. if (title.front() == ' ' || title.back() == ' ') {
48. throw InputError("Заголовок может начинаться или заканчиваться

пробелом.");

1. }
2. if (!Question::isAlpha(title.front())) {
3. throw InputError("Заголовок должен начинаться с буквы");
4. }
5. for (size\_t i = 0; i < title.size(); ++i) {
6. unsigned char ch = title[i];
7. if (!Question::isAlpha(ch) && !std::isdigit(ch)

&& !Question::isSymbol(ch)) {

1. throw InputError("Заголовок может содержать только русские/английские

буквы, пробелы и спец. символы.");

1. }
2. if (i > 0 && title[i - 1] == ' ' && ch == ' ') {
3. throw InputError("Заголовок не может содержать два и более пробела

подряд.");

1. }
2. }
3. this->title = title;
4. }
5. Question\* Question::loadFromStream(std::ifstream& in)
6. {
7. int typeInt;
8. in.read(reinterpret\_cast<char\*>(&typeInt), sizeof(typeInt));
9. QuestionType type = static\_cast<QuestionType>(typeInt);
10. switch (type)
11. {
12. case QuestionType::SingleChoice:
13. return new SingleChoiceQuestion(SingleChoiceQuestion::loadFromStream(in));
14. case QuestionType::MultipleChoice:
15. return new

MultipleChoiceQuestion(MultipleChoiceQuestion::loadFromStream(in));

1. case QuestionType::OpenAnswer:
2. return new OpenAnswerQuestion(OpenAnswerQuestion::loadFromStream(in));
3. default:
4. throw FileError("Неизвестный тип вопроса.");
5. }
6. }

**SingleChoiceQuestion.h**

1. #pragma once
2. #include "ChoiceQuestion.h"
3. class SingleChoiceQuestion : public ChoiceQuestion
4. {
5. public:
6. SingleChoiceQuestion(const SingleChoiceQuestion& other);
7. SingleChoiceQuestion(const std::string& title, const List<std::string>&

options, char correctIndex);

1. SingleChoiceQuestion() {}
2. SingleChoiceQuestion& operator=(const SingleChoiceQuestion& other);
3. QuestionType getType() const override;
4. bool checkAnswer(const std::string answer) const override;
5. void saveToStream(std::ofstream& out) override;
6. void printQuestion() override;
7. void setCorrectIndex(unsigned char correctIndex);
8. static SingleChoiceQuestion loadFromStream(std::ifstream& in);
9. private:
10. char correctIndex = 'A';
11. };

**SingleChoiceQuestion.cpp**

1. #include "SingleChoiceQuestion.h"
2. #include "InputError.h"
3. #include <iostream>
4. #include <string>
5. #include <cctype>
6. #include <fstream>
7. SingleChoiceQuestion::SingleChoiceQuestion(
8. const SingleChoiceQuestion& other) : ChoiceQuestion(other)
9. {
10. setCorrectIndex(other.correctIndex);
11. }
12. SingleChoiceQuestion::SingleChoiceQuestion(
13. const std::string& title,
14. const List<std::string>& options,
15. char correctIndex) : ChoiceQuestion(title, options)
16. {
17. setCorrectIndex(correctIndex);
18. }
19. SingleChoiceQuestion& SingleChoiceQuestion::operator=(const SingleChoiceQuestion&

other) {

1. if (this == &other) {
2. return \*this;
3. }
4. ChoiceQuestion::operator=(other);
5. setCorrectIndex(other.correctIndex);
6. return \*this;
7. }
8. QuestionType SingleChoiceQuestion::getType() const
9. {
10. return QuestionType::SingleChoice;
11. }
12. void SingleChoiceQuestion::printQuestion()
13. {
14. std::cout << title << "\n";
15. std::cout << "Варианты ответа:\n";
16. for (int i = 0; i < options.getSize(); ++i) {
17. std::cout << static\_cast<char>('A' + i) << ") " << options[i] << "\n";
18. }
19. }
20. bool SingleChoiceQuestion::checkAnswer(const std::string answer) const
21. {
22. if (answer.size() != 1)
23. {
24. throw InputError("Введите один символ.");
25. }
26. char userAnswer = std::toupper(answer[0]);
27. if (!std::isalpha(userAnswer))
28. {
29. throw InputError("Ответом может быть только буква латинского алфавита (A-

Z).");

1. }
2. if (userAnswer < 'A' || userAnswer >= 'A' + options.getSize())
3. {
4. throw InputError("Ответ выходит за пределы допустимых вариантов (A-" +
5. std::string(1, 'A' + options.getSize() - 1) + ").");
6. }
7. return userAnswer == correctIndex;
8. }
9. void SingleChoiceQuestion::setCorrectIndex(unsigned char correctIndex)
10. {
11. correctIndex = std::toupper(correctIndex);
12. if (!std::isalpha(correctIndex))
13. {
14. throw InputError("Правильный ответ должен быть буквой латинского

алфавита.");

1. }
2. if (correctIndex < 'A' || correctIndex >= 'A' + options.getSize())
3. {
4. throw InputError("Правильный ответ выходит за пределы допустимых

вариантов (A-" +

1. std::string(1, 'A' + options.getSize() - 1) + ").");
2. }
3. this->correctIndex = correctIndex;
4. }
5. void SingleChoiceQuestion::saveToStream(std::ofstream& out)
6. {
7. int type = static\_cast<int>(getType());
8. out.write(reinterpret\_cast<const char\*>(&type), sizeof(type));
9. size\_t titleSize = title.size();
10. out.write(reinterpret\_cast<const char\*>(&titleSize), sizeof(titleSize));
11. out.write(title.data(), titleSize);
12. size\_t optionsSize = options.getSize();
13. out.write(reinterpret\_cast<const char\*>(&optionsSize), sizeof(optionsSize));
14. for (const auto& option : options)
15. {
16. size\_t optSize = option.size();
17. out.write(reinterpret\_cast<const char\*>(&optSize), sizeof(optSize));
18. out.write(option.data(), optSize);
19. }
20. out.write(reinterpret\_cast<const char\*>(&correctIndex), sizeof(correctIndex));
21. }
22. SingleChoiceQuestion SingleChoiceQuestion::loadFromStream(std::ifstream& in)
23. {
24. size\_t titleSize;
25. in.read(reinterpret\_cast<char\*>(&titleSize), sizeof(titleSize));
26. std::string title(titleSize, '\0');
27. in.read(&title[0], titleSize);
28. size\_t optionsSize;
29. in.read(reinterpret\_cast<char\*>(&optionsSize), sizeof(optionsSize));
30. List<std::string> options;
31. for (size\_t i = 0; i < optionsSize; ++i)
32. {
33. size\_t optSize;
34. in.read(reinterpret\_cast<char\*>(&optSize), sizeof(optSize));
35. std::string option(optSize, '\0');
36. in.read(&option[0], optSize);
37. options.pushBack(option);
38. }
39. char correctIndex;
40. in.read(reinterpret\_cast<char\*>(&correctIndex), sizeof(correctIndex));
41. return SingleChoiceQuestion(title, options, correctIndex);
42. }

**MultipleChoiceQuestion.h**

1. #pragma once
2. #include "ChoiceQuestion.h"
3. class MultipleChoiceQuestion : public ChoiceQuestion
4. {
5. public:
6. MultipleChoiceQuestion(const MultipleChoiceQuestion& other);
7. MultipleChoiceQuestion(const std::string& title, const List<std::string>&
8. options, const List<unsigned char>& correctIndices);
9. MultipleChoiceQuestion() {}
10. MultipleChoiceQuestion& operator=(const MultipleChoiceQuestion& other);
11. QuestionType getType() const override;
12. bool checkAnswer(const std::string answer) const override;
13. void printQuestion() override;
14. void saveToStream(std::ofstream& out) override;
15. void setCorrectIndices(const List<unsigned char>& options);
16. static MultipleChoiceQuestion loadFromStream(std::ifstream& in);
17. private:
18. List<unsigned char> correctIndices;
19. };

**MultipleChoiceQuestion.cpp**

1. #include "MultipleChoiceQuestion.h"
2. #include "InputError.h"
3. #include <iostream>
4. #include <sstream>
5. #include <fstream>
6. MultipleChoiceQuestion::MultipleChoiceQuestion(
7. const MultipleChoiceQuestion& other) : ChoiceQuestion(other)
8. {
9. setCorrectIndices(other.correctIndices);
10. }
11. MultipleChoiceQuestion::MultipleChoiceQuestion(
12. const std::string& title,
13. const List<std::string>& options,
14. const List<unsigned char>& correctIndices) : ChoiceQuestion(title, options)
15. {
16. setCorrectIndices(correctIndices);
17. }
18. MultipleChoiceQuestion& MultipleChoiceQuestion::operator=(const

MultipleChoiceQuestion& other)

1. {
2. if (this == &other) {
3. return \*this;
4. }
5. ChoiceQuestion::operator=(other);
6. setCorrectIndices(other.correctIndices);
7. return \*this;
8. }
9. QuestionType MultipleChoiceQuestion::getType() const
10. {
11. return QuestionType::MultipleChoice;
12. }
13. void MultipleChoiceQuestion::printQuestion()
14. {
15. std::cout << title << "\n";
16. std::cout << "Варианты ответа:\n";
17. for (int i = 0; i < options.getSize(); ++i) {
18. std::cout << static\_cast<char>('A' + i) << "] " << options[i] << "\n";
19. }
20. }
21. bool MultipleChoiceQuestion::checkAnswer(const std::string answer) const
22. {
23. List<char> userIndices;
24. std::istringstream iss(answer);
25. char index;
26. while (iss >> index)
27. {
28. if (!std::isalpha(index))
29. {
30. throw InputError("Ответами могут быть только буквы латинского

алфавита (A-Z).");

1. }
2. char userChar = std::toupper(index);
3. if (userChar < 'A' || userChar >= 'A' + options.getSize())
4. {
5. throw InputError("Правильный ответ \"" + std::string(1, userChar) +
6. "\" выходит за пределы допустимых вариантов (A-" +
7. std::string(1, 'A' + options.getSize() - 1) + ").");
8. }
9. bool inserted = false;
10. for (int i = 0; i < userIndices.getSize(); ++i) {
11. if (userChar == userIndices[i]) {
12. throw InputError("Правильный ответ \"" + std::string(1, userChar)

+ "\" указан более одного раза.");

1. }
2. if (userIndices[i] > userChar) {
3. userIndices.insert(userChar, i);
4. inserted = true;
5. break;
6. }
7. }
8. if (!inserted) {
9. userIndices.pushBack(userChar);
10. }
11. }
12. if (userIndices.getSize() != correctIndices.getSize()) return false;
13. for (auto it = correctIndices.begin(); it != correctIndices.end(); ++it)
14. {
15. if (!userIndices.contains(\*it)) return false;
16. }
17. return true;
18. }
19. void MultipleChoiceQuestion::setCorrectIndices(const List<unsigned char>&

correctIndices)

1. {
2. if (correctIndices.isEmpty())
3. {
4. throw InputError("Необходимо указать хотя бы один правильный ответ.");
5. }
6. List<unsigned char> indices = {};
7. for (unsigned char index : correctIndices)
8. {
9. unsigned char upperIndex = std::toupper(index);
10. if (!std::isalpha(upperIndex))
11. {
12. throw InputError("Каждый правильный ответ должен быть буквой

латинского алфавита.");

1. }
2. if (upperIndex < 'A' || upperIndex >= 'A' + options.getSize())
3. {
4. throw InputError("Правильный ответ \"" + std::string(1, upperIndex) +
5. "\" выходит за пределы допустимых вариантов (A-" +
6. std::string(1, 'A' + options.getSize() - 1) + ").");
7. }
8. bool inserted = false;
9. for (int i = 0; i < indices.getSize(); ++i) {
10. if (upperIndex == indices[i]) {
11. throw InputError("Правильный ответ \"" + std::string(1,

upperIndex) +

1. "\" указан более одного раза.");
2. }
3. if (indices[i] > upperIndex) {
4. indices.insert(upperIndex, i);
5. inserted = true;
6. break;
7. }
8. }
9. if (!inserted) {
10. indices.pushBack(upperIndex);
11. }
12. }
13. this->correctIndices = indices;
14. }
15. void MultipleChoiceQuestion::saveToStream(std::ofstream& out)
16. {
17. int type = static\_cast<int>(getType());
18. out.write(reinterpret\_cast<const char\*>(&type), sizeof(type));
19. size\_t titleSize = title.size();
20. out.write(reinterpret\_cast<const char\*>(&titleSize), sizeof(titleSize));
21. out.write(title.data(), titleSize);
22. size\_t optionsSize = options.getSize();
23. out.write(reinterpret\_cast<const char\*>(&optionsSize), sizeof(optionsSize));
24. for (const auto& option : options)
25. {
26. size\_t optSize = option.size();
27. out.write(reinterpret\_cast<const char\*>(&optSize), sizeof(optSize));
28. out.write(option.data(), optSize);
29. }
30. size\_t correctSize = correctIndices.getSize();
31. out.write(reinterpret\_cast<const char\*>(&correctSize), sizeof(correctSize));
32. for (const char& index : correctIndices)
33. {
34. out.write(reinterpret\_cast<const char\*>(&index), sizeof(index));
35. }
36. }
37. MultipleChoiceQuestion MultipleChoiceQuestion::loadFromStream(std::ifstream& in)
38. {
39. size\_t titleSize;
40. in.read(reinterpret\_cast<char\*>(&titleSize), sizeof(titleSize));
41. std::string title(titleSize, '\0');
42. in.read(&title[0], titleSize);
43. size\_t optionsSize;
44. in.read(reinterpret\_cast<char\*>(&optionsSize), sizeof(optionsSize));
45. List<std::string> options;
46. for (size\_t i = 0; i < optionsSize; ++i)
47. {
48. size\_t optSize;
49. in.read(reinterpret\_cast<char\*>(&optSize), sizeof(optSize));
50. std::string option(optSize, '\0');
51. in.read(&option[0], optSize);
52. options.pushBack(option);
53. }
54. size\_t correctSize;
55. in.read(reinterpret\_cast<char\*>(&correctSize), sizeof(correctSize));
56. List<unsigned char> correctIndices;
57. for (size\_t i = 0; i < correctSize; ++i)
58. {
59. char index;
60. in.read(reinterpret\_cast<char\*>(&index), sizeof(index));
61. correctIndices.pushBack(index);
62. }
63. return MultipleChoiceQuestion(title, options, correctIndices);
64. }

**OpenAnswerQuestion.h**

1. #pragma once
2. #include "Question.h"
3. #define ALPHA\_OFFSET 32
4. class OpenAnswerQuestion : public Question
5. {
6. public:
7. OpenAnswerQuestion(const OpenAnswerQuestion& other);
8. OpenAnswerQuestion(const std::string& title, const std::string&

correctAnswer);

1. OpenAnswerQuestion() {}
2. OpenAnswerQuestion& operator=(const OpenAnswerQuestion& other);
3. QuestionType getType() const override;
4. void printQuestion() override;
5. bool checkAnswer(const std::string answer) const override;
6. void setAnswer(const std::string answer);
7. void saveToStream(std::ofstream& out) override;
8. static OpenAnswerQuestion loadFromStream(std::ifstream& in);
9. private:
10. unsigned char toLower(unsigned char) const;
11. int levenshteinDistance(const std::string& str1, const std::string& str2)
12. const;
13. std::string correctAnswer;
14. };

**OpenAnswerQuestion.cpp**

1. #include "OpenAnswerQuestion.h"
2. #include "InputError.h"
3. #include <iostream>
4. #include <cctype>
5. #include <string>
6. #include <locale>
7. #include <fstream>
8. OpenAnswerQuestion::OpenAnswerQuestion(const OpenAnswerQuestion& other) :
9. Question(other)
10. {
11. setAnswer(other.correctAnswer);
12. }
13. OpenAnswerQuestion::OpenAnswerQuestion(
14. const std::string& title,
15. const std::string& correctAnswer) : Question(title)
16. {
17. setAnswer(correctAnswer);
18. }
19. OpenAnswerQuestion& OpenAnswerQuestion::operator=(const OpenAnswerQuestion& other)
20. {
21. if (this == &other) {
22. return \*this;
23. }
24. Question::operator=(other);
25. setAnswer(other.correctAnswer);
26. return \*this;
27. }
28. QuestionType OpenAnswerQuestion::getType() const
29. {
30. return QuestionType::OpenAnswer;
31. }
32. bool OpenAnswerQuestion::checkAnswer(const std::string answer) const
33. {
34. if (answer.empty())
35. {
36. throw InputError("Ответ не может быть пустым.");
37. }
38. if (answer.size() > MAX\_ANSWER\_LENGTH)
39. {
40. throw InputError("Ответ превышает максимально допустимую длину (" +
41. std::to\_string(MAX\_ANSWER\_LENGTH) + " символов).");
42. }
43. for (char ch : answer)
44. {
45. if (!Question::isAlpha(ch) && !std::isdigit(ch)) {
46. throw InputError("Ответ может содержать только русские/английские

буквы и цифры");

1. }
2. }
3. std::string lowerAnswer;
4. for (unsigned char ch : answer) {
5. lowerAnswer += toLower(ch);
6. }
7. int distance = levenshteinDistance(lowerAnswer, this->correctAnswer);
8. int threshold = static\_cast<int>(std::round(correctAnswer.size() \* 0.2));
9. return distance <= threshold;
10. }
11. int OpenAnswerQuestion::levenshteinDistance(const std::string& str1, const

std::string& str2) const

1. {
2. size\_t len1 = str1.size();
3. size\_t len2 = str2.size();
4. List<int> prev;
5. List<int> curr;
6. for (unsigned int j = 0; j <= len2; ++j)
7. {
8. prev.pushBack(j);
9. }
10. for (unsigned int i = 1; i <= len1; ++i)
11. {
12. curr.clear();
13. curr.pushBack(i);
14. auto str2It = str2.begin();
15. for (unsigned int j = 1; j <= len2; ++j, ++str2It)
16. {
17. int cost = (str1[i - 1] == str2[j - 1]) ? 0 : 1;
18. curr.pushBack(std::min({prev[j] + 1,
19. curr[j - 1] + 1,
20. prev[j - 1] + cost }));
21. }
22. prev = curr;
23. }
24. return prev.back();
25. }
26. void OpenAnswerQuestion::printQuestion()
27. {
28. std::cout << title << "\n";
29. }
30. void OpenAnswerQuestion::setAnswer(const std::string answer)
31. {
32. if (answer.empty())
33. {
34. throw InputError("Ответ не может быть пустым.");
35. }
36. if (answer.size() > MAX\_ANSWER\_LENGTH)
37. {
38. throw InputError("Ответ превышает максимально допустимую длину (" +
39. std::to\_string(MAX\_ANSWER\_LENGTH) + " символов).");
40. }
41. for (char ch : answer)
42. {
43. if (!Question::isAlpha(ch) && !std::isdigit(ch)) {
44. throw InputError("Ответ может содержать только русские/английские

буквы и цифры");

1. }
2. }
3. this->correctAnswer = answer;
4. }
5. void OpenAnswerQuestion::saveToStream(std::ofstream& out)
6. {
7. int type = static\_cast<int>(getType());
8. out.write(reinterpret\_cast<const char\*>(&type), sizeof(type));
9. size\_t titleSize = title.size();
10. out.write(reinterpret\_cast<const char\*>(&titleSize), sizeof(titleSize));
11. out.write(title.data(), titleSize);
12. size\_t answerSize = correctAnswer.size();
13. out.write(reinterpret\_cast<const char\*>(&answerSize), sizeof(answerSize));
14. out.write(correctAnswer.data(), answerSize);
15. }
16. OpenAnswerQuestion OpenAnswerQuestion::loadFromStream(std::ifstream& in)
17. {
18. size\_t titleSize;
19. in.read(reinterpret\_cast<char\*>(&titleSize), sizeof(titleSize));
20. std::string title(titleSize, '\0');
21. in.read(&title[0], titleSize);
22. size\_t answerSize;
23. in.read(reinterpret\_cast<char\*>(&answerSize), sizeof(answerSize));
24. std::string correctAnswer(answerSize, '\0');
25. in.read(&correctAnswer[0], answerSize);
26. return OpenAnswerQuestion(title, correctAnswer);
27. }
28. unsigned char OpenAnswerQuestion::toLower(unsigned char ch) const{
29. if (ch >= 'А' && ch <= 'Я') {
30. return ch + ALPHA\_OFFSET;
31. }
32. else if (ch == 'Ё') {
33. return 'ё';
34. }
35. return static\_cast<char>(std::tolower(static\_cast<unsigned char>(ch)));
36. }

**ChoiceQuestion.h**

1. #pragma once
2. #include "Question.h"
3. class ChoiceQuestion : public Question
4. {
5. public:
6. ChoiceQuestion(const ChoiceQuestion& other);
7. ChoiceQuestion(const std::string& title, const List<std::string>& options);
8. ChoiceQuestion(): Question() {}
9. ChoiceQuestion& operator=(const ChoiceQuestion& other);
10. virtual ~ChoiceQuestion() = default;
11. List<std::string> getOptions() const;
12. std::string getOption(unsigned int index);
13. int getOptionCount() const;
14. void addOption(const std::string& option, int index = -1);
15. void removeOption(unsigned int index);
16. protected:
17. List<std::string> options = {};
18. };

**ChoiceQuestion.cpp**

1. #include "ChoiceQuestion.h"
2. #include "InputError.h"
3. #include <string>
4. #include <cctype>
5. ChoiceQuestion::ChoiceQuestion(const ChoiceQuestion& other) : Question(other)
6. {
7. for (auto option : other.options) {
8. addOption(option);
9. }
10. }
11. ChoiceQuestion::ChoiceQuestion(const std::string& title, const List<std::string>&

options) : Question(title)

1. {
2. for (auto option : options) {
3. addOption(option);
4. }
5. }
6. ChoiceQuestion& ChoiceQuestion::operator=(const ChoiceQuestion& other)
7. {
8. if (this == &other) {
9. return \*this;
10. }
11. Question::operator=(other);
12. options.clear();
13. for (auto option : other.options) {
14. addOption(option);
15. }
16. return \*this;
17. }
18. void ChoiceQuestion::addOption(const std::string& option, int index)
19. {
20. if (option.empty())
21. {
22. throw InputError("Вариант ответа не может быть пустым.");
23. }
24. if (option.size() > MAX\_QUESTION\_OPTION\_LENGTH)
25. {
26. throw InputError("Вариант ответа превышает максимально допустимую длину

(" + std::to\_string(MAX\_QUESTION\_OPTION\_LENGTH) + " символов).");

1. }
2. if (option.empty()) {
3. throw InputError("Вариант ответа не может быть пустым.");
4. }
5. if (option.front() == ' ' || option.back() == ' ') {
6. throw InputError("Вариант ответа не может начинаться или заканчиваться

пробелом.");

1. }
2. for (size\_t i = 0; i < option.size(); ++i) {
3. unsigned char ch = option[i];
4. if (!Question::isAlpha(ch) && !std::isdigit(ch)
5. && !Question::isSymbol(ch)) { throw InputError("Вариант ответа может содержать только русские/английские буквы, пробелы и спец. символы.");
6. }
7. if (i > 0 && option[i - 1] == ' ' && ch == ' ') {
8. throw InputError("Вариант ответа не может содержать два и более

пробела подряд.");

1. }
2. }
3. if (index == -1) {
4. options.pushBack(option);
5. }
6. else {
7. if (index > options.getSize()) {
8. throw InputError("Индекс выходит за пределы доступных вариантов.");
9. }
10. options.insert(option, index);
11. }
13. }
14. void ChoiceQuestion::removeOption(unsigned int index)
15. {
16. if ((int)index >= options.getSize())
17. {
18. throw InputError("Индекс выходит за пределы диапазона.");
19. }
20. options.removeAt(index);
21. }
22. int ChoiceQuestion::getOptionCount() const {
23. return options.getSize();
24. }
25. List<std::string> ChoiceQuestion::getOptions() const
26. {
27. auto newOptions = options;
28. return newOptions;
29. }
30. std::string ChoiceQuestion::getOption(unsigned int index) {
31. return options[index];
32. }

**List.h**

1. #pragma once
2. #include <iostream>
3. enum RemoveMode
4. {
5. All,
6. First
7. };
8. template<typename T>
9. class List
10. {
11. private:
12. template<typename T>
13. struct Node
14. {
15. Node(T data = T(), Node<T> \*ptrPrevious = nullptr, Node<T> \*ptrNext =
16. nullptr)
17. {
18. this->data = data;
19. this->ptrPrevious = ptrPrevious;
20. this->ptrNext = ptrNext;
21. }
22. T data;
23. Node<T> \*ptrPrevious;
24. Node<T> \*ptrNext;
25. };
26. Node<T> \*findElement(const unsigned int index)
27. {
28. if (size <= (int)index)
29. return nullptr;
30. Node<T>\* findedNode;
31. if (size / 2 >= (int)index)
32. {
33. findedNode = head;
34. for (unsigned int i = 0; i < index; ++i)
35. findedNode = findedNode->ptrNext;
36. }
37. else
38. {
39. findedNode = tail;
40. for (unsigned int i = size - 1; i > index; --i)
41. findedNode = findedNode->ptrPrevious;
42. }
43. return findedNode;
44. }
45. void setDefaultValues();
46. int size;
47. Node<T> \*head;
48. Node<T> \*tail;
49. public:
50. List(const T\*, const int);
51. List();
52. List(std::initializer\_list<T>);
53. ~List();
54. T &operator[](const unsigned int);
55. List<T> &operator=(const List<T>& other);
56. T &front() const;
57. T &back() const;
58. void pushFront(const T);
59. void pushBack(const T);
60. void popFront();
61. void popBack();
62. void insert(const T &, const unsigned int);
63. void removeAt(const unsigned int);
64. void remove(const T &, RemoveMode);
65. void clear();
66. bool isEmpty() const;
67. int getSize() const;
68. int firstIndexOf(const T &) const;
69. bool contains(const T &) const;
70. template<typename T>
71. class Iterator {
72. public:
73. Iterator() : node(nullptr) {}
74. Iterator(Node<T>\* node) : node(node) {}
75. Iterator(const Iterator &other) : node(other.node) {}
76. Iterator& operator=(const Iterator &other) {
77. node = other.node;
78. return \*this;
79. }
80. T &operator\*() const { return node->data; }
81. T\* operator->() { return &(node->data); }
82. bool operator==(const Iterator &other) const { return node ==

other.node; }

1. bool operator!=(const Iterator &other) const { return node !=

other.node; }

1. Iterator &operator++()
2. {
3. node = node->ptrNext;
4. return \*this;
5. }
6. Iterator operator++(int)
7. {
8. Iterator temp(\*this);
9. node = node->ptrNext;
10. return temp;
11. }
12. Iterator &operator--()
13. {
14. node = node->ptrPrevious;
15. return \*this;
16. }
17. Iterator operator--(int)
18. {
19. Iterator temp(\*this);
20. node = node->ptrPrevious;
21. return temp;
22. }
23. Iterator operator+(int n) const {
24. Node<T>\* newNode = node;
25. for (int i = 0; i < n; ++i)
26. {
27. if (newNode != nullptr)
28. newNode = newNode->ptrNext;
29. }
30. return Iterator(newNode);
31. }
33. Iterator operator-(int n) const {
34. Node<T>\* newNode = node;
35. for (int i = 0; i < n; ++i)
36. {
37. if (newNode != nullptr)
38. newNode = newNode->ptrPrevious;
39. }
40. return Iterator(newNode);
41. }
42. private:
43. Node<T>\* node;
44. };
45. Iterator<T> begin() const { return Iterator<T>(head); }
46. Iterator<T> end() const { return Iterator<T>(nullptr); }
47. Iterator<T> rbegin() const { return Iterator<T>(tail); }
48. Iterator<T> rend() const { return Iterator<T>(nullptr); }
49. Iterator<T> find(const T& data) const {
50. for (Iterator<T> it = begin(); it != end(); ++it) {
51. if (\*it == data)
52. return it;
53. }
54. return end();
55. }
56. Iterator<T> findAt(const unsigned int index) const {
57. if ((unsigned)size <= index)
58. throw std::out\_of\_range("Ошибка(Iterator<T> findAt(const unsigned

int): Выход за границы списка");

1. Node<T>\* findedNode;
2. if ((unsigned)size / 2 >= index)
3. {
4. findedNode = head;
5. for (unsigned int i = 0; i < index; ++i)
6. findedNode = findedNode->ptrNext;
7. }
8. else
9. {
10. findedNode = tail;
11. for (unsigned int i = size - 1; i > index; --i)
12. findedNode = findedNode->ptrPrevious;
13. }
14. return Iterator<T>(findedNode);
15. }
16. };
17. template<typename T>
18. List<T>::List(const T\* data, const int size)
19. {
20. for (int i = 0; i < size; ++i)
21. {
22. pushBack(\*(data + i));
23. }
24. }
25. template<typename T>
26. List<T>::List()
27. {
28. setDefaultValues();
29. }
30. template<typename T>
31. List<T>& List<T>::operator=(const List<T>& other) {
32. if (this == &other)
33. return \*this;
34. clear();
35. Node<T>\* currentNode = other.head;
36. while (currentNode != nullptr) {
37. pushBack(currentNode->data);
38. currentNode = currentNode->ptrNext;
39. }
40. return \*this;
41. }
42. template<typename T>
43. List<T>::List(std::initializer\_list<T> data)
44. {
45. setDefaultValues();
46. for (auto item : data)
47. pushBack(item);
48. }
49. template<typename T>
50. List<T>::~List()
51. {
52. clear();
53. }
54. template<typename T>
55. T& List<T>::front() const
56. {
57. if (isEmpty())
58. throw std::exception("Ошибка(T & List<T>::front()): Выход за границы

списка");

1. return head->data;
2. }
3. template<typename T>
4. T& List<T>::back() const
5. {
6. if (isEmpty())
7. throw std::out\_of\_range("Ошибка(T & List<T>::back()): Выход за границы
8. списка");
9. return tail->data;
10. }
11. template<typename T>
12. T& List<T>::operator[](const unsigned int index)
13. {
14. if (size <= (int)index)
15. throw std::out\_of\_range("Ошибка(T & List<T>::operator[](const int

index)): Выход за границы списка");

1. Node<T>\* currentNode = findElement(index);
2. return currentNode->data;
3. }
4. template<typename T>
5. void List<T>::setDefaultValues()
6. {
7. size = 0;
8. head = nullptr;
9. tail = nullptr;
10. }
11. template<typename T>
12. void List<T>::pushFront(const T element)
13. {
14. if (size == 0)
15. {
16. head = new Node<T>(element, nullptr, nullptr);
17. tail = head;
18. }
19. else
20. {
21. Node<T>\* newNode = new Node<T>(element, nullptr, head);
22. head->ptrPrevious = newNode;
23. head = newNode;
24. }
25. ++size;
26. }
27. template<typename T>
28. void List<T>::pushBack(const T element)
29. {
30. if (size <= 1)
31. {
32. if (size == 0)
33. {
34. head = new Node<T>(element, nullptr, nullptr);
35. tail = head;
36. }
37. else
38. {
39. tail = new Node<T>(element, head, nullptr);
40. head->ptrNext = tail;
41. }
42. }
43. else
44. {
45. Node<T>\* newNode = new Node<T>(element, tail, nullptr);
46. tail->ptrNext = newNode;
47. tail = newNode;
48. }
49. ++size;
50. }
51. template<typename T>
52. void List<T>::popFront()
53. {
54. if (size <= 1)
55. {
56. if (size == 0)
57. throw std::exception("Ошибка: Cписок уже пуст!");
58. if (size == 1)
59. {
60. delete head;
61. setDefaultValues();
62. }
63. return;
64. }
65. Node<T>\* nextNode = head->ptrNext;
66. delete head;
67. nextNode->ptrPrevious = nullptr;
68. head = nextNode;
69. --size;
70. }
71. template<typename T>
72. void List<T>::popBack()
73. {
74. if (size <= 1)
75. {
76. if (size == 0)
77. throw std::exception("Ошибка: Cписок уже пуст!");
78. if (size == 1)
79. {
80. delete tail;
81. setDefaultValues();
82. }
83. return;
84. }
85. Node<T>\* previousNode = tail->ptrPrevious;
86. delete tail;
87. previousNode->ptrNext = nullptr;
88. tail = previousNode;
89. --size;
90. }
91. template<typename T>
92. void List<T>::insert(const T& element, const unsigned int index)
93. {
94. if (index == 0)
95. pushFront(element);
96. else if (size == (int)index)
97. pushBack(element);
98. else if (size < (int)index)
99. throw std::out\_of\_range("Ошибка(void List<T>::insert(const T &element,

const int index)): Выход за границы списка");

1. else
2. {
3. Node<T>\* nextNode = findElement(index);
4. Node<T>\* newNode = new Node<T>(element, nextNode->ptrPrevious, nextNode);
5. nextNode->ptrPrevious->ptrNext = newNode;
6. nextNode->ptrPrevious = newNode;
7. ++size;
8. }
9. }
10. template<typename T>
11. void List<T>::removeAt(const unsigned int index)
12. {
13. if (size <= (int)index)
14. throw std::exception("Ошибка(void List<T>::removeAt(const int index)):

Выход за границы списка");

1. else if (index == 0)
2. popFront();
3. else if ((int)index == size - 1)
4. popBack();
5. else
6. {
7. Node<T>\* currentNode = findElement(index);
8. if (currentNode->ptrNext != nullptr)
9. currentNode->ptrNext->ptrPrevious = currentNode->ptrPrevious;
10. if (currentNode->ptrPrevious != nullptr)
11. currentNode->ptrPrevious->ptrNext = currentNode->ptrNext;
12. delete currentNode;
13. --size;
14. }
15. }
16. template<typename T>
17. void List<T>::remove(const T& element, RemoveMode mode)
18. {
19. Node<T>\* currentNode = head;
20. while (currentNode != nullptr)
21. {
22. if (currentNode->data == element)
23. {
24. Node<T>\* tempNode = currentNode->ptrNext;
25. if (currentNode->ptrNext != nullptr)
26. currentNode->ptrNext->ptrPrevious = currentNode-

>ptrPrevious;

1. else
2. tail = currentNode->ptrPrevious;
3. if (currentNode->ptrPrevious != nullptr)
4. currentNode->ptrPrevious->ptrNext = currentNode->ptrNext;
5. else
6. head = currentNode->ptrNext;
7. delete currentNode;
8. currentNode = tempNode;
9. --size;
10. if (mode == All)
11. continue;
12. else
13. break;
14. }
15. currentNode = currentNode->ptrNext;
16. }
17. }
18. template<typename T>
19. int List<T>::firstIndexOf(const T& element) const
20. {
21. Node<T>\* currentNode = head;
22. for (int i = 0; i < size; ++i)
23. {
24. if (currentNode->data == element)
25. return i;
26. currentNode = currentNode->ptrNext;
27. }
28. return -1;
29. }
30. template<typename T>
31. bool List<T>::contains(const T& element) const
32. {
33. return firstIndexOf(element) != -1;
34. }
35. template<typename T>
36. void List<T>::clear()
37. {
38. while (size != 0)
39. popFront();
40. }
41. template<typename T>
42. bool List<T>::isEmpty() const
43. {
44. return size == 0;
45. }
46. template<typename T>
47. int List<T>::getSize() const
48. {
49. return size;
50. }

**KeyVerifier.h**

1. #pragma once
2. #include <string>
3. #include <cstdint>
4. #define KEY\_HASH 11194
5. #define KEY\_SIZE 128
6. #define KEY\_FILENAME "kts\_key.dat"
7. class KeyVerifier {
8. public:
9. static bool verify();
10. private:
11. static bool isRemovableDrive(const std::string& drivePath);
12. static uint32\_t getHash(const std::string& data);
13. };

**KeyVerifier.cpp**

1. #include "KeyVerifier.h"
2. #include "FileError.h"
3. #include "windows.h"
4. #include <fstream>
5. #include <iostream>
6. uint32\_t KeyVerifier::getHash(const std::string& data) {
7. uint32\_t hash = 0;
8. for (unsigned char byte : data) {
9. hash += byte;
10. }
11. return hash;
12. }
13. bool KeyVerifier::isRemovableDrive(const std::string& drivePath) {
14. UINT driveType = GetDriveTypeA(drivePath.c\_str());
15. return driveType == DRIVE\_REMOVABLE;
16. }
17. bool KeyVerifier::verify() {
18. DWORD drives = GetLogicalDrives();
19. if (drives == 0) {
20. return false;
21. }
22. for (char drive = 'A'; drive <= 'Z'; ++drive) {
23. if (drives & (1 << (drive - 'A'))) {
24. std::string drivePath = std::string(1, drive) + ":\\";
25. if (!isRemovableDrive(drivePath)) {
26. continue;
27. }
28. std::ifstream file(drivePath + KEY\_FILENAME, std::ios::binary);
29. if (!file.is\_open()) {
30. continue;
31. }
32. std::string fileContent((std::istreambuf\_iterator<char>(file)),
33. std::istreambuf\_iterator<char>());
34. file.close();
35. // std::cout << getHash(fileContent) << std::endl;
36. // std::cout << fileContent.size() << std::endl;
37. if (fileContent.size() == KEY\_SIZE && getHash(fileContent) ==
38. KEY\_HASH)
39. {
40. return true;
41. }
42. }
43. }
44. return false;
45. }

**InputError.h**

1. #pragma once
2. #include <stdexcept>
3. class InputError : public std::exception {
4. public:
5. explicit InputError(const std::string& message) : message\_(message) {}
6. const char\* what() const noexcept override {
7. return message\_.c\_str();
8. }
9. private:
10. std::string message\_;
11. };

**FileError.h**

1. #pragma once
2. #include <stdexcept>
3. class FileError : public std::exception {
4. public:
5. explicit FileError(const std::string& message) : message\_(message) {}
6. const char\* what() const noexcept override {
7. return message\_.c\_str();
8. }
9. private:
10. std::string message\_;
11. };