ПРИЛОЖЕНИЕ Г

(обязательное)

Исходный текст программы (с комментариями)

TestingSystem.cpp

```
001 #include "Menu.h"
002 #include <windows.h>
003
004 int main()
005 {
006     SetConsoleOutputCP(1251);
007     SetConsoleCP(1251);
008     Menu menu;
009     menu.displayRoleSelectionMenu();
010 }
```

Menu.h

```
001 #pragma once
002 #include <iostream>
003 #include "Test.h"
004 #include "SingleChoiceQuestion.h"
005 #include "OpenAnswerQuestion.h"
006 #include "MultipleChoiceQuestion.h"
007
008 #define MIN_FULL_NAME_LENGTH 7
009 #define MAX_FULL_NAME_LENGTH 25
010 #define MIN_LAST_NAME_LENGTH 2
011 #define INITIALS_LENGTH 4
012 #define MIN_TEST_QUESTIONS 3
013
014 using namespace std;
015
016 class Menu {
017 public:
018     Menu();
019     ~Menu();
020     void displayRoleSelectionMenu();
021     static int getValidatedNumber(const std::string& input, int min, int max);
022     static char getInput(const string& validOptions);
023 private:
024     void displayAdminMenu();
025     void displayStudentMenu(const string& studentName);
026     void checkStudentName(const string& studentName);
027     void startTesting(const string& studentName);
028     void displayTestList();
029     void addTest();
030     void deleteTest();
031     void editTest();
032     void editTestDetails(Test* test);
033     bool saveTest(Test* test);
034     void addQuestionsToTest(Test* newTest);
035     void addSingleChoiceQuestion(Test* test);
036     void addMultipleChoiceQuestion(Test* test);
037     void addOpenAnswerQuestion(Test* test);
038     void changeQuestion(SingleChoiceQuestion* question);
039     void changeQuestion(MultipleChoiceQuestion* question);
040     void changeQuestion(OpenAnswerQuestion* question);
041     bool setQuestionTitle(Question* question);
```

```
042     bool setQuestionOptions(ChoiceQuestion* question);
043     bool handleExistingOption(ChoiceQuestion* question, char& option, unsigned
044  int index);
045     bool replaceOption(ChoiceQuestion* question, char option, unsigned int
    index);
046     bool handleNewOption(ChoiceQuestion* question, char& option);
047     bool setAnswer(SingleChoiceQuestion* question);
048     bool setAnswer(MultipleChoiceQuestion* question);
049     bool setAnswer(OpenAnswerQuestion* question);
050     void changeTestTitle(Test* test);
051     void deleteQuestionFromTest(Test* test);
052     void editQuestionInTest(Test* test);
053     int printTestList();
054     void printQuestion(Question* question);
055     void sortTests();
056
057     List<Test*> tests;
058 };
```

Menu.cpp

```
001 #include "Menu.h"
002 #include <limits>
003 #include <string>
004 #include <cstdlib>
005 #include <functional>
006 #include "InputError.h"
007 #include "FileError.h"
008 #include "KeyVerifier.h"
009 #include "TestSession.h"
010 #include "StatisticMenu.h"
011
012 Menu::Menu()
013 {
014     tests = Test::loadAllTests();
015 }
016
017 Menu::~Menu()
018 {
019     for (auto test : tests) {
020         delete test;
021     }
022     tests.clear();
023 }
024
025 void Menu::displayRoleSelectionMenu()
026 {
027     char choice = 0;
028     cout << "Добро пожаловать!¥n";
029     do {
030         cout << "Выберите вашу роль:¥n";
031         cout << "1. Учитель/Администратор¥n";
032         cout << "2. Ученик¥n";
033         cout << "0. Выход¥n";
034
035         try {
036             choice = getInput("012");
037         }
038         catch (const InputError& e) {
039             cout << e.what() << endl;
040             continue;
041         }
```

```
042
043         switch (choice) {
044         case '1':
045             for (;;) {
046                 if (KeyVerifier::verify()) {
047                     displayAdminMenu();
048                     break;
049                 }
050
051                 cout << "Верификация не пройдена. Проверьте наличие ключа и
    попробуйте снова.¥n";
052                 cout << "Введите 0 для выхода или любую другую клавишу для
    повторной попытки: ";
053                 string adminChoice;
054                 getline(cin, adminChoice);
055                 system("cls");
056                 if (adminChoice == "0") {
057                     break;
058                 }
059             }
060             break;
061         case '2':
062             for (;;) {
063                 cout << "Введите вашу фамилию и инициалы (Фамилия И.О.) или '0'
    для возврата: ";
064                 string studentName;
065                 getline(cin, studentName);
066                 system("cls");
067                 if (studentName == "0") break;
068                 try {
069                     checkStudentName(studentName);
070                     cout << "Добро пожаловать, " << studentName << "!¥n";
071                     displayStudentMenu(studentName);
072                     break;
073                 }
074                 catch (InputError& e) {
075                     cout << "Ошибка: " << e.what() << endl;
076                 }
077             }
078             break;
079         }
080     } while (choice != '0');
081     cout << "Выход из программы...¥n";
082 }
083
084 void Menu::displayAdminMenu()
085 {
086     char choice = 0;
087     do {
088         cout << "Меню учителя/администратора:¥n";
089         cout << "1. Список тестов¥n";
090         cout << "2. Добавить тест¥n";
091         cout << "3. Удалить тест¥n";
092         cout << "4. Редактировать тест¥n";
093         cout << "5. Просмотр статистики¥n";
094         cout << "0. Вернуться в меню выбора роли¥n";
095
096         try {
097             choice = getInput("012345");
098         }
099         catch (const InputError& e) {
100             cout << e.what() << endl;
```

```
101             continue;
102         }
103 
104         switch (choice) {
105         case '1':
106             displayTestList();
107             break;
108         case '2':
109             addTest();
110             break;
111         case '3':
112             deleteTest();
113             break;
114         case '4':
115             editTest();
116             break;
117         case '5':
118             {
119                 auto statMenu = new StatisticMenu();
120                 statMenu->start();
121                 delete statMenu;
122                 break;
123             }
124         }
125     } while (choice != '0');
126 }
127 
128 void Menu::displayStudentMenu(const string& studentName)
129 {
130     char choice = 0;
131     do {
132         cout << "Меню ученика:¥n";
133         cout << "1. Начать тестирование¥n";
134         cout << "0. Вернуться в меню выбора роли¥n";
135 
136         try {
137             choice = getInput("01");
138         }
139         catch (const InputError& e) {
140             cout << e.what() << endl;
141             continue;
142         }
143 
144         switch (choice) {
145         case '1':
146             startTesting(studentName);
147             break;
148         }
149     } while (choice != '0');
150 }
151 
152 
153 void Menu::startTesting(const string& studentName)
154 {
155     do {
156         int maxIndex = printTestList();
157         if (maxIndex == 0) {
158             cout << "Список тестов пуст.¥n¥nВведите любой символ для
    возврата...";
159             string input;
160             getline(cin, input);
161             system("cls");
```

```
162             return;
163         }
164 
165         cout << "¥nВведите номер теста для продолжения, либо 0 для возврата: ";
166         string input;
167         getline(cin, input);
168         system("cls");
169         if (input == "0") return;
170         try {
171             int selectedIndex = getValidatedNumber(input, 1, maxIndex);
172             auto session = new TestSession(tests[selectedIndex - 1],
    studentName);
173             if (!session->prepare()) {
174                 delete session;
175                 continue;
176             }
177             session->start();
178             delete session;
179             return;
180         }
181         catch (const InputError& e)
182         {
183             cout << "Ошибка: " << e.what() << "¥n";
184         }
185     } while (true);
186 }
187 
188 void Menu::displayTestList()
189 {
190     while (true) {
191         int maxIndex = printTestList();
192         if (maxIndex == 0) {
193             cout << "Список тестов пуст.¥n¥nВведите любой символ для
    возврата...";
194             string input;
195             getline(cin, input);
196             system("cls");
197             return;
198         }
199 
200         cout << "¥nВведите номер теста для просмотра, либо 0 для возврата: ";
201         string input;
202         getline(cin, input);
203         system("cls");
204         if (input == "0") return;
205         try {
206             int selectedIndex = getValidatedNumber(input, 1, maxIndex);
207             tests[selectedIndex - 1]->print();
208             cout << "¥nВведите любой символ для возврата...";
209             string input;
210             getline(cin, input);
211             system("cls");
212         }
213         catch (const InputError& e)
214         {
215             cout << "Ошибка: " << e.what() << "¥n";
216         }
217     }
218 }
219 
220 void Menu::addTest()
221 {
```

```
222     cout << "Добавление нового теста¥n";
223     do {
224         cout << "Введите название теста (или 0 для отмены): ";
225         string testTitle;
226         getline(cin, testTitle);
227         system("cls");
228 
229         if (testTitle == "0") return;
230         try {
231             Test* newTest = new Test(testTitle);
232             do {
233                 addQuestionsToTest(newTest);
234                 if (newTest->getQuestionCount() >= MIN_TEST_QUESTIONS) {
235                     if (saveTest(newTest)) {
236                         tests.pushBack(newTest);
237                         sortTests();
238                         cout << "Тест успешно сохранен.¥n";
239                     }
240                     else {
241                         delete newTest;
242                         cout << "Тест не был сохранен.¥n";
243                     }
244                     return;
245                 }
246                 else {
247                     cout << "Тест содержит недостаточно вопросов.¥n";
248                     cout << "Введите любой символ для продолжения (или 0 для
    выхода без сохранения):¥n";
249                     string userInput;
250                     getline(cin, userInput);
251                     system("cls");
252                     if (userInput == "0") {
253                         return;
254                     }
255                 }
256             } while (true);
257             return;
258         }
259         catch (const InputError& e) {
260             cout << "Ошибка: " << e.what() << "¥n";
261         }
262     } while (true);
263 }
264 
265 bool Menu::saveTest(Test* test)
266 {
267     do {
268         try {
269             test->saveToFile();
270             return true;
271         }
272         catch (const FileError& e) {
273             cout << "Ошибка: " << e.what() << endl;
274             cout << "Введите 0 для выхода без сохранения или любой другой символ
    для повторной попытки: ";
275             string input;
276             getline(cin, input);
277             system("cls");
278             if (input == "0") {
279                 return false;
280             }
281         }
```

```
282     } while (true);
283 }
284
285 void Menu::deleteTest()
286 {
287     while (true) {
288         int maxIndex = printTestList();
289         if (maxIndex == 0) {
290             cout << "Нет тестов для удаления.¥n¥nВведите любой символ для
    возврата...";
291             string input;
292             getline(cin, input);
293             system("cls");
294             return;
295         }
296
297         cout << "¥nВведите номер теста для удаления (или 0 для отмены): ";
298         string input;
299         getline(cin, input);
300         if (input == "0") {
301             system("cls");
302             return;
303         }
304         try {
305             int selectedIndex = getValidatedNumber(input, 1, maxIndex);
306
307             cout << "Вы действительно хотите удалить тест ¥"" <<
308 tests[selectedIndex - 1]->getTitle()
309                  << "¥"?¥n(Введите Y для подтверждения или любой другой символ для
    отмены): ";
310             getline(cin, input);
311             system("cls");
312
313             if (input != "Y" && input != "y") {
314                 cout << "Тест не был удалён.¥n";
315                 continue;
316             }
317
318             tests[selectedIndex - 1]->deleteTestFile();
319             tests.removeAt(selectedIndex - 1);
320             cout << "Тест успешно удалён!¥n";
321         }
322         catch (const InputError& e)
323         {
324             system("cls");
325             cout << "Ошибка: " << e.what() << "¥n";
326         }
327         catch (const FileError& e) {
328             cout << "Ошибка: " << e.what() << "¥n";
329             return;
330         }
331     }
332 }
333
334 void Menu::editTest()
335 {
336     while (true) {
337         int maxIndex = printTestList();
338         if (maxIndex == 0) {
339             cout << "Нет тестов для изменения.¥n¥nВведите любой символ для
    возврата...";
340             string input;
```

```
341             getline(cin, input);
342             system("cls");
343             return;
344         }
345 
346         cout << "¥nВведите номер теста для изменения (или 0 для отмены): ";
347         string input;
348         getline(cin, input);
349         system("cls");
350         if (input == "0") return;
351         try {
352             int selectedIndex = getValidatedNumber(input, 1, maxIndex);
353             auto test = tests[selectedIndex - 1];
354             editTestDetails(test);
355             break;
356         }
357         catch (const InputError& e)
358         {
359             cout << "Ошибка: " << e.what() << "¥n";
360         }
361     }
362 }
363 
364 void Menu::editTestDetails(Test* test)
365 {
366     auto changedTest = test;
367     char choice = 0;
368     while (true) {
369         cout << "Редактирование теста ¥"" << test->getTitle() << "¥":¥n";
370         cout << "1. Изменить название теста¥n";
371         cout << "2. Добавить новые вопросы¥n";
372         cout << "3. Удалить вопрос¥n";
373         cout << "4. Изменить существующий вопрос¥n";
374         cout << "0. Вернуться в меню администратора¥n";
375 
376         try {
377             choice = getInput("01234");
378             switch (choice) {
379             case '1': {
380                 changeTestTitle(test);
381                 break;
382             }
383             case '2':
384                 addQuestionsToTest(test);
385                 break;
386             case '3':
387                 deleteQuestionFromTest(test);
388                 break;
389             case '4':
390                 editQuestionInTest(test);
391                 break;
392             case '0':
393                 {
394                     if (changedTest->getQuestionCount() >= MIN_TEST_QUESTIONS) {
395                         if (saveTest(changedTest)) {
396                             test = changedTest;
397                             sortTests();
398                             cout << "Тест успешно сохранен.¥n";
399                         }
400                         else {
401                             delete changedTest;
402                             cout << "Тест не был сохранен.¥n";
```

```
                        }
                        return;
                    }
                    else {
                        cout << "Тест содержит недостаточно вопросов.¥n";
                        cout << "Введите любой символ для продолжения (или 0 для выхода без сохранения):¥n";
                        string userInput;
                        getline(cin, userInput);
                        system("cls");
                        if (userInput == "0") {
                            return;
                        }
                    }
                    break;
                }
            }
        }
        catch (const InputError& e) {
            cout << "Ошибка: " << e.what() << "¥n";
        }
    }
}

void Menu::addQuestionsToTest(Test* newTest)
{
    char choice;
    do {
        cout << "Добавление нового вопроса:¥n";
        cout << "1. Вопрос с одним правильным ответом¥n";
        cout << "2. Вопрос с несколькими правильными ответами¥n";
        cout << "3. Вопрос со свободным ответом¥n";
        cout << "0. Завершить добавление вопросов¥n";
        try {
            choice = getInput("0123");
        }
        catch (const InputError& e) {
            cout << "Ошибка: " << e.what() << endl;
            continue;
        }

        switch (choice) {
        case '1': {
            addSingleChoiceQuestion(newTest);
            break;
        }
        case '2': {
            addMultipleChoiceQuestion(newTest);
            break;
        }
        case '3': {
            addOpenAnswerQuestion(newTest);
            break;
        }
        case '0':
            return;
        }
    } while (true);
}

void Menu::addSingleChoiceQuestion(Test* test)
{
```

```
464     SingleChoiceQuestion* question = new SingleChoiceQuestion();
465     cout << "Добавление вопроса с одним правильным ответом¥n";
466     if (!setQuestionTitle(question) || !setQuestionOptions(question)
    || !setAnswer(question)) {
467         delete question;
468         cout << "Вопрос не был добавлен!¥n";
469         return;
470     }
471     test->addQuestion(question);
472     cout << "Вопрос успешно добавлен!¥n";
473 }
474
475 void Menu::addMultipleChoiceQuestion(Test* test)
476 {
477     MultipleChoiceQuestion* question = new MultipleChoiceQuestion();
478     cout << "Добавление вопроса с несколькими правильным ответом¥n";
479     if (!setQuestionTitle(question) || !setQuestionOptions(question)
    || !setAnswer(question)) {
480         delete question;
481         cout << "Вопрос не был добавлен!¥n";
482         return;
483     }
484     test->addQuestion(question);
485     cout << "Вопрос успешно добавлен!¥n";
486 }
487
488 void Menu::addOpenAnswerQuestion(Test* test)
489 {
490     OpenAnswerQuestion* question = new OpenAnswerQuestion();
491     cout << "Добавление вопроса со свободным ответом¥n";
492     if (!setQuestionTitle(question) || !setAnswer(question)) {
493         delete question;
494         cout << "Вопрос не был добавлен!¥n";
495         return;
496     }
497     test->addQuestion(question);
498     cout << "Вопрос успешно добавлен!¥n";
499 }
500
501 void Menu::changeTestTitle(Test* test)
502 {
503     while (true) {
504         cout << "Текущее название теста: ¥"" << test->getTitle() << "¥"¥n";
505         cout << "Введите новое название теста (или 0 для отмены): ";
506         string newTitle;
507         getline(cin, newTitle);
508         system("cls");
509         if (newTitle == "0") return;
510         try {
511             test->setTitle(newTitle);
512             cout << "Название теста успешно изменено.¥n";
513             return;
514         }
515         catch (const InputError& e) {
516             cout << "Ошибка: " << e.what() << "¥n";
517         }
518     }
519 }
520
521 void Menu::deleteQuestionFromTest(Test* test)
522 {
523     while (true) {
```

```
524         if (test->getQuestionCount() == 0) {
525             cout << "В данном тесте нет вопросов для удаления.¥n¥nВведите любой
    символ для возврата...";
526             string input;
527             getline(cin, input);
528             system("cls");
529             return;
530         }
531 
532         cout << "Список вопросов теста ¥"" << test->getTitle() << "¥":¥n";
533         for (int i = 0; i < test->getQuestionCount(); ++i) {
534             cout << i + 1 << ". " << test->getQuestion(i)->getTitle() << "¥n";
535         }
536 
537         cout << "¥nВведите номер вопроса для удаления (или 0 для отмены): ";
538         string input;
539         getline(cin, input);
540         if (input == "0") {
541             system("cls");
542             return;
543         }
544 
545         try {
546             int selectedIndex = getValidatedNumber(input, 1, test-
    >getQuestionCount());
547 
548             cout << "Вы действительно хотите удалить вопрос ¥"" << test-
    >getQuestion(selectedIndex - 1)->getTitle()
549                 << "¥"?¥n(Введите Y для подтверждения или любой другой символ для
550 отмены): ";
551             getline(cin, input);
552             system("cls");
553 
554             if (input != "Y" && input != "y") {
555                 cout << "Вопрос не был удалён.¥n";
556                 continue;
557             }
558 
559             test->removeQuestion(selectedIndex - 1);
560             cout << "Вопрос успешно удалён!¥n";
561         }
562         catch (const InputError& e) {
563             system("cls");
564             cout << "Ошибка: " << e.what() << "¥n";
565         }
566     }
567 }
568 
569 void Menu::editQuestionInTest(Test* test)
570 {
571     while (true) {
572         if (test->getQuestionCount() == 0) {
573             cout << "В данном тесте нет вопросов для редактирования.¥n¥nВведите
    любой символ для возврата...";
574             string input;
575             getline(cin, input);
576             system("cls");
577             return;
578         }
579 
580         cout << "Список вопросов теста ¥"" << test->getTitle() << "¥":¥n";
581         for (int i = 0; i < test->getQuestionCount(); ++i) {
```

```
582             cout << i + 1 << ". " << test->getQuestion(i)->getTitle() << "¥n";
583         }
584 
585         cout << "¥nВведите номер вопроса для редактирования (или 0 для отмены):
    ";
586         string input;
587         getline(cin, input);
588         system("cls");
589         if (input == "0") {
590             return;
591         }
592         try {
593             int selectedIndex = getValidatedNumber(input, 1, test-
    >getQuestionCount());
594             Question* question = test->getQuestion(selectedIndex - 1);
595             if (auto singleChoiceQuestion =
    dynamic_cast<SingleChoiceQuestion*>(question)) {
596                 changeQuestion(singleChoiceQuestion);
597             }
598             else if (auto multipleChoiceQuestion =
    dynamic_cast<MultipleChoiceQuestion*>(question)) {
599                 changeQuestion(multipleChoiceQuestion);
600             }
601             else if (auto openAnswerQuestion =
    dynamic_cast<OpenAnswerQuestion*>(question)) {
602                 changeQuestion(openAnswerQuestion);
603             }
604             else {
605                 cout << "Ошибка: тип вопроса не поддерживается.¥n";
606             }
607         }
608         catch (const InputError& e) {
609             cout << "Ошибка: " << e.what() << "¥n";
610         }
611     }
612 }
613 
614 void Menu::changeQuestion(SingleChoiceQuestion* question)
615 {
616     while (true) {
617         cout << "Редактирование вопроса ¥"" << question->getTitle() << "¥":¥n";
618         cout << "1. Изменить заголовок¥n";
619         cout << "2. Изменить варианты ответа¥n";
620         cout << "3. Изменить правильный ответ¥n";
621         cout << "4. Показать вопрос¥n";
622         cout << "0. Вернуться¥n";
623 
624         char choice = getInput("01234");
625 
626         try {
627             switch (choice) {
628             case '1':
629                 setQuestionTitle(question);
630                 break;
631             case '2':
632             {
633                 auto changedQuestion = new SingleChoiceQuestion(*question);
634                 setQuestionOptions(changedQuestion);
635                 if (changedQuestion->getOptionCount() != question-
    >getOptionCount()) {
636                     if (!setAnswer(changedQuestion)) {
637                         delete changedQuestion;
```

```
638                         break;
639                     }
640                 }
641                 *question = *changedQuestion;
642                 delete changedQuestion;
643                 break;
644             }
645             case '3':
646                 setAnswer(question);
647                 break;
648             case '4':
649                 printQuestion(question);
650                 break;
651             case '0':
652                 return;
653             }
654         }
655         catch (const InputError& e) {
656             cout << "Ошибка: " << e.what() << "¥n";
657         }
658     }
659 }
660 void Menu::changeQuestion(MultipleChoiceQuestion* question)
661 {
662     while (true) {
663         cout << "Редактирование вопроса ¥"" << question->getTitle() << "¥":¥n";
664         cout << "1. Изменить заголовок¥n";
665         cout << "2. Изменить варианты ответа¥n";
666         cout << "3. Изменить правильные ответы¥n";
667         cout << "4. Показать вопрос¥n";
668         cout << "0. Вернуться¥n";
669 
670         char choice = getInput("01234");
671 
672         try {
673             switch (choice) {
674             case '1':
675                 setQuestionTitle(question);
676                 break;
677             case '2':
678             {
679                 auto changedQuestion = new MultipleChoiceQuestion(*question);
680                 setQuestionOptions(changedQuestion);
681                 if (changedQuestion->getOptionCount() != question-
    >getOptionCount()) {
682                     if (!setAnswer(changedQuestion)) {
683                         delete changedQuestion;
684                         break;
685                     }
686                 }
687                 *question = *changedQuestion;
688                 delete changedQuestion;
689                 break;
690             }
691             case '3':
692                 setAnswer(question);
693                 break;
694             case '4':
695                 printQuestion(question);
696                 break;
697             case '0':
698                 return;
```

```
699             }
700         }
701         catch (const InputError& e) {
702             cout << "Ошибка: " << e.what() << "¥n";
703         }
704     }
705 }
706
707 void Menu::changeQuestion(OpenAnswerQuestion* question)
708 {
709     while (true) {
710         cout << "Редактирование вопроса ¥"" << question->getTitle() << "¥":¥n";
711         cout << "1. Изменить заголовок¥n";
712         cout << "2. Изменить ответ¥n";
713         cout << "3. Показать вопрос¥n";
714         cout << "0. Вернуться¥n";
715
716         char choice = getInput("0123");
717
718         try {
719             switch (choice) {
720             case '1':
721                 setQuestionTitle(question);
722                 break;
723             case '2':
724                 setAnswer(question);
725                 break;
726             case '3':
727                 printQuestion(question);
728                 break;
729             case '0':
730                 return;
731             }
732         }
733         catch (const InputError& e) {
734             cout << "Ошибка: " << e.what() << "¥n";
735         }
736     }
737 }
738
739 char Menu::getInput(const string& validOptions) {
740     string input;
741     cout << "Введите выбор: ";
742     getline(cin, input);
743     system("cls");
744     if (input.length() == 1) {
745         if (validOptions.find(tolower(input[0])) == string::npos) throw
    InputError("Некорректный ввод. Операции (" + input + ") не существует.");
746         return input[0];
747     }
748
749     throw InputError("Некорректный ввод. Введите один символ.");
750 }
751
752 void Menu::checkStudentName(const string& studentName) {
753     if (studentName.size() < MIN_FULL_NAME_LENGTH || studentName.size() >
    MAX_FULL_NAME_LENGTH) {
754         throw InputError("Некорректный формат ФИО. Длина должна быть от "
755             + std::to_string(MIN_FULL_NAME_LENGTH) + " до "
756             + std::to_string(MAX_FULL_NAME_LENGTH)
757             + " символов. Пример: Иванов И.И.");
758     }
```

```
759
760     size_t spacePos = studentName.find(' ');
761     if (spacePos == string::npos || spacePos == 0 || spacePos ==
    studentName.size() - 1) {
762         throw InputError("Некорректный формат ФИО. Пример: Иванов И.И.");
763     }
764
765     string lastName = studentName.substr(0, spacePos);
766     string initials = studentName.substr(spacePos + 1);
767
768     if (lastName.empty() || ((lastName[0] < 'А' || lastName[0] > 'Я') &&
    lastName[0] != 'Ё')) {
769         throw InputError("Фамилия должна начинаться с заглавной буквы.");
770     }
771
772     if (lastName.size() < MIN_LAST_NAME_LENGTH) {
773         throw InputError("Фамилия слишком короткая.");
774     }
775
776     for (size_t i = 1; i < lastName.size(); ++i) {
777         if ((lastName[i] < 'а' || lastName[i] > 'я') && lastName[i] != 'ё') {
778             throw InputError("Фамилия должна содержать только русские буквы и
    соответствовать формату ¥"Фамилия¥"");
779         }
780     }
781
782     if (initials.size() != INITIALS_LENGTH || initials[1] != '.' ||
    initials[3] != '.' ||
783         ((initials[0] < 'А' || initials[0] > 'Я') && initials[0] != 'Ё') ||
784         ((initials[2] < 'А' || initials[2] > 'Я') && initials[2] != 'Ё')) {
785         throw InputError("Инициалы должны быть в формате И.О. с заглавными
    русскими буквами.");
786     }
787 }
788
789 bool Menu::setQuestionTitle(Question* question) {
790     auto currentTitle = question->getTitle();
791     do {
792         if (!currentTitle.empty()) {
793             cout << "Текущий заголовок: ¥"" << question->getTitle() << "¥"¥n";
794         }
795         cout << "Введите " << (!currentTitle.empty() ? "новый " : "") <<
    "заголовок вопроса (или 0 для отмены): ";
796         string questionText;
797         getline(cin, questionText);
798         system("cls");
799         if (questionText == "0") {
800             return false;
801         }
802         try {
803             question->setTitle(questionText);
804             return true;
805         }
806         catch (const InputError& e) {
807             cout << "Ошибка: " << e.what() << "¥n";
808         }
809     } while (true);
810 }
811
812 bool Menu::setQuestionOptions(ChoiceQuestion* question) {
813     char option = 'A';
814
```

```
815     while (option <= 'Z') {
816         int index = option - 'A';
817         if (index < question->getOptionCount()) {
818             if (!handleExistingOption(question, option, index)) {
819                 return true;
820             }
821         }
822         else {
823             int startCount = question->getOptionCount();
824             if (!handleNewOption(question, option)) {
825                 return false;
826             }
827             if (startCount == question->getOptionCount()) {
828                 return true;
829             }
830         }
831     }
832     return true;
833 }
834
835 bool Menu::handleExistingOption(ChoiceQuestion* question, char& option, unsigned
    int index) {
836     while (true) {
837         cout << "Текущий вариант ответа " << option << ": " << question-
    >getOption(index) << "¥n";
838         cout << "Выберите действие:¥n";
839         cout << "1. Заменить¥n";
840         cout << "2. Удалить¥n";
841         cout << "3. Пропустить¥n";
842         cout << "0. Завершить изменение вариантов¥n";
843         char choice;
844         try {
845             choice = getInput("0123");
846         }
847         catch (const InputError& e) {
848             cout << "Ошибка: " << e.what() << "¥n";
849             continue;
850         }
851         switch (choice) {
852         case '1':
853             if (replaceOption(question, option, index)) {
854                 cout << "Вариант ответа " << option << " успешно изменен.¥n";
855                 option++;
856                 return true;
857             }
858             continue;
859         case '2':
860             question->removeOption(index);
861             cout << "Вариант ответа " << option << " успешно удалён.¥n";
862             return true;
863         case '3':
864             cout << "Вариант ответа " << option << " пропущен.¥n";
865             option++;
866             return true;
867         case '0':
868             return false;
869         }
870     }
871 }
872
873 bool Menu::replaceOption(ChoiceQuestion* question, char option, unsigned int
    index) {
```

```
874     while (true) {
875         cout << "Введите новый текст для варианта " << option << " (или оставьте
    пустым для отмены): ";
876         string newOptionText;
877         getline(cin, newOptionText);
878         system("cls");
879         if (newOptionText.empty()) {
880             return false;
881         }
882         try {
883             question->addOption(newOptionText, index);
884             question->removeOption(index + 1);
885             return true;
886         }
887         catch (const InputError& e) {
888             cout << "Ошибка: " << e.what() << "¥n";
889         }
890     }
891 }
892
893
894 bool Menu::handleNewOption(ChoiceQuestion* question, char& option) {
895     while (true) {
896         cout << "Введите вариант ответа " << option << " (или оставьте пустым для
    завершения): ";
897         string optionText;
898         getline(cin, optionText);
899         system("cls");
900
901         if (optionText.empty()) {
902             if (question->getOptionCount() < 2) {
903                 cout << "Ошибка: Минимум два варианта ответа обязательно.¥n";
904                 cout << "Введите любой символ для продолжения (или 0 для выхода
    без сохранения): ";
905                 string userInput;
906                 getline(cin, userInput);
907                 system("cls");
908                 if (userInput == "0") {
909                     return false;
910                 }
911                 continue;
912             }
913             else {
914                 return true;
915             }
916         }
917         try {
918             question->addOption(optionText);
919             option++;
920             return true;
921         }
922         catch (const InputError& e) {
923             cout << "Ошибка: " << e.what() << "¥n";
924         }
925     }
926 }
927
928 bool Menu::setAnswer(SingleChoiceQuestion* question) {
929     while (true) {
930         cout << "Вопрос: ";
931         question->printQuestion();
932         cout << "Введите правильный вариант ответа (A-" << static_cast<char>('A'
```

```
    + question->getOptionCount() - 1)
933             << ") или 0 для отмены: ";
934         string input;
935         getline(cin, input);
936         system("cls");
937         if (input.size() != 1) {
938             cout << "Ошибка: Ответ должен состоять из одного символа¥n";
939         }
940         if (input == "0") {
941             return false;
942         }
943         try {
944             char correctOption = input[0];
945             question->setCorrectIndex(correctOption);
946             break;
947         }
948         catch (const InputError& e) {
949             cout << "Ошибка: " << e.what() << "¥n";
950         }
951     }
952     return true;
953 }
954
955 bool Menu::setAnswer(MultipleChoiceQuestion* question) {
956     while (true) {
957         cout << "Вопрос: ";
958         question->printQuestion();
959         cout << "Введите правильные варианты ответа (A-" << static_cast<char>('A'
    + question->getOptionCount() - 1)
960             << "), например ¥"AB¥", или 0 для отмены: ";
961         string input;
962         getline(cin, input);
963         system("cls");
964         if (input == "0") {
965             return false;
966         }
967         List<unsigned char> correctOptions;
968         try {
969             for (unsigned char ch : input) {
970                 correctOptions.pushBack(toupper(ch));
971             }
972             question->setCorrectIndices(correctOptions);
973             break;
974         }
975         catch (const InputError& e) {
976             cout << "Ошибка: " << e.what() << "¥n";
977         }
978     }
979     return true;
980 }
981
982 bool Menu::setAnswer(OpenAnswerQuestion* question) {
983     while (true) {
984         cout << "Вопрос: ";
985         question->printQuestion();
986         cout << "Введите правильный ответ, или оставьте пустым для отмены: ";
987         string input;
988         getline(cin, input);
989         system("cls");
990         if (input.empty()) {
991             return false;
992         }
```

```
993         try {
994             question->setAnswer(input);
995             break;
996         }
997         catch (const InputError& e) {
998             cout << "Ошибка: " << e.what() << "¥n";
999         }
1000      }
1001      return true;
1002  }
1003
1004  int Menu::getValidatedNumber(const std::string& input, int min = 0, int max =
1005  100) {
1006      try {
1007          int number = std::stoi(input);
1008
1009          if (number < min || number > max) {
1010              throw InputError("Число должно быть в диапазоне от " +
1011  std::to_string(min) + " до " + std::to_string(max) + ".");
1012          }
1013
1014          return number;
1015      }
1016      catch (const std::invalid_argument&) {
1017          throw InputError("Некорректный формат числа.");
1018      }
1019      catch (const std::out_of_range&) {
1020          throw InputError("Введённое число слишком велико.");
1021      }
1022  }
1023
1024  int Menu::printTestList() {
1025      if (tests.isEmpty()) {
1026          return 0;
1027      }
1028      cout << "Список тестов:¥n";
1029      int  maxIndex = 0;
1030      for (auto test : tests) {
1031          cout << ++maxIndex << ". " << test->getTitle() << "¥n";
1032      }
1033      return maxIndex;
1034  }
1035
1036
1037  void Menu::printQuestion(Question* question) {
1038      cout << "Вопрос: ";
1039      question->printQuestion();;
1040      question->printAnswer();
1041      cout << "¥nВведите любой символ для возврата...";
1042      string input;
1043      getline(cin, input);
1044      system("cls");
1045  }
1046
1047  void Menu::sortTests() {
1048      int n = tests.getSize();
1049
1050      std::function<void(int, int)> heapify;
1051
1052      heapify = [&](int n, int i) {
1053          int largest = i;
1054          int left = 2 * i + 1;
```

```
1055          int right = 2 * i + 2;
1056
1057          if (left < n && tests[left]->getTitle() > tests[largest]->getTitle())
1058  {
1059              largest = left;
1060          }
1061
1062          if (right < n && tests[right]->getTitle() > tests[largest]-
    >getTitle())
1063   {
1064              largest = right;
1065          }
1066
1067          if (largest != i) {
1068              std::swap(tests[i], tests[largest]);
1069              heapify(n, largest);
1070          }
1071      };
1072
1073      for (int i = n / 2 - 1; i >= 0; --i) {
1074          heapify(n, i);
1075      }
1076
1077      for (int i = n - 1; i > 0; --i) {
1078          std::swap(tests[0], tests[i]);
1079          heapify(i, 0);
1080      }
1081  }
1082
```

TestSession.h

```
001 #pragma once
002 #include "Test.h"
003 #include "InputError.h"
004 #include <iostream>
005 #include <string>
006 #include <cstdlib>
007
008 class TestSession {
009 public:
010     TestSession(Test* test, std::string name);
011     void start();
012     bool prepare();
013 private:
014     void askQuestion(Question* question, int number);
015     void showResult();
016
017     Test* test;
018     std::string name;
019     List<std::string> userAnswers;
020     int correctAnswers;
021     int score;
022 };
023
```

TestSession.cpp

```
001 #include "TestSession.h"
002 #include "Statistic.h"
003
```

```
004 TestSession::TestSession(Test* test, std::string name): name(name)
005 {
006     this->test = new Test(*test);
007     userAnswers = {};
008     correctAnswers = 0;
009     score = 0;
010 }
011
012 bool TestSession::prepare()
013 {
014     if (!test || name.empty()) {
015         std::cout << "Необходимо указать тест и имя тестируемого.¥n";
016         return false;
017     }
018     std::cout << "Тест: " << test->getTitle();
019     std::cout << "¥nКоличество вопросов: " << test->getQuestionCount() << "¥n";
020     std::cout << "¥nЖелаете начать тест?¥n";
021     std::cout << "(Введите Y для подтверждения или любой другой символ для
    отмены): ";
022
023     std::string input;
024     getline(std::cin, input);
025     system("cls");
026     if (input != "Y" && input != "y") {
027         return false;
028     }
029
030     test->shuffleQuestions();
031     correctAnswers = 0;
032     userAnswers.clear();
033     return true;
034 }
035
036 void TestSession::start()
037 {
038     auto stat = new Statistic(name, test);
039     int max = test->getQuestionCount();
040     for (int i = 0; i < max; ++i) {
041         Question* question = test->getQuestion(i);
042         askQuestion(question, i + 1);
043     }
044     stat->setAnswers(userAnswers);
045     stat->setCorrectAnswersCount(correctAnswers);
046     stat->saveToFile();
047     delete stat;
048     showResult();
049 }
050
051 void TestSession::askQuestion(Question* question, int number)
052 {
053     do {
054         std::cout << "Вопрос " << number << ": ";
055         question->printQuestion();
056         switch (question->getType()) {
057         case QuestionType::SingleChoice:
058             std::cout << "Введите правильный вариант ответа ";
059             break;
060         case QuestionType::MultipleChoice:
061             std::cout << "Введите правильные варианты ответа ";
062             break;
063         case QuestionType::OpenAnswer:
064             std::cout << "Введите ответ ";
```

```
065             break;
066         default:
067             throw InputError("Вопрос имеет неизвестный тип");
068         }
069         std::cout << "(либо введите пустым для пропуска): ";
070         std::string input;
071         getline(std::cin, input);
072         if (input.empty()) {
073             std::cout << "¥nВы уверены, что хотите пропустить вопрос?¥n";
074             std::cout << "!!! Вы не сможете вернуться к этому вопросу !!!¥n";
075             std::cout << "(Введите Y для подтверждения или любой другой символ
    для отмены): ";
076             getline(std::cin, input);
077             system("cls");
078             if (input != "Y" && input != "y") continue;
079             userAnswers.pushBack(SKIP);
080             return;
081         }
082         system("cls");
083         try {
084             bool result = question->checkAnswer(input);
085             if (result) correctAnswers++;
086             userAnswers.pushBack(input);
087             return;
088         }
089         catch (const InputError& e) {
090             std::cout << "Ошибка: " << e.what() << std::endl;
091         }
092     } while (true);
093
094 }
095
096 void TestSession::showResult()
097 {
098     score = (int)std::round(((double)(correctAnswers) / test->getQuestionCount())
    * 100.0);
099
100     std::cout << "Тест завершён! Ваш результат: " << correctAnswers << " из "
101         << test->getQuestionCount() << " (" << score << "%).¥n";
102     std::cout << "¥nВведите любой символ для возврата...";
103     std::string input;
104     getline(std::cin, input);
105     system("cls");
106 }
107
```

Statistic.h

```
001 #pragma once
002 #include "Test.h"
003 #include <string>
004 #include <vector>
005 #include <chrono>
006
007 #define SKIP "Пропущен"
008 #define STAT_FILE "stat.dat"
009
010 class Statistic {
011 public:
012     Statistic(const std::string name, Test* test);
013     void setCurrentTime();
014     void setAnswers(const List<std::string>& answers);
```

```
015     void setTime(const std::string time);
016     void setCorrectAnswersCount(int count);
017     std::string getTestTitle();
018     std::string getUserName();
019     std::string getTime();
020     void print();
021     void saveToFile();
022     static Statistic* loadFromStream(std::ifstream& in);
023     static List<Statistic*> loadAllFromFile();
024 private:
025     std::string name;
026     std::string time;
027     List<std::string> answers;
028     int correctAnswersCount;
029     int score;
030     Test* test;
031 };
032
```

Statistic.cpp

```
001 #include "Statistic.h"
002 #include "FileError.h"
003 #include <sstream>
004 #include <iomanip>
005 #include <chrono>
006 #include <ctime>
007 #include <fstream>
008
009 Statistic::Statistic(const std::string name, Test* test)
010     : name(name) {
011     this->test = new Test(*test);
012     setCurrentTime();
013     answers = {};
014     score = 0;
015     correctAnswersCount = 0;
016 }
017
018 void Statistic::setAnswers(const List<std::string>& answers)
019 {
020     this->answers = answers;
021 }
022
023 void Statistic::setCorrectAnswersCount(int count)
024 {
025     correctAnswersCount = count;
026     score = (int)std::round(((double)(correctAnswersCount) / test-
027 >getQuestionCount()) * 100.0);
028 }
029
030 void Statistic::setTime(const std::string time)
031 {
032     this->time = time;
033 }
034
035 void Statistic::setCurrentTime()
036 {
037     auto now = std::chrono::system_clock::now();
038     auto nowTimeT = std::chrono::system_clock::to_time_t(now);
039     std::tm localTime;
040     localtime_s(&localTime, &nowTimeT);
041     std::ostringstream oss;
```

```
042     oss << std::put_time(&localTime, "%d.%m.%y %H:%M");
043     this->time = oss.str();
044 }
045
046 std::string Statistic::getTestTitle()
047 {
048     return test->getTitle();
049 }
050 std::string Statistic::getUserName()
051 {
052     return name;
053 }
054 std::string Statistic::getTime()
055 {
056     return time;
057 }
058
059 void Statistic::print()
060 {
061     std::cout << "Имя тестируемого: " << name << "¥n";
062     std::cout << "Название теста: " << test->getTitle() << "¥n";
063     std::cout << "Дата и время прохождения: "<< time << "¥n";
064     std::cout << "==============================";
065     for (int i = 0; i < answers.getSize(); ++i) {
066         std::cout << "¥nВопрос " << (i + 1) << ": ";
067         test->getQuestion(i)->printQuestion();
068         if (answers[i] == SKIP) {
069             std::cout << "Вопрос был пропущен¥n";
070             test->getQuestion(i)->printAnswer();
071         }
072         else {
073             std::cout << "Ответ: " << answers[i];
074             if (test->getQuestion(i)->checkAnswer(answers[i])) {
075                 std::cout << " [V]¥n";
076             }
077             else {
078                 std::cout << " [X]¥n";
079                 test->getQuestion(i)->printAnswer();
080             }
081         }
082
083     }
084     std::cout << "==============================¥n";
085     std::cout << "Результат: " << correctAnswersCount << " из "
086         << test->getQuestionCount() << " (" << score << "%).¥n";
087     std::cout << "¥nВведите любой символ для возврата...";
088     std::string input;
089     getline(std::cin, input);
090     system("cls");
091 }
092
093 void Statistic::saveToFile()
094 {
095     std::ofstream out(STAT_FILE, std::ios::binary | std::ios::app);
096     if (!out.is_open()) {
097         std::string errorTest = "Не удалось открыть файл для записи: ";
098         throw FileError(errorTest + STAT_FILE);
099     }
100
101     test->saveToStream(out);
102
103     size_t nameSize = name.size();
```

```
104     out.write(reinterpret_cast<const char*>(&nameSize), sizeof(nameSize));
105     out.write(name.data(), nameSize);
106 
107     size_t timeSize = time.size();
108     out.write(reinterpret_cast<const char*>(&timeSize), sizeof(timeSize));
109     out.write(time.data(), timeSize);
110 
111     size_t answersCount = answers.getSize();
112     out.write(reinterpret_cast<const                        char*>(&answersCount),
    sizeof(answersCount));
113     for (int i = 0; i < answersCount; ++i) {
114         size_t answerSize = answers[i].size();
115         out.write(reinterpret_cast<const                      char*>(&answerSize),
    sizeof(answerSize));
116         out.write(answers[i].data(), answerSize);
117     }
118 
119     out.write(reinterpret_cast<const char*>(&correctAnswersCount),
120 sizeof(correctAnswersCount));
121 
122     out.close();
123 }
124 
125 List<Statistic*> Statistic::loadAllFromFile()
126 {
127     List<Statistic*> statistics = {};
128     std::ifstream in(STAT_FILE, std::ios::binary);
129     if (!in.is_open()) return statistics;
130 
131     while (in.peek() != EOF) {
132         try {
133             Statistic* stat = Statistic::loadFromStream(in);
134             statistics.pushBack(stat);
135         }
136         catch (const std::exception&) {
137             throw FileError("Ошибка: Не удалось прочитать файл статистики");
138         }
139     }
140 
141     in.close();
142     return statistics;
143 }
144 
145 
146 Statistic* Statistic::loadFromStream(std::ifstream& in)
147 {
148     Test* test = Test::loadFromStream(in);
149     Statistic* stat = new Statistic("", test);
150 
151     size_t nameSize;
152     in.read(reinterpret_cast<char*>(&nameSize), sizeof(nameSize));
153     stat->name.resize(nameSize);
154     in.read(&stat->name[0], nameSize);
155 
156     size_t timeSize;
157     in.read(reinterpret_cast<char*>(&timeSize), sizeof(timeSize));
158     stat->time.resize(timeSize);
159     in.read(&stat->time[0], timeSize);
160 
161     size_t answersCount;
162     in.read(reinterpret_cast<char*>(&answersCount), sizeof(answersCount));
163     stat->answers.clear();
```

```
164     for (size_t i = 0; i < answersCount; ++i) {
165         size_t answerSize;
166         in.read(reinterpret_cast<char*>(&answerSize), sizeof(answerSize));
167         std::string answer(answerSize, '¥0');
168         in.read(&answer[0], answerSize);
169         stat->answers.pushBack(answer);
170     }
171 
172     in.read(reinterpret_cast<char*>(&stat->correctAnswersCount), sizeof(stat-
    >correctAnswersCount));
173 
174     return stat;
175 }
```

StatisticMenu.h

```
001 #pragma once
002 #include "Statistic.h"
003 #include "InputError.h"
004 #include <iostream>
005 #include <string>
006 
007 class StatisticMenu {
008 public:
009     StatisticMenu();
010     ~StatisticMenu();
011     void start();
012 
013 private:
014     void displayStatisticList();
015     void deleteStatistic();
016     int printStatisticList();
017     void saveStatistics();
018     void sortStatistics();
019     List<Statistic*> statistics;
020 };
```

StatisticMenu.cpp

```
001 #include "StatisticMenu.h"
002 #include "Menu.h"
003 #include "FileError.h"
004 #include <fstream>
005 #include <iomanip>
006 #include <functional>
007 
008 StatisticMenu::StatisticMenu() {
009     statistics = {};
010     statistics = Statistic::loadAllFromFile();
011     sortStatistics();
012 }
013 
014 StatisticMenu::~StatisticMenu()
015 {
016     for (auto statistic : statistics) {
017         delete statistic;
018     }
019     statistics.clear();
020 }
021 
022 void StatisticMenu::start()
023 {
```

```
024     char choice;
025     do {
026         std::cout << "Меню статистики:¥n";
027         std::cout << "1. Просмотреть статистику¥n";
028         std::cout << "2. Удалить статистику¥n";
029         std::cout << "0. Вернуться в режим администратора¥n";
030         try {
031             choice = Menu::getInput("012");
032         }
033         catch (const InputError& e) {
034             cout << e.what() << endl;
035             continue;
036         }
037         switch (choice) {
038         case '1':
039             displayStatisticList();
040             break;
041         case '2':
042             deleteStatistic();
043             saveStatistics();
044             break;
045         }
046
047     } while (choice != '0');
048 }
049
050 void StatisticMenu::displayStatisticList()
051 {
052     while (true) {
053         int maxIndex = printStatisticList();
054         if (maxIndex == 0) {
055             std::cout << "Список статистики пуст.¥n¥nВведите любой символ для
    возврата...";
056             std::string input;
057             getline(std::cin, input);
058             system("cls");
059             return;
060         }
061
062         std::cout << "¥nВведите номер статистики для просмотра, либо 0 для
    возврата: ";
063         std::string input;
064         getline(std::cin, input);
065         system("cls");
066         if (input == "0") {
067             return;
068         }
069         try {
070             int selectedIndex = Menu::getValidatedNumber(input, 1, maxIndex);
071             statistics[selectedIndex - 1]->print();
072         }
073         catch (const InputError& e) {
074             std::cout << "Ошибка: " << e.what() << "¥n";
075         }
076     }
077 }
078
079 int StatisticMenu::printStatisticList()
080 {
081     if (statistics.getSize() == 0) {
082         return 0;
083     }
```

```
084
085     const int colWidthIndex = 4;
086     const int colWidthTestTitle = MAX_TEST_TITLE_LENGTH + 1;
087     const int colWidthUserName = MAX_FULL_NAME_LENGTH + 1;
088     const int colWidthTime = 15;
089
090     std::cout << "Список статистики:¥n";
091     std::cout << std::left << std::setw(colWidthIndex) << "№"
092         << std::left << std::setw(colWidthTestTitle) << "Название теста"
093         << std::left << std::setw(colWidthUserName) << "Имя тестируемого"
094         << std::left << std::setw(colWidthTime) << "Дата и время" << "¥n";
095
096     std::cout << std::string(colWidthIndex + colWidthTestTitle + colWidthUserName
    + colWidthTime, '-') << "¥n";
097
098     for (int i = 0; i < statistics.getSize(); ++i) {
099         const auto& stat = statistics[i];
100         std::cout << std::left << std::setw(colWidthIndex) << (i + 1)
101             << std::left << std::setw(colWidthTestTitle) << stat->getTestTitle()
102             << std::left << std::setw(colWidthUserName) << stat->getUserName()
103             << std::left << std::setw(colWidthTime) << stat->getTime() << "¥n";
104     }
105
106     return statistics.getSize();
107 }
108
109 void StatisticMenu::deleteStatistic()
110 {
111     while (true) {
112         int maxIndex = printStatisticList();
113         if (maxIndex == 0) {
114             std::cout << "Список статистики пуст.¥n¥nВведите любой символ для
    возврата...";
115             std::string input;
116             getline(std::cin, input);
117             system("cls");
118             return;
119         }
120
121         std::cout << "¥nВведите номер статистики для удаления (или 0 для отмены):
     ";
122         std::string input;
123         getline(std::cin, input);
124
125         if (input == "0") {
126             system("cls");
127             return;
128         }
129
130         try {
131             int selectedIndex = Menu::getValidatedNumber(input, 1, maxIndex);
132             auto stat = statistics[selectedIndex - 1];
133             std::cout << "Вы действительно хотите удалить статистику для теста
    ¥""
134                 << stat->getTestTitle()
135                 << "¥" от пользователя ¥""
136                 << stat->getUserName()
137                 << "¥"?¥n(Введите Y для подтверждения или любой другой символ для
138 отмены): ";
139
140             getline(std::cin, input);
141             system("cls");
```

```
142
143             if (input != "Y" && input != "y") {
144                 std::cout << "Удаление отменено.¥n";
145                 continue;
146             }
147
148             delete statistics[selectedIndex - 1];
149             statistics.removeAt(selectedIndex - 1);
150             std::cout << "Статистика успешно удалена!¥n";
151         }
152         catch (const InputError& e) {
153             system("cls");
154             std::cout << "Ошибка: " << e.what() << "¥n";
155         }
156     }
157 }
158
159 void StatisticMenu::saveStatistics()
160 {
161     do {
162         try {
163             std::ofstream out(STAT_FILE, std::ios::binary);
164             if (!out.is_open()) {
165                 std::string errorTest = "Не удалось открыть файл для записи: ";
166                 throw FileError(errorTest + STAT_FILE);
167             }
168             out.close();
169             for (const auto& stat : statistics) {
170                 stat->saveToFile();
171             }
172             break;
173         }
174         catch (const FileError& e) {
175             cout << "Ошибка: " << e.what() << endl;
176             cout << "Введите 0 для выхода без сохранения или любой другой символ
    для повторной попытки: ";
177             string input;
178             getline(cin, input);
179             system("cls");
180             if (input == "0") return;
181         }
182     } while (true);
183 }
184
185 void StatisticMenu::sortStatistics() {
186     int n = statistics.getSize();
187
188     std::function<void(int, int)> heapify = [&](int n, int i) {
189         int largest = i;
190         int left = 2 * i + 1;
191         int right = 2 * i + 2;
192
193         if (left < n && statistics[left]->getTestTitle() > statistics[largest]-
    >getTestTitle()) {
194             largest = left;
195         }
196
197         if (right < n && statistics[right]->getTestTitle() > statistics[largest]-
    >getTestTitle()) {
198             largest = right;
199         }
200
```

```
201         if (largest != i) {
202             std::swap(statistics[i], statistics[largest]);
203             heapify(n, largest);
204         }
205         };
206 
207     for (int i = n / 2 - 1; i >= 0; --i) {
208         heapify(n, i);
209     }
210 
211     for (int i = n - 1; i > 0; --i) {
212         std::swap(statistics[0], statistics[i]);
213         heapify(i, 0);
214     }
215 }
```

Test.h

```
001 #pragma once
002 #include "Question.h"
003 
004 #define PATH "tests/"
005 
006 class Test
007 {
008 public:
009    Test(const Test& other);
010    Test(const std::string& title);
011    ~Test();
012    Test& operator=(const Test& other);
013    const std::string getTitle() const;
014    int getQuestionCount() const;
015    void setFileName(const std::string& fileName);
016    void setTitle(const std::string& newTitle);
017    void addQuestion(Question* question);
018    void removeQuestion(int index);
019    Question* getQuestion(int index);
020    void saveToFile();
021    void deleteTestFile();
022    static Test* loadFromFile(const std::string& fileName);
023    static List<Test*> loadAllTests();
024 private:
025    std::string sanitizeFileName(const std::string& input) const;
026    std::string generateFileName() const;
027    std::string title;
028    List<Question*> questions;
029    std::string fileName = "";
030 };
```

Test.cpp

```
001 #include "Test.h"
002 #include "InputError.h"
003 #include "FileError.h"
004 #include <algorithm>
005 #include <string>
006 #include <cctype>
007 #include <fstream>
008 #include <sstream>
009 #include <iostream>
010 #include <unordered_map>
011 #include "dirent.h"
```

```
012 #include <direct.h>
013
014 Test::Test(const Test& other)
015 {
016     title = other.title;
017     questions = other.questions;
018     fileName = other.fileName;
019 }
020
021 Test::Test(const std::string& title)
022 {
023     setTitle(title);
024 }
025
026 Test::~Test()
027 {
028    for (Question* question : questions) {
029          delete question;
030    }
031     questions.clear();
032 }
033
034 Test& Test::operator=(const Test& other) {
035     if (this == &other) {
036         return *this;
037     }
038
039     title = other.title;
040     questions = other.questions;
041     fileName = other.fileName;
042
043     return *this;
044 }
045
046 const std::string Test::getTitle() const
047 {
048     return title;
049 }
050
051 int Test::getQuestionCount() const
052 {
053    return questions.getSize();
054 }
055
056 void Test::setTitle(const std::string& newTitle)
057 {
058     if (newTitle.size() > MAX_TEST_TITLE_LENGTH) {
059         throw InputError("Название превышает максимально допустимую длину (" +
060             std::to_string(MIN_TITLE_LENGTH) + " символов).");
061     }
062
063     if (newTitle.size() < MIN_TITLE_LENGTH) {
064         throw InputError("Название должно содерать минимум " +
065             std::to_string(MIN_TITLE_LENGTH) + " символов.");
066     }
067
068     if (newTitle.empty()) {
069         throw InputError("Название не может быть пустым.");
070     }
071
072     if (newTitle.front() == ' ' || newTitle.back() == ' ') {
073         throw InputError("Название не может начинаться или заканчиваться
```

```
    пробелом.");
074     }
075
076     if (!Question::isAlpha(newTitle.front())) {
077         throw InputError("Название должно начинаться с буквы");
078     }
079
080     for (size_t i = 0; i < newTitle.size(); ++i) {
081         unsigned char ch = newTitle[i];
082         if (!Question::isAlpha(ch) && !std::isdigit(ch) && !(ch == ' ' || ch ==
    '+' || ch == '-' || ch == '_')) {
083             throw InputError("Название может содержать только русские/английские
    буквы, пробелы и символы ¥"+-_¥"");
084         }
085
086         if (i > 0 && newTitle[i - 1] == ' ' && ch == ' ') {
087             throw InputError("Название не может содержать два и более пробела
    подряд.");
088         }
089
090     }
091
092     this->title = newTitle;
093 }
094
095 // Методы для управления вопросами
096 void Test::addQuestion(Question* question)
097 {
098     questions.pushBack(question);
099 }
100
101 void Test::removeQuestion(int index) {
102     auto question = getQuestion(index);
103     questions.removeAt(index);
104     delete question;
105 }
106
107 Question* Test::getQuestion(int index) {
108    return questions[(unsigned int)index];
109 }
110
111 std::string Test::generateFileName() const {
112     std::string baseName = sanitizeFileName(title);
113     std::string extension = ".ktst";
114     std::string fileName = baseName + extension;
115
116     int counter = 1;
117
118     while (true) {
119         bool fileExists = false;
120         DIR* dir = opendir(PATH);
121         if (!dir) {
122             throw FileError("Не удалось открыть текущий каталог для проверки.");
123         }
124
125         struct dirent* entry;
126         while ((entry = readdir(dir)) != nullptr) {
127             if (fileName == entry->d_name) {
128                 fileExists = true;
129                 break;
130             }
131         }
```

```
132         closedir(dir);
133 
134         if (!fileExists) {
135             break;
136         }
137 
138         std::ostringstream newFileName;
139         newFileName << baseName << counter << extension;
140         fileName = newFileName.str();
141         ++counter;
142     }
143 
144     return fileName;
145 }
146 
147 std::string Test::sanitizeFileName(const std::string& input) const
148 {
149     static const std::unordered_map<unsigned char, std::string> transliteration =
150 {
151         {'А', "a"}, {'а', "a"}, {'Б', "b"}, {'б', "b"}, {'В', "v"}, {'в', "v"},
152         {'Г', "g"}, {'г', "g"}, {'Д', "d"}, {'д', "d"}, {'Е', "e"}, {'е', "e"},
153         {'Ё', "e"}, {'ё', "e"}, {'Ж', "zh"}, {'ж', "zh"}, {'З', "z"}, {'з', "z"},
154         {'И', "i"}, {'и', "i"}, {'Й', "y"}, {'й', "y"}, {'К', "k"}, {'к', "k"},
155         {'Л', "l"}, {'л', "l"}, {'М', "m"}, {'м', "m"}, {'Н', "n"}, {'н', "n"},
156         {'О', "o"}, {'о', "o"}, {'П', "p"}, {'п', "p"}, {'Р', "r"}, {'р', "r"},
157         {'С', "s"}, {'с', "s"}, {'Т', "t"}, {'т', "t"}, {'У', "u"}, {'у', "u"},
158         {'Ф', "f"}, {'ф', "f"}, {'Х', "kh"}, {'х', "kh"}, {'Ц', "ts"}, {'ц',
    "ts"},
159         {'Ч', "ch"}, {'ч', "ch"}, {'Ш', "sh"}, {'ш', "sh"}, {'Щ', "shch"}, {'щ',
    "shch"},
160         {'Ъ', ""},  {'ъ', ""},  {'Ы', "y"}, {'ы', "y"}, {'Ь', ""},  {'ь', ""},
161         {'Э', "e"}, {'э', "e"}, {'Ю', "yu"}, {'ю', "yu"}, {'Я', "ya"}, {'я',
    "ya"}
162     };
163 
164     std::string fileName;
165     for (unsigned char ch : input)
166     {
167         auto it = transliteration.find(ch);
168         if (it != transliteration.end() ||  std::isalnum(ch) || ch == '-' || ch
    == '_')
169         {
170             if (it != transliteration.end())
171             {
172                 fileName += it->second;
173             }
174             else if (std::isalpha(ch))
175             {
176                 fileName += std::tolower(ch);
177             }
178             else
179             {
180                 fileName += ch;
181             }
182         }
183     }
184     return fileName;
185 }
186 
187 void Test::saveToFile()
188 {
189     std::string finalFileName = fileName.empty() ? generateFileName() : fileName;
```

```
190
191     std::ofstream outFile(PATH + finalFileName, std::ios::binary);
192     if (!outFile.is_open())
193     {
194         throw FileError("Невозможно открыть файл для записи: " + finalFileName);
195     }
196
197     size_t titleSize = title.size();
198     outFile.write(reinterpret_cast<const char*>(&titleSize), sizeof(titleSize));
199     outFile.write(title.data(), titleSize);
200
201     int questionCount = questions.getSize();
202     outFile.write(reinterpret_cast<const char*>(&questionCount),
    sizeof(questionCount));
203
204     for (auto question : questions)
205     {
206         question->saveToStream(outFile);
207     }
208
209     outFile.close();
210     fileName = finalFileName;
211 }
212
213 void Test::deleteTestFile() {
214     if (fileName.empty())
215     {
216         throw FileError("Невозможно удалить файл, так как он ещё не создан");
217     }
218     if (std::remove((PATH + fileName).c_str()) != 0) {
219         throw FileError("Не удалось удалить файл теста ¥"" + fileName + "¥".");
220     }
221     fileName.clear();
222 }
223
224 Test* Test::loadFromFile(const std::string& fileName)
225 {
226     std::ifstream inFile(PATH + fileName, std::ios::binary);
227     if (!inFile.is_open())
228     {
229         throw FileError("Невозможно открыть файл для чтения: " + fileName);
230     }
231
232     size_t titleSize;
233     inFile.read(reinterpret_cast<char*>(&titleSize), sizeof(titleSize));
234     std::string title(titleSize, '¥0');
235     inFile.read(&title[0], titleSize);
236     Test* test = new Test(title);
237     int questionCount;
238     inFile.read(reinterpret_cast<char*>(&questionCount), sizeof(questionCount));
239
240     for (int i = 0; i < questionCount; ++i)
241     {
242         Question* question = Question::loadFromStream(inFile);
243         test->addQuestion(question);
244     }
245
246     inFile.close();
247     test->setFileName(fileName);
248     return test;
249 }
250
```

```
List<Test*> Test::loadAllTests() {
    List<Test*> tests = {};
    const std::string extension = ".ktst";
    DIR* dir;
    struct dirent* entry;

    dir = opendir(PATH);
    if (!dir) {
        if (_mkdir(PATH) != 0) {
            throw FileError("Ошибка: Не удалось создать каталог для тестов");
        }
        return tests;
    }

    while ((entry = readdir(dir)) != nullptr) {
        std::string fileName = entry->d_name;
        if (fileName.size() > extension.size() &&
            fileName.substr(fileName.size() - extension.size()) == extension) {
            try {
                tests.pushBack(Test::loadFromFile(fileName));
            } catch (std::exception&) {}
        }
    }

    closedir(dir);

    return tests;
}

void Test::setFileName(const std::string& fileName) {
    this->fileName = fileName;
}
```

Question.h

```
#pragma once
#include "List.h"

#define MIN_TITLE_LENGTH 6
#define MAX_TEST_TITLE_LENGTH 32
#define MAX_QUESTION_TITLE_LENGTH 128
#define MAX_QUESTION_OPTION_LENGTH 64
#define MAX_QUESTION_ANSWER_LENGTH 20
#define MAX_ANSWER_LENGTH 32

enum class QuestionType
{
    SingleChoice = 1,
    MultipleChoice = 2,
    OpenAnswer = 3
};

class Question
{
public:
    Question(const Question& other);
    Question(const std::string& title);
    Question() {}
    Question& operator=(const Question& other);
    virtual ~Question() = default;
    virtual QuestionType getType() const = 0;
    virtual void printQuestion() = 0;
```

```
028     virtual bool checkAnswer(const std::string answer) const = 0;
029     virtual void saveToStream(std::ofstream& out) = 0;
030     std::string getTitle() const;
031     void setTitle(const std::string& title);
032     static Question* loadFromStream(std::ifstream& in);
033     static bool isAlpha(unsigned char ch);
034     static bool isSymbol(unsigned char ch);
035 protected:
036     std::string title = "";
037 };
```

Question.cpp

```
001 #include "Question.h"
002 #include "SingleChoiceQuestion.h"
003 #include "MultipleChoiceQuestion.h"
004 #include "OpenAnswerQuestion.h"
005 #include "InputError.h"
006 #include "FileError.h"
007 #include <string>
008 #include <memory>
009 #include <fstream>
010
011 Question::Question(const Question& other) { setTitle(other.title); }
012
013 Question::Question(const std::string& title) { setTitle(title); }
014
015 Question& Question::operator=(const Question& other)
016 {
017     if (this == &other) {
018         return *this;
019     }
020
021     setTitle(other.title);
022
023     return *this;
024 }
025
026 std::string Question::getTitle() const
027 {
028     return title;
029 }
030
031 bool Question::isAlpha(unsigned char ch)
032 {
033     return std::isalpha(ch) ||
034         (ch >= 0xC0 && ch <= 0xFF) ||
035         (ch == 0xB8 || ch == 0xA8);
036 }
037
038 bool Question::isSymbol(unsigned char ch) {
039     static const std::string validSymbols = "-+=?!;%№
    $()*&|#@:/.,><~_[]{}^'¥"¥¥";
040     return validSymbols.find(ch) != std::string::npos;
041 }
042
043 void Question::setTitle(const std::string& title)
044 {
045     if (title.size() > MAX_QUESTION_TITLE_LENGTH) {
046         throw InputError("Заголовок превышает максимально допустимую длину (" +
047             std::to_string(MAX_QUESTION_TITLE_LENGTH) + " символов).");
048     }
```

```
049     if (title.size() < MIN_TITLE_LENGTH) {
050         throw InputError("Заголовок должен содерать минимум " +
051             std::to_string(MIN_TITLE_LENGTH) + " символов.");
052     }
053 
054     if (title.empty()) {
055         throw InputError("Заголовок не может быть пустым.");
056     }
057 
058     if (title.front() == ' ' || title.back() == ' ') {
059         throw InputError("Заголовок может начинаться или заканчиваться
    пробелом.");
060     }
061 
062     if (!Question::isAlpha(title.front())) {
063         throw InputError("Заголовок должен начинаться с буквы");
064     }
065 
066     for (size_t i = 0; i < title.size(); ++i) {
067         unsigned char ch = title[i];
068         if (!Question::isAlpha(ch) && !std::isdigit(ch)
    && !Question::isSymbol(ch)) {
069             throw InputError("Заголовок может содержать только русские/английские
    буквы, пробелы и спец. символы.");
070         }
071 
072         if (i > 0 && title[i - 1] == ' ' && ch == ' ') {
073             throw InputError("Заголовок не может содержать два и более пробела
    подряд.");
074         }
075     }
076 
077     this->title = title;
078 }
079 
080 Question* Question::loadFromStream(std::ifstream& in)
081 {
082     int typeInt;
083     in.read(reinterpret_cast<char*>(&typeInt), sizeof(typeInt));
084     QuestionType type = static_cast<QuestionType>(typeInt);
085 
086     switch (type)
087     {
088     case QuestionType::SingleChoice:
089         return new
    SingleChoiceQuestion(SingleChoiceQuestion::loadFromStream(in));
090     case QuestionType::MultipleChoice:
091         return new
    MultipleChoiceQuestion(MultipleChoiceQuestion::loadFromStream(in));
092     case QuestionType::OpenAnswer:
093         return new OpenAnswerQuestion(OpenAnswerQuestion::loadFromStream(in));
094     default:
095         throw FileError("Неизвестный тип вопроса.");
096     }
097 }
```

SingleChoiceQuestion.h

```
001 #pragma once
002 #include "ChoiceQuestion.h"
003 
004 class SingleChoiceQuestion : public ChoiceQuestion
```

```
005 {
006 public:
007     SingleChoiceQuestion(const SingleChoiceQuestion& other);
008     SingleChoiceQuestion(const std::string& title, const List<std::string>&
    options, char correctIndex);
009     SingleChoiceQuestion() {}
010     SingleChoiceQuestion& operator=(const SingleChoiceQuestion& other);
011     QuestionType getType() const override;
012     bool checkAnswer(const std::string answer) const override;
013     void saveToStream(std::ofstream& out) override;
014     void printQuestion() override;
015     void setCorrectIndex(unsigned char correctIndex);
016     static SingleChoiceQuestion loadFromStream(std::ifstream& in);
017 private:
018     char correctIndex = 'A';
019 };
```

SingleChoiceQuestion.cpp

```
001 #include "SingleChoiceQuestion.h"
002 #include "InputError.h"
003 #include <iostream>
004 #include <string>
005 #include <cctype>
006 #include <fstream>
007
008 SingleChoiceQuestion::SingleChoiceQuestion(
009     const SingleChoiceQuestion& other) : ChoiceQuestion(other)
010 {
011     setCorrectIndex(other.correctIndex);
012 }
013
014 SingleChoiceQuestion::SingleChoiceQuestion(
015     const std::string& title,
016     const List<std::string>& options,
017     char correctIndex) : ChoiceQuestion(title, options)
018 {
019     setCorrectIndex(correctIndex);
020 }
021
022 SingleChoiceQuestion& SingleChoiceQuestion::operator=(const SingleChoiceQuestion&
    other) {
023     if (this == &other) {
024         return *this;
025     }
026
027     ChoiceQuestion::operator=(other);
028
029     setCorrectIndex(other.correctIndex);
030
031     return *this;
032 }
033
034 QuestionType SingleChoiceQuestion::getType() const
035 {
036     return QuestionType::SingleChoice;
037 }
038
039 void SingleChoiceQuestion::printQuestion()
040 {
041     std::cout << title << "¥n";
042     std::cout << "Варианты ответа:¥n";
```

```
043     for (int i = 0; i < options.getSize(); ++i) {
044         std::cout << static_cast<char>('A' + i) << ") " << options[i] << "¥n";
045     }
046 }
047
048 bool SingleChoiceQuestion::checkAnswer(const std::string answer) const
049 {
050     if (answer.size() != 1)
051     {
052         throw InputError("Введите один символ.");
053     }
054
055     char userAnswer = std::toupper(answer[0]);
056
057     if (!std::isalpha(userAnswer))
058     {
059         throw InputError("Ответом может быть только буква латинского алфавита (A-
    Z).");
060     }
061
062     if (userAnswer < 'A' || userAnswer >= 'A' + options.getSize())
063     {
064         throw InputError("Ответ выходит за пределы допустимых вариантов (A-" +
065             std::string(1, 'A' + options.getSize() - 1) + ").");
066     }
067
068     return userAnswer == correctIndex;
069 }
070
071 void SingleChoiceQuestion::setCorrectIndex(unsigned char correctIndex)
072 {
073     correctIndex = std::toupper(correctIndex);
074
075     if (!std::isalpha(correctIndex))
076     {
077         throw InputError("Правильный ответ должен быть буквой латинского
    алфавита.");
078     }
079
080     if (correctIndex < 'A' || correctIndex >= 'A' + options.getSize())
081     {
082         throw InputError("Правильный ответ выходит за пределы допустимых
    вариантов (A-" +
083             std::string(1, 'A' + options.getSize() - 1) + ").");
084     }
085
086     this->correctIndex = correctIndex;
087 }
088
089 void SingleChoiceQuestion::saveToStream(std::ofstream& out)
090 {
091     int type = static_cast<int>(getType());
092     out.write(reinterpret_cast<const char*>(&type), sizeof(type));
093
094     size_t titleSize = title.size();
095     out.write(reinterpret_cast<const char*>(&titleSize), sizeof(titleSize));
096     out.write(title.data(), titleSize);
097
098     size_t optionsSize = options.getSize();
099     out.write(reinterpret_cast<const char*>(&optionsSize), sizeof(optionsSize));
100
101     for (const auto& option : options)
```

```
102     {
103         size_t optSize = option.size();
104         out.write(reinterpret_cast<const char*>(&optSize), sizeof(optSize));
105         out.write(option.data(), optSize);
106     }
107
108     out.write(reinterpret_cast<const char*>(&correctIndex),
    sizeof(correctIndex));
109 }
110
111 SingleChoiceQuestion SingleChoiceQuestion::loadFromStream(std::ifstream& in)
112 {
113     size_t titleSize;
114     in.read(reinterpret_cast<char*>(&titleSize), sizeof(titleSize));
115     std::string title(titleSize, '¥0');
116     in.read(&title[0], titleSize);
117
118     size_t optionsSize;
119     in.read(reinterpret_cast<char*>(&optionsSize), sizeof(optionsSize));
120     List<std::string> options;
121     for (size_t i = 0; i < optionsSize; ++i)
122     {
123         size_t optSize;
124         in.read(reinterpret_cast<char*>(&optSize), sizeof(optSize));
125         std::string option(optSize, '¥0');
126         in.read(&option[0], optSize);
127         options.pushBack(option);
128     }
129
130     char correctIndex;
131     in.read(reinterpret_cast<char*>(&correctIndex), sizeof(correctIndex));
132
133     return SingleChoiceQuestion(title, options, correctIndex);
134 }
```

MultipleChoiceQuestion.h

```
001 #pragma once
002 #include "ChoiceQuestion.h"
003
004 class MultipleChoiceQuestion : public ChoiceQuestion
005 {
006 public:
007     MultipleChoiceQuestion(const MultipleChoiceQuestion& other);
008     MultipleChoiceQuestion(const std::string& title, const List<std::string>&
009 options, const List<unsigned char>& correctIndices);
010     MultipleChoiceQuestion() {}
011     MultipleChoiceQuestion& operator=(const MultipleChoiceQuestion& other);
012     QuestionType getType() const override;
013     bool checkAnswer(const std::string answer) const override;
014     void printQuestion() override;
015     void saveToStream(std::ofstream& out) override;
016     void setCorrectIndices(const List<unsigned char>& options);
017     static MultipleChoiceQuestion loadFromStream(std::ifstream& in);
018 private:
019     List<unsigned char> correctIndices;
020 };
021
```

MultipleChoiceQuestion.cpp

```
001 #include "MultipleChoiceQuestion.h"
```

```
002 #include "InputError.h"
003 #include <iostream>
004 #include <sstream>
005 #include <fstream>
006
007 MultipleChoiceQuestion::MultipleChoiceQuestion(
008     const MultipleChoiceQuestion& other) : ChoiceQuestion(other)
009 {
010     setCorrectIndices(other.correctIndices);
011 }
012
013 MultipleChoiceQuestion::MultipleChoiceQuestion(
014     const std::string& title,
015     const List<std::string>& options,
016     const List<unsigned char>& correctIndices) : ChoiceQuestion(title, options)
017 {
018     setCorrectIndices(correctIndices);
019 }
020
021 MultipleChoiceQuestion& MultipleChoiceQuestion::operator=(const
    MultipleChoiceQuestion& other)
022 {
023     if (this == &other) {
024         return *this;
025     }
026
027     ChoiceQuestion::operator=(other);
028
029     setCorrectIndices(other.correctIndices);
030
031     return *this;
032 }
033
034 QuestionType MultipleChoiceQuestion::getType() const
035 {
036     return QuestionType::MultipleChoice;
037 }
038
039 void MultipleChoiceQuestion::printQuestion()
040 {
041     std::cout << title << "¥n";
042     std::cout << "Варианты ответа:¥n";
043     for (int i = 0; i < options.getSize(); ++i) {
044         std::cout << static_cast<char>('A' + i) << "] " << options[i] << "¥n";
045     }
046 }
047
048 bool MultipleChoiceQuestion::checkAnswer(const std::string answer) const
049 {
050     List<char> userIndices;
051
052     std::istringstream iss(answer);
053     char index;
054
055     while (iss >> index)
056     {
057         if (!std::isalpha(index))
058         {
059             throw InputError("Ответами могут быть только буквы латинского
    алфавита (A-Z).");
060         }
061
```

```
062         char userChar = std::toupper(index);
063 
064         if (userChar < 'A' || userChar >= 'A' + options.getSize())
065         {
066             throw InputError("Правильный ответ ¥"" + std::string(1, userChar) +
067                 "¥" выходит за пределы допустимых вариантов (A-" +
068                 std::string(1, 'A' + options.getSize() - 1) + ").");
069         }
070 
071         bool inserted = false;
072         for (int i = 0; i < userIndices.getSize(); ++i) {
073             if (userChar == userIndices[i]) {
074                 throw InputError("Правильный ответ ¥"" + std::string(1, userChar)
    + "¥" указан более одного раза.");
075             }
076             if (userIndices[i] > userChar) {
077                 userIndices.insert(userChar, i);
078                 inserted = true;
079                 break;
080             }
081         }
082         if (!inserted) {
083             userIndices.pushBack(userChar);
084         }
085     }
086 
087     if (userIndices.getSize() != correctIndices.getSize()) return false;
088 
089     for (auto it = correctIndices.begin(); it != correctIndices.end(); ++it)
090     {
091         if (!userIndices.contains(*it)) return false;
092     }
093 
094     return true;
095 }
096 
097 
098 void MultipleChoiceQuestion::setCorrectIndices(const List<unsigned char>&
    correctIndices)
099 {
100     if (correctIndices.isEmpty())
101     {
102         throw InputError("Необходимо указать хотя бы один правильный ответ.");
103     }
104 
105     List<unsigned char> indices = {};
106 
107     for (unsigned char index : correctIndices)
108     {
109         unsigned char upperIndex = std::toupper(index);
110 
111         if (!std::isalpha(upperIndex))
112         {
113             throw InputError("Каждый правильный ответ должен быть буквой
    латинского алфавита.");
114         }
115 
116         if (upperIndex < 'A' || upperIndex >= 'A' + options.getSize())
117         {
118             throw InputError("Правильный ответ ¥"" + std::string(1, upperIndex) +
119                 "¥" выходит за пределы допустимых вариантов (A-" +
120                 std::string(1, 'A' + options.getSize() - 1) + ").");
```

```
121         }
122 
123         bool inserted = false;
124         for (int i = 0; i < indices.getSize(); ++i) {
125             if (upperIndex == indices[i]) {
126                 throw InputError("Правильный ответ ¥"" + std::string(1,
    upperIndex) +
127                     "¥" указан более одного раза.");
128             }
129             if (indices[i] > upperIndex) {
130                 indices.insert(upperIndex, i);
131                 inserted = true;
132                 break;
133             }
134         }
135 
136         if (!inserted) {
137             indices.pushBack(upperIndex);
138         }
139     }
140 
141     this->correctIndices = indices;
142 }
143 
144 
145 void MultipleChoiceQuestion::saveToStream(std::ofstream& out)
146 {
147     int type = static_cast<int>(getType());
148     out.write(reinterpret_cast<const char*>(&type), sizeof(type));
149 
150     size_t titleSize = title.size();
151     out.write(reinterpret_cast<const char*>(&titleSize), sizeof(titleSize));
152     out.write(title.data(), titleSize);
153 
154     size_t optionsSize = options.getSize();
155     out.write(reinterpret_cast<const char*>(&optionsSize), sizeof(optionsSize));
156     for (const auto& option : options)
157     {
158         size_t optSize = option.size();
159         out.write(reinterpret_cast<const char*>(&optSize), sizeof(optSize));
160         out.write(option.data(), optSize);
161     }
162 
163     size_t correctSize = correctIndices.getSize();
164     out.write(reinterpret_cast<const char*>(&correctSize), sizeof(correctSize));
165     for (const char& index : correctIndices)
166     {
167         out.write(reinterpret_cast<const char*>(&index), sizeof(index));
168     }
169 }
170 
171 MultipleChoiceQuestion MultipleChoiceQuestion::loadFromStream(std::ifstream& in)
172 {
173     size_t titleSize;
174     in.read(reinterpret_cast<char*>(&titleSize), sizeof(titleSize));
175     std::string title(titleSize, '¥0');
176     in.read(&title[0], titleSize);
177     size_t optionsSize;
178     in.read(reinterpret_cast<char*>(&optionsSize), sizeof(optionsSize));
179     List<std::string> options;
180     for (size_t i = 0; i < optionsSize; ++i)
181     {
```

```
182         size_t optSize;
183         in.read(reinterpret_cast<char*>(&optSize), sizeof(optSize));
184         std::string option(optSize, '¥0');
185         in.read(&option[0], optSize);
186         options.pushBack(option);
187     }
188     size_t correctSize;
189     in.read(reinterpret_cast<char*>(&correctSize), sizeof(correctSize));
190     List<unsigned char> correctIndices;
191     for (size_t i = 0; i < correctSize; ++i)
192     {
193         char index;
194         in.read(reinterpret_cast<char*>(&index), sizeof(index));
195         correctIndices.pushBack(index);
196     }
197     return MultipleChoiceQuestion(title, options, correctIndices);
198 }
```

OpenAnswerQuestion.h

```
001 #pragma once
002 #include "Question.h"
003 #define ALPHA_OFFSET 32
004 class OpenAnswerQuestion : public Question
005 {
006 public:
007     OpenAnswerQuestion(const OpenAnswerQuestion& other);
008     OpenAnswerQuestion(const std::string& title, const std::string&
    correctAnswer);
009     OpenAnswerQuestion() {}
010     OpenAnswerQuestion& operator=(const OpenAnswerQuestion& other);
011     QuestionType getType() const override;
012     void printQuestion() override;
013     bool checkAnswer(const std::string answer) const override;
014     void setAnswer(const std::string answer);
015     void saveToStream(std::ofstream& out) override;
016     static OpenAnswerQuestion loadFromStream(std::ifstream& in);
017 private:
018     unsigned char toLower(unsigned char) const;
019     int levenshteinDistance(const std::string& str1, const std::string& str2)
020 const;
021     std::string correctAnswer;
022 };
023
```

OpenAnswerQuestion.cpp

```
001 #include "OpenAnswerQuestion.h"
002 #include "InputError.h"
003 #include <iostream>
004 #include <cctype>
005 #include <string>
006 #include <locale>
007 #include <fstream>
008
009 OpenAnswerQuestion::OpenAnswerQuestion(const OpenAnswerQuestion& other) :
010 Question(other)
011 {
012     setAnswer(other.correctAnswer);
013 }
014
015 OpenAnswerQuestion::OpenAnswerQuestion(
```

```
016     const std::string& title,
017     const std::string& correctAnswer) : Question(title)
018 {
019     setAnswer(correctAnswer);
020 }
021
022 OpenAnswerQuestion& OpenAnswerQuestion::operator=(const OpenAnswerQuestion&
    other)
023 {
024     if (this == &other) {
025         return *this;
026     }
027
028     Question::operator=(other);
029
030     setAnswer(other.correctAnswer);
031
032     return *this;
033 }
034
035 QuestionType OpenAnswerQuestion::getType() const
036 {
037     return QuestionType::OpenAnswer;
038 }
039
040 bool OpenAnswerQuestion::checkAnswer(const std::string answer) const
041 {
042     if (answer.empty())
043     {
044         throw InputError("Ответ не может быть пустым.");
045     }
046
047     if (answer.size() > MAX_ANSWER_LENGTH)
048     {
049         throw InputError("Ответ превышает максимально допустимую длину (" +
050             std::to_string(MAX_ANSWER_LENGTH) + " символов).");
051     }
052
053     for (char ch : answer)
054     {
055         if (!Question::isAlpha(ch) && !std::isdigit(ch)) {
056             throw InputError("Ответ может содержать только русские/английские
    буквы и цифры");
057         }
058     }
059
060     std::string lowerAnswer;
061
062     for (unsigned char ch : answer) {
063         lowerAnswer += toLower(ch);
064     }
065
066     int distance = levenshteinDistance(lowerAnswer, this->correctAnswer);
067     int threshold = static_cast<int>(std::round(correctAnswer.size() * 0.2));
068     return distance <= threshold;
069 }
070
071
072 int OpenAnswerQuestion::levenshteinDistance(const std::string& str1, const
    std::string& str2) const
073 {
074     size_t len1 = str1.size();
```

```
075     size_t len2 = str2.size();
076 
077     List<int> prev;
078     List<int> curr;
079 
080     for (unsigned int j = 0; j <= len2; ++j)
081     {
082         prev.pushBack(j);
083     }
084 
085     for (unsigned int i = 1; i <= len1; ++i)
086     {
087         curr.clear();
088         curr.pushBack(i);
089 
090         auto str2It = str2.begin();
091         for (unsigned int j = 1; j <= len2; ++j, ++str2It)
092         {
093             int cost = (str1[i - 1] == str2[j - 1]) ? 0 : 1;
094             curr.pushBack(std::min({prev[j] + 1,
095                                     curr[j - 1] + 1,
096                                     prev[j - 1] + cost }));
097         }
098 
099         prev = curr;
100     }
101 
102     return prev.back();
103 }
104 
105 
106 void OpenAnswerQuestion::printQuestion()
107 {
108     std::cout << title << "¥n";
109 }
110 
111 void OpenAnswerQuestion::setAnswer(const std::string answer)
112 {
113     if (answer.empty())
114     {
115         throw InputError("Ответ не может быть пустым.");
116     }
117 
118     if (answer.size() > MAX_ANSWER_LENGTH)
119     {
120         throw InputError("Ответ превышает максимально допустимую длину (" +
121         std::to_string(MAX_ANSWER_LENGTH) + " символов).");
122     }
123 
124     for (char ch : answer)
125     {
126         if (!Question::isAlpha(ch) && !std::isdigit(ch)) {
127             throw InputError("Ответ может содержать только русские/английские
    буквы и цифры");
128         }
129     }
130 
131     this->correctAnswer = answer;
132 }
133 
134 void OpenAnswerQuestion::saveToStream(std::ofstream& out)
135 {
```

```
136     int type = static_cast<int>(getType());
137     out.write(reinterpret_cast<const char*>(&type), sizeof(type));
138 
139     size_t titleSize = title.size();
140     out.write(reinterpret_cast<const char*>(&titleSize), sizeof(titleSize));
141     out.write(title.data(), titleSize);
142 
143     size_t answerSize = correctAnswer.size();
144     out.write(reinterpret_cast<const char*>(&answerSize), sizeof(answerSize));
145     out.write(correctAnswer.data(), answerSize);
146 }
147 
148 OpenAnswerQuestion OpenAnswerQuestion::loadFromStream(std::ifstream& in)
149 {
150     size_t titleSize;
151     in.read(reinterpret_cast<char*>(&titleSize), sizeof(titleSize));
152     std::string title(titleSize, '¥0');
153     in.read(&title[0], titleSize);
154 
155     size_t answerSize;
156     in.read(reinterpret_cast<char*>(&answerSize), sizeof(answerSize));
157     std::string correctAnswer(answerSize, '¥0');
158     in.read(&correctAnswer[0], answerSize);
159     return OpenAnswerQuestion(title, correctAnswer);
160 }
161 
162 unsigned char OpenAnswerQuestion::toLower(unsigned char ch) const{
163     if (ch >= 'A' && ch <= 'Я') {
164         return ch + ALPHA_OFFSET;
165     }
166     else if (ch == 'Ё') {
167         return 'ё';
168     }
169     return static_cast<char>(std::tolower(static_cast<unsigned char>(ch)));
170 }
```

ChoiceQuestion.h

```
001 #pragma once
002 #include "Question.h"
003 
004 class ChoiceQuestion : public Question
005 {
006 public:
007     ChoiceQuestion(const ChoiceQuestion& other);
008     ChoiceQuestion(const std::string& title, const List<std::string>& options);
009     ChoiceQuestion(): Question() {}
010     ChoiceQuestion& operator=(const ChoiceQuestion& other);
011     virtual ~ChoiceQuestion() = default;
012     List<std::string> getOptions() const;
013     std::string getOption(unsigned int index);
014     int getOptionCount() const;
015     void addOption(const std::string& option, int index = -1);
016     void removeOption(unsigned int index);
017 protected:
018     List<std::string> options = {};
019 };
```

ChoiceQuestion.cpp

```
001 #include "ChoiceQuestion.h"
002 #include "InputError.h"
```

```
003 #include <string>
004 #include <cctype>
005
006 ChoiceQuestion::ChoiceQuestion(const ChoiceQuestion& other) : Question(other)
007 {
008     for (auto option : other.options) {
009         addOption(option);
010     }
011 }
012
013 ChoiceQuestion::ChoiceQuestion(const std::string& title, const List<std::string>&
    options) : Question(title)
014 {
015     for (auto option : options) {
016         addOption(option);
017     }
018 }
019
020 ChoiceQuestion& ChoiceQuestion::operator=(const ChoiceQuestion& other)
021 {
022     if (this == &other) {
023         return *this;
024     }
025
026     Question::operator=(other);
027
028     options.clear();
029     for (auto option : other.options) {
030         addOption(option);
031     }
032
033     return *this;
034 }
035
036 void ChoiceQuestion::addOption(const std::string& option, int index)
037 {
038     if (option.empty())
039     {
040         throw InputError("Вариант ответа не может быть пустым.");
041     }
042
043     if (option.size() > MAX_QUESTION_OPTION_LENGTH)
044     {
045         throw InputError("Вариант ответа превышает максимально допустимую длину
    (" + std::to_string(MAX_QUESTION_OPTION_LENGTH) + " символов).");
046     }
047
048     if (option.empty()) {
049         throw InputError("Вариант ответа не может быть пустым.");
050     }
051
052     if (option.front() == ' ' || option.back() == ' ') {
053         throw InputError("Вариант ответа не может начинаться или заканчиваться
    пробелом.");
054     }
055
056     for (size_t i = 0; i < option.size(); ++i) {
057         unsigned char ch = option[i];
058         if (!Question::isAlpha(ch) && !std::isdigit(ch)
059 && !Question::isSymbol(ch)) { throw InputError("Вариант ответа может содержать
    только русские/английские буквы, пробелы и спец. символы.");
060         }
```

```
061
062         if (i > 0 && option[i - 1] == ' ' && ch == ' ') {
063             throw InputError("Вариант ответа не может содержать два и более
    пробела подряд.");
064         }
065     }
066     if (index == -1) {
067         options.pushBack(option);
068     }
069     else {
070         if (index > options.getSize()) {
071             throw InputError("Индекс выходит за пределы доступных вариантов.");
072         }
073         options.insert(option, index);
074     }
075
076 }
077
078 void ChoiceQuestion::removeOption(unsigned int index)
079 {
080     if ((int)index >= options.getSize())
081     {
082         throw InputError("Индекс выходит за пределы диапазона.");
083     }
084
085     options.removeAt(index);
086 }
087
088 int ChoiceQuestion::getOptionCount() const {
089     return options.getSize();
090 }
091
092 List<std::string> ChoiceQuestion::getOptions() const
093 {
094     auto newOptions = options;
095     return newOptions;
096 }
097
098 std::string ChoiceQuestion::getOption(unsigned int index) {
099     return options[index];
100 }
```

List.h

```
001 #pragma once
002 #include <iostream>
003
004 enum RemoveMode
005 {
006    All,
007    First
008 };
009
010 template<typename T>
011 class List
012 {
013 private:
014
015    template<typename T>
016    struct Node
017    {
018           Node(T data = T(), Node<T> *ptrPrevious = nullptr, Node<T> *ptrNext =
```

```
019  nullptr)
020         {
021                this->data = data;
022                this->ptrPrevious = ptrPrevious;
023                this->ptrNext = ptrNext;
024         }
025
026         T data;
027         Node<T> *ptrPrevious;
028         Node<T> *ptrNext;
029   };
030
031   Node<T> *findElement(const unsigned int index)
032   {
033         if (size <= (int)index)
034                return nullptr;
035
036         Node<T>* findedNode;
037
038         if (size / 2 >= (int)index)
039         {
040                findedNode = head;
041                for (unsigned int i = 0; i < index; ++i)
042                       findedNode = findedNode->ptrNext;
043         }
044         else
045         {
046                findedNode = tail;
047                for (unsigned int i = size - 1; i > index; --i)
048                       findedNode = findedNode->ptrPrevious;
049         }
050
051         return findedNode;
052   }
053
054   void setDefaultValues();
055
056   int size;
057   Node<T> *head;
058   Node<T> *tail;
059
060 public:
061
062   List(const T*, const int);
063   List();
064   List(std::initializer_list<T>);
065   ~List();
066   T &operator[](const unsigned int);
067
068   List<T> &operator=(const List<T>& other);
069   T &front() const;
070   T &back() const;
071   void pushFront(const T);
072   void pushBack(const T);
073   void popFront();
074   void popBack();
075   void insert(const T &, const unsigned int);
076   void removeAt(const unsigned int);
077   void remove(const T &, RemoveMode);
078   void clear();
079   bool isEmpty() const;
080   int getSize() const;
```

```
081
082   int firstIndexOf(const T &) const;
083   bool contains(const T &) const;
084
085   template<typename T>
086   class Iterator {
087   public:
088         Iterator() : node(nullptr) {}
089         Iterator(Node<T>* node) : node(node) {}
090         Iterator(const Iterator &other) : node(other.node) {}
091
092         Iterator& operator=(const Iterator &other) {
093               node = other.node;
094               return *this;
095         }
096
097         T &operator*() const { return node->data; }
098         T* operator->() { return &(node->data); }
099         bool operator==(const Iterator &other) const { return node ==
    other.node; }
100         bool operator!=(const Iterator &other) const { return node !=
    other.node; }
101
102         Iterator &operator++()
103         {
104               node = node->ptrNext;
105               return *this;
106         }
107
108         Iterator operator++(int)
109         {
110               Iterator temp(*this);
111               node = node->ptrNext;
112               return temp;
113         }
114
115         Iterator &operator--()
116         {
117               node = node->ptrPrevious;
118               return *this;
119         }
120
121         Iterator operator--(int)
122         {
123               Iterator temp(*this);
124               node = node->ptrPrevious;
125               return temp;
126         }
127
128         Iterator operator+(int n) const {
129               Node<T>* newNode = node;
130               for (int i = 0; i < n; ++i)
131               {
132                     if (newNode != nullptr)
133                           newNode = newNode->ptrNext;
134               }
135               return Iterator(newNode);
136         }
137
138
139         Iterator operator-(int n) const {
140               Node<T>* newNode = node;
```

```
141                 for (int i = 0; i < n; ++i)
142                 {
143                       if (newNode != nullptr)
144                             newNode = newNode->ptrPrevious;
145                 }
146                 return Iterator(newNode);
147           }
148
149     private:
150           Node<T>* node;
151     };
152
153     Iterator<T> begin() const { return Iterator<T>(head); }
154     Iterator<T> end() const { return Iterator<T>(nullptr); }
155     Iterator<T> rbegin() const { return Iterator<T>(tail); }
156     Iterator<T> rend() const { return Iterator<T>(nullptr); }
157     Iterator<T> find(const T& data) const {
158           for (Iterator<T> it = begin(); it != end(); ++it) {
159                 if (*it == data)
160                       return it;
161           }
162           return end();
163     }
164
165     Iterator<T> findAt(const unsigned int index) const {
166
167           if ((unsigned)size <= index)
168                 throw std::out_of_range("Ошибка(Iterator<T> findAt(const unsigned
    int): Выход за границы списка");
169
170           Node<T>* findedNode;
171
172           if ((unsigned)size / 2 >= index)
173           {
174                 findedNode = head;
175                 for (unsigned int i = 0; i < index; ++i)
176                       findedNode = findedNode->ptrNext;
177           }
178           else
179           {
180                 findedNode = tail;
181                 for (unsigned int i = size - 1; i > index; --i)
182                       findedNode = findedNode->ptrPrevious;
183           }
184
185           return Iterator<T>(findedNode);
186     }
187 };
188
189 template<typename T>
190 List<T>::List(const T* data, const int size)
191 {
192     for (int i = 0; i < size; ++i)
193     {
194           pushBack(*(data + i));
195     }
196 }
197
198 template<typename T>
199 List<T>::List()
200 {
201     setDefaultValues();
```

```
202 }
203
204 template<typename T>
205 List<T>& List<T>::operator=(const List<T>& other) {
206    if (this == &other)
207           return *this;
208
209    clear();
210
211    Node<T>* currentNode = other.head;
212    while (currentNode != nullptr) {
213           pushBack(currentNode->data);
214           currentNode = currentNode->ptrNext;
215    }
216
217    return *this;
218 }
219
220 template<typename T>
221 List<T>::List(std::initializer_list<T> data)
222 {
223    setDefaultValues();
224    for (auto item : data)
225           pushBack(item);
226 }
227
228 template<typename T>
229 List<T>::~List()
230 {
231    clear();
232 }
233
234 template<typename T>
235 T& List<T>::front() const
236 {
237    if (isEmpty())
238           throw std::exception("Ошибка(T & List<T>::front()): Выход за границы
    списка");
239    return head->data;
240 }
241
242 template<typename T>
243 T& List<T>::back() const
244 {
245    if (isEmpty())
246           throw std::out_of_range("Ошибка(T & List<T>::back()): Выход за границы
247 списка");
248    return tail->data;
249 }
250
251 template<typename T>
252 T& List<T>::operator[](const unsigned int index)
253 {
254    if (size <= (int)index)
255           throw std::out_of_range("Ошибка(T & List<T>::operator[](const int
    index)): Выход за границы списка");
256
257    Node<T>* currentNode = findElement(index);
258
259    return currentNode->data;
260 }
261
```

```
262 template<typename T>
263 void List<T>::setDefaultValues()
264 {
265    size = 0;
266    head = nullptr;
267    tail = nullptr;
268 }
269
270 template<typename T>
271 void List<T>::pushFront(const T element)
272 {
273    if (size == 0)
274    {
275           head = new Node<T>(element, nullptr, nullptr);
276           tail = head;
277    }
278    else
279    {
280           Node<T>* newNode = new Node<T>(element, nullptr, head);
281           head->ptrPrevious = newNode;
282           head = newNode;
283    }
284    ++size;
285 }
286
287 template<typename T>
288 void List<T>::pushBack(const T element)
289 {
290    if (size <= 1)
291    {
292           if (size == 0)
293           {
294                  head = new Node<T>(element, nullptr, nullptr);
295                  tail = head;
296           }
297           else
298           {
299                  tail = new Node<T>(element, head, nullptr);
300                  head->ptrNext = tail;
301           }
302    }
303    else
304    {
305           Node<T>* newNode = new Node<T>(element, tail, nullptr);
306           tail->ptrNext = newNode;
307           tail = newNode;
308    }
309    ++size;
310 }
311
312 template<typename T>
313 void List<T>::popFront()
314 {
315    if (size <= 1)
316    {
317           if (size == 0)
318                  throw std::exception("Ошибка: Список уже пуст!");
319           if (size == 1)
320           {
321                  delete head;
322                  setDefaultValues();
323           }
```

```
324         return;
325   }
326   Node<T>* nextNode = head->ptrNext;
327   delete head;
328   nextNode->ptrPrevious = nullptr;
329   head = nextNode;
330   --size;
331 }
332
333 template<typename T>
334 void List<T>::popBack()
335 {
336   if (size <= 1)
337   {
338         if (size == 0)
339               throw std::exception("Ошибка: Список уже пуст!");
340         if (size == 1)
341         {
342               delete tail;
343               setDefaultValues();
344         }
345         return;
346   }
347   Node<T>* previousNode = tail->ptrPrevious;
348   delete tail;
349   previousNode->ptrNext = nullptr;
350   tail = previousNode;
351   --size;
352 }
353
354 template<typename T>
355 void List<T>::insert(const T& element, const unsigned int index)
356 {
357   if (index == 0)
358         pushFront(element);
359   else if (size == (int)index)
360         pushBack(element);
361   else if (size < (int)index)
362         throw std::out_of_range("Ошибка(void List<T>::insert(const T &element,
    const int index)): Выход за границы списка");
363   else
364   {
365         Node<T>* nextNode = findElement(index);
366         Node<T>* newNode = new Node<T>(element, nextNode->ptrPrevious,
    nextNode);
367         nextNode->ptrPrevious->ptrNext = newNode;
368         nextNode->ptrPrevious = newNode;
369         ++size;
370   }
371 }
372
373 template<typename T>
374 void List<T>::removeAt(const unsigned int index)
375 {
376   if (size <= (int)index)
377         throw std::exception("Ошибка(void List<T>::removeAt(const int index)):
    Выход за границы списка");
378   else if (index == 0)
379         popFront();
380   else if ((int)index == size - 1)
381         popBack();
382   else
```

```
383   {
384         Node<T>* currentNode = findElement(index);
385         if (currentNode->ptrNext != nullptr)
386               currentNode->ptrNext->ptrPrevious = currentNode->ptrPrevious;
387         if (currentNode->ptrPrevious != nullptr)
388               currentNode->ptrPrevious->ptrNext = currentNode->ptrNext;
389         delete currentNode;
390         --size;
391   }
392 }
393
394 template<typename T>
395 void List<T>::remove(const T& element, RemoveMode mode)
396 {
397   Node<T>* currentNode = head;
398   while (currentNode != nullptr)
399   {
400         if (currentNode->data == element)
401         {
402               Node<T>* tempNode = currentNode->ptrNext;
403
404               if (currentNode->ptrNext != nullptr)
405                     currentNode->ptrNext->ptrPrevious = currentNode-
    >ptrPrevious;
406               else
407                     tail = currentNode->ptrPrevious;
408               if (currentNode->ptrPrevious != nullptr)
409                     currentNode->ptrPrevious->ptrNext = currentNode->ptrNext;
410               else
411                     head = currentNode->ptrNext;
412
413               delete currentNode;
414               currentNode = tempNode;
415               --size;
416               if (mode == All)
417                     continue;
418               else
419                     break;
420         }
421         currentNode = currentNode->ptrNext;
422   }
423 }
424
425 template<typename T>
426 int List<T>::firstIndexOf(const T& element) const
427 {
428   Node<T>* currentNode = head;
429   for (int i = 0; i < size; ++i)
430   {
431         if (currentNode->data == element)
432               return i;
433         currentNode = currentNode->ptrNext;
434   }
435   return -1;
436 }
437
438 template<typename T>
439 bool List<T>::contains(const T& element) const
440 {
441   return firstIndexOf(element) != -1;
442 }
443
```

```
444 template<typename T>
445 void List<T>::clear()
446 {
447    while (size != 0)
448           popFront();
449 }
450
451 template<typename T>
452 bool List<T>::isEmpty() const
453 {
454    return size == 0;
455 }
456
457 template<typename T>
458 int List<T>::getSize() const
459 {
460    return size;
461 }
```

KeyVerifier.h

```
001 #pragma once
002 #include <string>
003 #include <cstdint>
004 #define KEY_HASH 11194
005 #define KEY_SIZE 128
006 #define KEY_FILENAME "kts_key.dat"
007
008 class KeyVerifier {
009 public:
010     static bool verify();
011 private:
012     static bool isRemovableDrive(const std::string& drivePath);
013     static uint32_t getHash(const std::string& data);
014 };
```

KeyVerifier.cpp

```
001 #include "KeyVerifier.h"
002 #include "FileError.h"
003 #include "windows.h"
004 #include <fstream>
005 #include <iostream>
006
007
008 uint32_t KeyVerifier::getHash(const std::string& data) {
009     uint32_t hash = 0;
010     for (unsigned char byte : data) {
011         hash += byte;
012     }
013     return hash;
014 }
015
016 bool KeyVerifier::isRemovableDrive(const std::string& drivePath) {
017     UINT driveType = GetDriveTypeA(drivePath.c_str());
018     return driveType == DRIVE_REMOVABLE;
019 }
020
021 bool KeyVerifier::verify() {
022     DWORD drives = GetLogicalDrives();
023     if (drives == 0) {
024         return false;
```

```
025     }
026
027     for (char drive = 'A'; drive <= 'Z'; ++drive) {
028         if (drives & (1 << (drive - 'A'))) {
029             std::string drivePath = std::string(1, drive) + ":¥¥";
030             if (!isRemovableDrive(drivePath)) {
031                 continue;
032             }
033
034             std::ifstream file(drivePath + KEY_FILENAME, std::ios::binary);
035             if (!file.is_open()) {
036                 continue;
037             }
038
039             std::string fileContent((std::istreambuf_iterator<char>(file)),
040                 std::istreambuf_iterator<char>());
041             file.close();
042             // std::cout << getHash(fileContent) << std::endl;
043             // std::cout << fileContent.size() << std::endl;
044             if (fileContent.size() == KEY_SIZE && getHash(fileContent) ==
045  KEY_HASH)
046  {
047                 return true;
048             }
049         }
050     }
051
052     return false;
053 }
```

InputError.h

```
001 #pragma once
002 #include <stdexcept>
003 class InputError : public std::exception {
004 public:
005     explicit InputError(const std::string& message) : message_(message) {}
006
007     const char* what() const noexcept override {
008         return message_.c_str();
009     }
010
011 private:
012     std::string message_;
013 };
```

FileError.h

```
001 #pragma once
002 #include <stdexcept>
003 class FileError : public std::exception {
004 public:
005     explicit FileError(const std::string& message) : message_(message) {}
006
007     const char* what() const noexcept override {
008         return message_.c_str();
009     }
010
011 private:
012     std::string message_;
013 };
```