

**Содержание**

[Введение 6](#_Toc514513491)

[1 Постановка задачи 7](#_Toc514513492)

[2 Обоснование выбора метода решения поставленной задачи 8](#_Toc514513493)

[3 Обоснование выбора структуры данных 11](#_Toc514513494)

[4 Описание алгоритмов решения задачи 12](#_Toc514513495)

[5 Особенности программной реализации 18](#_Toc514513496)

[6 Описание пользовательского интерфейса 21](#_Toc514513497)

[Заключение 26](#_Toc514513498)

[Список использованных источников 27](#_Toc514513499)

[Приложение А (обязательное) Листинги программы 28](#_Toc514513500)

# **Введение**

Тесты — стандартизованные задачи, выполнение которых позволяет измерить некоторые психологические, интеллектуальные характеристики, уровень знаний. Тематические тесты (тесты на заданную тему) позволяют относительно легко, быстро и объективно оценить качество знаний тестируемого в данной области науки, искусства, развлечений и т.д. Они также дают возможность за достаточно короткий промежуток времени оценить показатель успеваемости в данной сфере или осведомленности по данной проблеме большого количества людей.

В повседневной жизни из-за огромного количества информации, окружающей нас, человек может задуматься о том, насколько хорошо он разбирается в той или иной тематике. Для решения данной проблемы, ему просто необходимо будет найти нужный этому человеку тест на интересующую его тематику, пройти его, получить результаты и, проанализировав их, сделать для себя выводы о глубине познаний в данной области.

Целью данной курсовой работы является разработка программы «Тестирование» на различные тематики.

Задачами данной курсовой работы, которые необходимо выполнить для достижения поставленной цели, являются:

— Выбор метода решения.

— Выбор подходящих структур данных.

— Разработка алгоритмов решения.

— Реализация программы на языке С++.

— Описание пользовательского интерфейса.

# **1 Постановка задачи**

Предоставить пользователю возможность выбирать тематику тестирования (для каждой тематики свой файл). Вопросы теста должны появляться случайным образом и не повторяться. Программа должна выставлять оценку по следующему правилу: «*отлично*» – за правильные ответы на 86-100% вопросов, «*хорошо*» – если испытуемый правильно ответил на 71-85% вопросов, «*удовлетворительно*» – если правильных ответов 60-70%, «*плохо*» – если правильных ответов менее 60%.

## **2 Обоснование выбора метода решения поставленной задачи**

Реализация поставленной задачи была возможна двумя методами: консольным или графическим приложением.

Ссылаясь на источник [6], были выявлены нижеперечисленные преимущества и недостатки текстового интерфейса.

Преимущества:

* используя небольшую комбинацию клавиш, может быть вызвана любая команда;
* если пользователь не успел разобрать какую-либо быстро прокрученную информацию, то её можно заново увидеть, пролистнув содержимое консольного приложения;
* из-за отсутствия определённых частей интерфейса (рамки окон, пусковые панели), командная строка допускает помещение заметно большего количества текста на страницу;
* при проведении параллелей с Graphical User Interface (GUI), командная строка потребляет меньше оперативной памяти ЭВМ.

Недостатки:

* необходимость запоминая сокращений, а также проработки построения команд осложняется и тем, что любая команда может располагать своими уникальными символами;
* в командной строке отсутствует автоматическое дополнение слов, поэтому ввод команд большой длинны и/или параметров со специальными символами с клавиатуры осложняется.

Также, основываясь на информации из источников [3] и [7] были выявлены нижеперечисленные преимущества и недостатки графического интерфейса.

Преимущества:

* пользователям, начавшим знакомиться с компьютером при помощи графического интерфейса, будет проще в нём освоиться из-за знакомого и простого интерфейса;
* определённые программы нередко можно реализовать только при помощи графического интерфейса (к примеру, программы, работающие с графикой).

Недостатки:

* если сравнивать с консольным приложением, графический интерфейс потребляет больше памяти;
* удаленная работа сложнее организовывается;
* для пользователей, начавших работу с консольного интерфейса, графический интерфейс окажется менее «доброжелательным».

Проанализировав вышеизложенную информацию о графическом и текстовом интерфейсе, выбор был сделан в пользу графического интерфейса из-за следующих причин:

1. Графический интерфейс более понятен пользователю, так как он привык работать с оконными приложениями, где есть кнопки, рамки, подписаны различные названия.

2. Приложение с графическим интерфейсом более простое для использования пользователю, так как он на интуитивном уровне может понять, как использовать ту или иную кнопку (так как она подписана).

3. Все действия (помимо ввода данных) выполняются кликами мышкой, что также является более привычным обычному человеку, нежели чем поиск и нажатие нужной для выполнения определённого действия клавиши на клавиатуре.

В связи с этим, для реализации поставленной задачи создания «Тестирующей системы» была выбрана среда С++ Builder, позволяющая создавать оконные графические приложения с помощью набора стандартных компонентов данной среды. Конкретно для данной задачи, С++ Builder оказался более полезен, чем использование командной строки по нескольким причинам:

1. Данная графическая среда для визуального отображения информации поддерживает так называемые формы («*Form»*), которые соответствуют обычным окнам, что, в свою очередь, более привычно для обращения пользователю, чем командная строка.
2. Пользователю для прохождения теста придётся совершать более знакомые и привычные для него действия: нажатие на кнопки посредством курсора мыши, выбор возможного варианта ответа при помощи щелчка по выбранному варианту (при использовании консольного приложения, все действия пользователю пришлось бы совершать при помощи ввода символов с клавиатуры, что менее привычно обычному человеку).

## **3 Обоснование выбора структуры данных**

Для реализации поставленной задачи можно было использовать такие структуры данных, как:

1. Вектор или массив для хранения данных.
2. Класс или структура для объединения полей и методов в связные группы.
3. Бинарный и текстовый файл для хранения информации о тесте.

В первом пункте выбор был сделан в пользу вектора по той причине, что вектор является стандартным динамическим массивом, реализованным как класс со своими уже реализованными методами, что делает работу с этим массивом намного проще.

Далее, между классом и структурой выбор был сделан в пользу первого по той причине, что при использовании объектно-ориентированного метода программирования класс является основной единицей структуры данных.

В третьем пункте выбор был сделан в пользу текстового файла, а не бинарного по той причине, что в файле бинарного типа информация содержится в виде последовательности байт, а в текстовом – в символьном. Так как тесты считываются из файла, то удобнее было использовать текстовый файл, чтобы была прямая возможность добавления, удаления или изменения информации по тестам.

## **4 Описание алгоритмов решения задачи**

Общий алгоритм работы программы представлен на рисунке 1 и состоит в следующем:

1. Запуск приложения.
2. Ввод имени пользователя (по желанию).
3. Выбор одной из доступных тем для тестирования.
4. В зависимости от выбранной темы, из соответствующего файла происходит считывание информации о тесте.
5. Прохождение пользователем теста (ответ на вопросы посредством выбора одного из четырех вариантов ответа).
6. Вывод результатов.
7. Выход из программы или прохождение нового теста.

Алгоритм загрузки полной информации о тесте из файла представлен на рисунке 2 и состоит в следующем:

1. В специально отведенную переменную записывается первое число в файле, соответствующее количеству вопросов.
2. Если считанное количество вопросов не равно нулю, то в цикле построчно считываются:
   1. Вопрос (в соответствующую ему переменную типа «*string*»).
   2. 4 варианта ответа (также в соответствующие им переменные типа «*string*»).
   3. Номер правильного варианта (в соответствующую ему переменную типа «*int*»).
   4. Количество повторений цикла равно количеству вопросов, считанному ранее из файла.
3. Вызывается конструктор, принимающий как параметры считанные ранее данные создается объект, который записывается в созданный вектор.



Рисунок 1 – Схема общего алгоритма работы программы

Данный алгоритм срабатывает каждый раз, когда пользователь в предоставленном ему меню выбирает тему вопроса, и после этого открывается соответствующий выбранному пользователем текстовый файл, откуда и происходит считывание данных.

Алгоритм тестирования представлен на рисунке 3 и состоит в следующем:

1. Загрузка вопроса, вариантов ответа и номера правильного ответа.
2. Вывод вопроса и соответствующих ему вариантов ответа на экран пользователя.
3. Сравнение выбранного пользователем варианта ответа с правильным: если выбран правильный, то счетчик правильных ответов увеличивается на единицу; если выбран неправильный, то загружается следующий вопрос.
4. Загрузка следующего вопроса, вариантов ответа и правильного номера ответа.

Данный алгоритм работает в цикле количество раз, равное считанному из файла количеству вопросов (первая строка в текстовом файле).

Алгоритм вывода результата тестирования, представленный на рисунке 4, состоит в следующем:

1. Число правильных ответов (вычисляемое в процессе работы алгоритма тестирования пользователя) делится на число всех ответов (считанное ранее из файла), явно приводясь к вещественному типу «*float*».
2. В зависимости от дробной части выводится соответствующая результату надпись в совокупности с именем пользователя (если он ввел его в самом начале).



Рисунок 2 – Схема алгоритма загрузки полной информации о вопросе из файла



Рисунок 3 – Схема алгоритма тестирования



Рисунок 4 – Схема вывода результата тестирования

## **5 Особенности программной реализации**

Основной структурой данных в программе является класс «*Question»*, позволяющий хранить всю информацию о выводимом пользователю вопросе. На С++ структура данных выглядит так:

1. Поле *«question»* типа «*string»*, в котором хранится сам вопрос.
2. Поле *«answer1»* типа «*string»*, в котором хранится вариант ответа №1.
3. Поле *«answer2»* типа «*string»*, в котором хранится вариант ответа №2.
4. Поле *«answer3»* типа «*string»*, в котором хранится вариант ответа №3.
5. Поле *«answer4»* типа «*string»*, в котором хранится вариант ответа №4.
6. Поле *«trueAnswer»* типа «*int»*, в котором хранится номер правильного варианта ответа.

Все поля данного класса являются приватными в соответствии с одним из главных принципов построения классов – инкапсуляцией. Это сделано для того, чтобы обезопасить вводимые данные от вмешательств извне или неправильного использования.

Также, класс имеет несколько методов:

1. Конструктор «*Question»*, принимающий на вход 6 параметров: 5 параметров типа «*string»*, соответствующих вопросу и 4 вариантам ответа, и 1 параметр типа «*int»*, соответствующий номеру правильного варианта ответа. Конструктор вызывается при создании объекта класса «*Question»*.
2. Метод «*getQuestion ()*» типа «*string*», который возвращает вопрос.
3. Метод «*getAnswer1* *()*» типа «*string*», который возвращает вариант ответа №1.
4. Метод «*getAnswer2* *()*» типа «*string*», который возвращает вариант ответа №2.
5. Метод «*getAnswer3* *()*» типа «*string*», который возвращает вариант ответа №3.
6. Метод «*getAnswer4* *()*» типа «*string*», который возвращает вариант ответа №4.
7. Метод «*getTrueAnswer* *()*» типа «*int*», который возвращает номер правильного варианта ответа.

Методы 2 – 7 введены для того, чтобы была возможность просмотра содержимого скрытых (private) полей класса.

Данный класс представлен в Приложении А: в файле «*Question.h*» представлены поля и прототипы методов, а в файле «*Question.cpp*» представлена реализация данных методов.

Так как тест включает в себя не один вопрос, а некоторое множество вопросов, то для их хранения как объектов вышеуказанного класса был использован vector – стандартный контейнер библиотеки STL языка C++ по той причине, что он имеет значительный ряд преимуществ в сопоставлении с обычным динамическим массивом:

1. Более быстрый в сравнении с массивом поиск элементов в векторе по индексу с использованием функции «*at ()»* или обычного обращения по индексу [].
2. Простое изменение размеров вектора: добавление новых элементов с помощью функции «*push\_back* *()»*; удаление элемента с помощью функции «*erase* *()»* (как следствие, мы избавляемся от необходимости «ручного» выделение и перераспределения памяти, как в случае с обычным массивом).
3. Автоматическое освобождение используемой памяти после удаления элемента из вектора.

Для хранения списка вопросов с вариантами ответов и номером правильного ответа используется текстовый файл, позволяющий просматривать его содержимое и, при необходимости, изменять его: добавить или удалить какой-либо вопрос или изменить информацию о каком-то конкретном вопросе.

Сам файл представляет собой структуру, хранящую в себе: количество вопросов в данном файле, вопрос на отдельной строке, 4 варианта ответа (каждый на отдельной строке) и номер правильного варианта ответа (также на отдельной строке).

Алгоритм тестирования, представленный на рисунке 3, запускается после выбора пользователем интересующей его темы; реализован в методе кнопки «Следующий вопрос», вызывающемся при нажатии пользователем на данный пункт формы:

**void \_\_fastcall** TForm3::Button2Click ()

Основная цель данного метода – последовательный вывод вопросов и соответствующих им вариантов ответов из вектора объектов, а также проверка на правильность выбранного ответа и увеличение счетчика правильных ответов на единицу, если такой был действительно правильным. Для вывода вопросов и вариантов ответа используется компонент «*RadioGroup*», при открытии которого в верхнюю строку заносится вопрос и далее в 4 компонентах «*RadioButton*» располагаются варианты ответа [4].

Запись введённого пользователем имени происходит в поле компонента «*Edit*» и осуществляется по нажатии на кнопку «*Принять*»:

**void\_\_fastcall** TForm1::Edit1KeyPress()

Метод при вызове производит проверку на соответствие своего параметра «*Key*» набору допустимых для данного поля символов. Если пользователем был введён символ, не соответствующий букве, то символ удалится, пользователь получит сообщение о неправильно введённом символе и предложение попытаться ещё раз ввести свое имя [5].

## **6 Описание пользовательского интерфейса**

Главный экран (рисунок 5) приложения представляет собой набор из следующих основных компонентов: приветствующей надписи, поля для ввода пользователем своего имени, кнопки «*Перейти к тестированию*», по нажатии на которую загружается меню, состоящее из доступных тем для прохождения теста, и кнопки «*Выход*», по нажатии на которую происходит закрытие приложения.

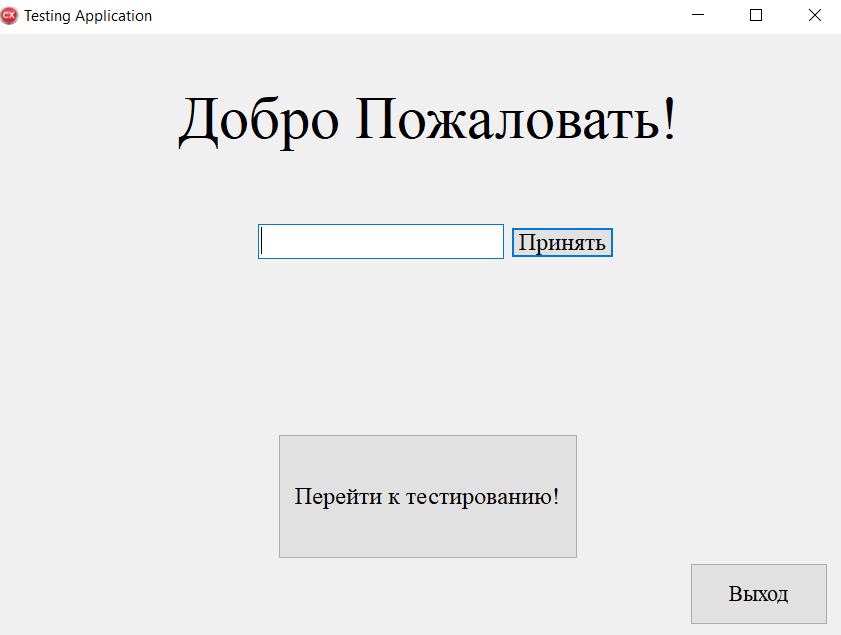


Рисунок 5 - Главный экран

Далее, после нажатия на кнопку «*Перейти к тестированию*», пользователю предоставляется 4 темы для прохождения теста на выбор (рисунок 6). Кнопка «*Назад*» служит для выхода в главное меню.

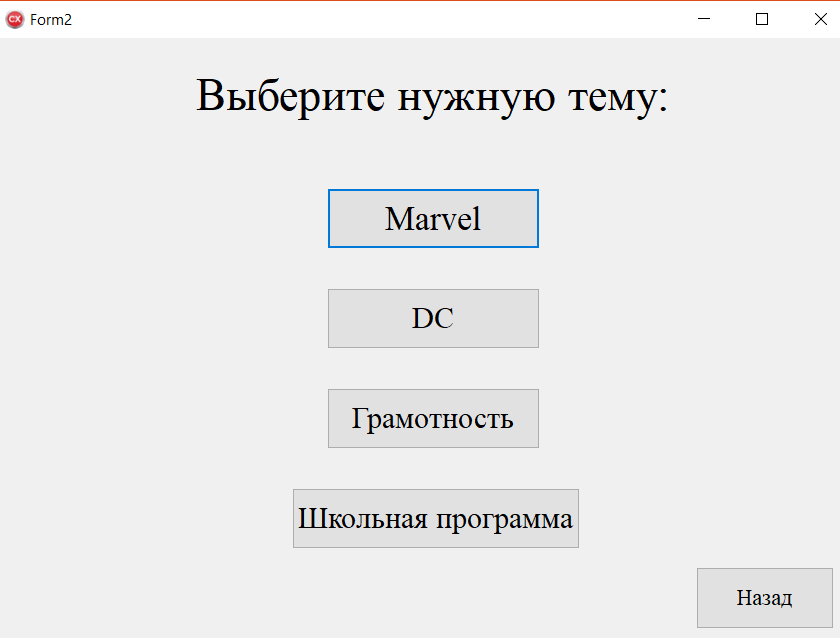


Рисунок 6 - Меню выбора темы теста

Далее по выбранной пользователем теме загружается тест. Вверху написано название темы, по которой проходится в данный момент тест. На экран выводится вопрос и 4 варианта ответа (рисунок 7). Пользователь должен нажать на возможный по его мнению вариант. Пользователь может не выбрать никакого варианта. Тогда его ответ засчитается, как неправильный и по нажатии на кнопку «*Следующий вопрос*» будет загружен следующий вопрос соответственно. Кнопка «*Следующий вопрос*» служит для подтверждения пользователем своего выбора и загрузки следующего вопроса.

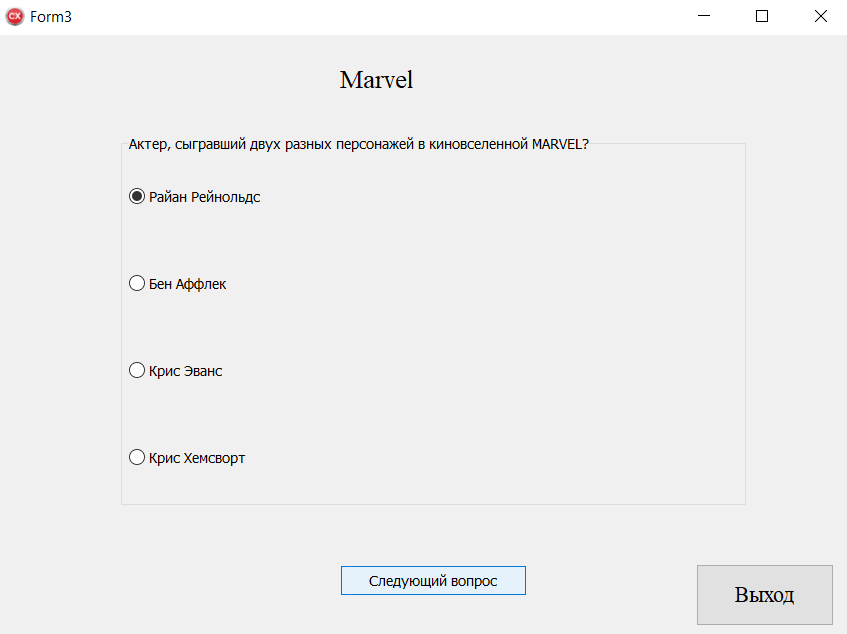


Рисунок 7 - Процесс тестирования

После прохождения процесса тестирования, пользователю на экран будет выведено окно, содержащее в себе один из четырёх вариантов результата тестирования, зависящий от количества правильных вопросов:

* «*Плохо*» – если правильных ответов менее 60% (Рисунок 8).
* «*Удовлетворительно*» – если правильных ответов 60-70% (Рисунок 9).
* «*Хорошо*» – если испытуемый правильно ответил на 71-85% вопросов (Рисунок 10).
* «*Отлично*» – за правильные ответы на 86-100% вопросов (Рисунок 11).

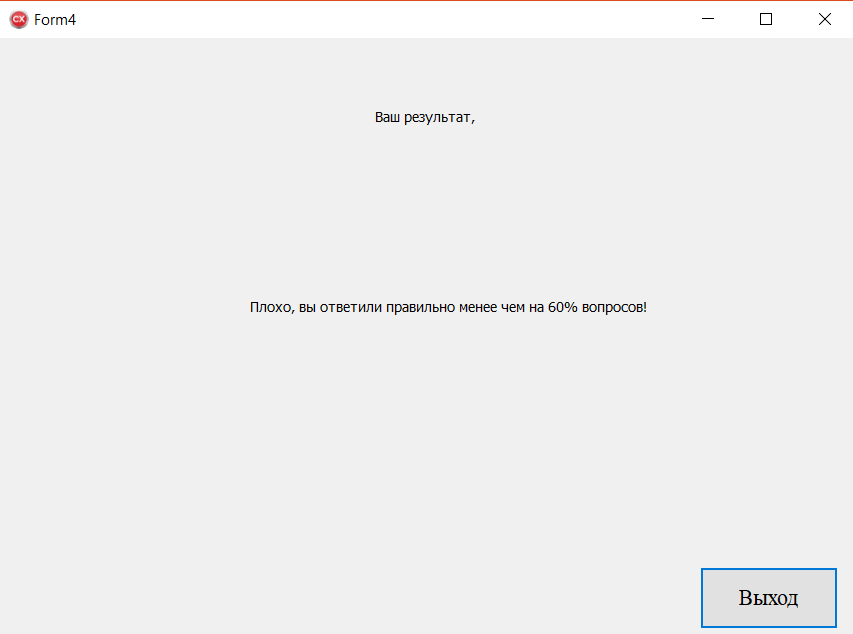


Рисунок 8 - Первый вариант результата прохождения теста

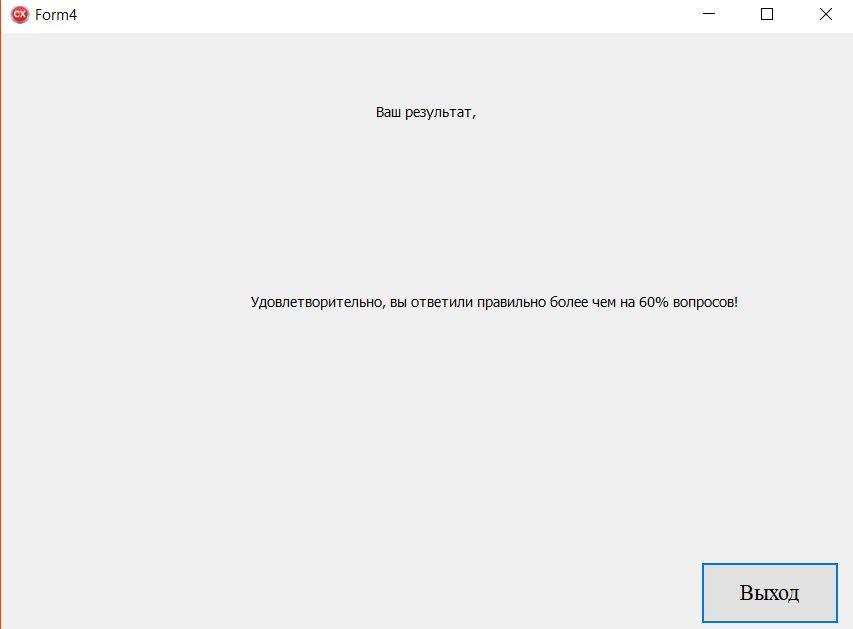
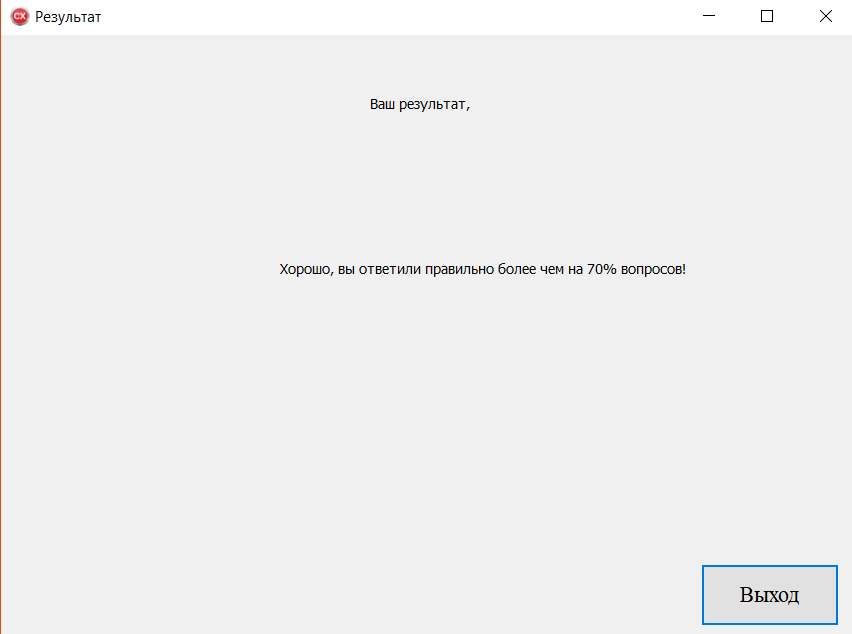


Рисунок 9 - Второй вариант результата прохождения теста



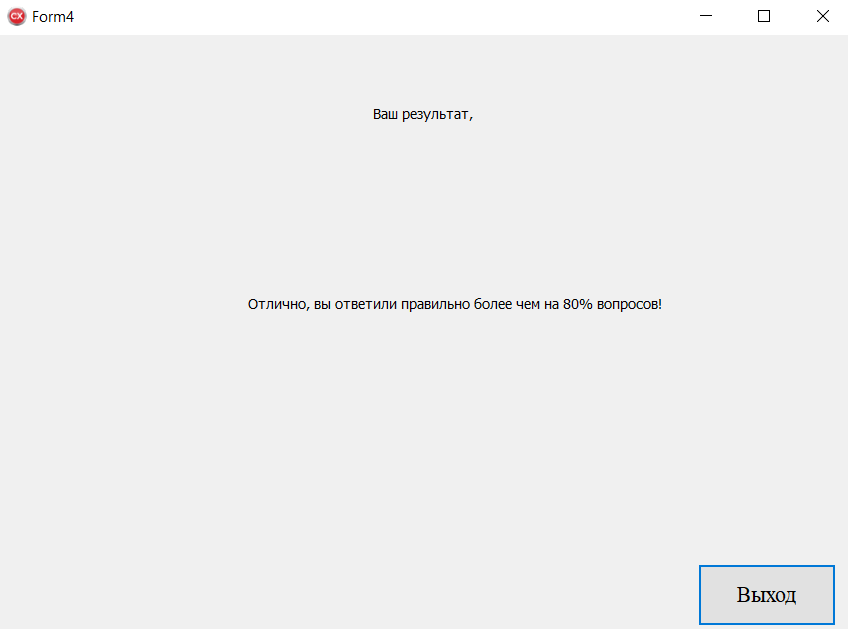
Рисунок 10 - Третий вариант результата прохождения теста

Рисунок 11 - Четвёртый вариант результата прохождения теста

## **Заключение**

Таким образом, поскольку следующие задачи:

— выбор метода решения;

— выбор подходящих структур данных;

— разработка алгоритмов решения;

— реализация программы на языке С++;

— описание пользовательского интерфейса

реализованы, то поставленную цель разработки программы тестирования можно считать достигнутой.

В ходе выполнения курсовой работы мной были закреплены знания по работе с объектами, классами, файлами и графической средой разработки С++ Builder, поддерживающей язык С++.

## **Список использованных источников**

1. Архангельский, А. Я. Язык C++ в C++Builder: справочное и методическое пособие [Текст] / А. Я. Архангельский. – М.: Бином, 2008. – 942 с.
2. Шилдт Г. Самоучитель С++, 3-е издание [Текст] / Г. Шилдт. – Санкт-Петербург: БХВ-Петербург, 2003. – 688 с.
3. Гращенко Л. А. Обобщенная модель угроз информационной безопасности визуальных интерфейсов пользователя // Известия Орловского государственного технического университета. Серия: Информационные системы и технологии [Текст]/ Л. А. Гращенко. – Орёл: ОГУ 2006. – №. 1. – 393 с.
4. Холингвэрт Д., Баттерфилд Д., Сворт Б., Оллсоп Д. C++Builder 5. Руководство разработчика [Текст] / Д. Холингвэрт, Д. Баттерфилд, Б. Сворт, Д. Оллсоп. – М.: Вильямс, 2001. – 869 c.
5. Культин Н.Б. Самоучитель C++ Builder [Текст] / Н.Б. Культин. – СПб.: БХВ-Петербург, 2004. – 320 c.
6. Свободная энциклопедия Википедия, статья «Текстовый интерфейс пользователя» [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Текстовый_интерфейс_пользователя> (дата обращения 18.04.18).
7. Свободная энциклопедия Википедия, статья «Графический интерфейс пользователя» [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Графический_интерфейс_пользователя> (дата обращения 24.04.18).

## **Приложение А**

## **(обязательное)**

## **Листинги программы**

Листинг 1 – Файл «Question.h»

#include <iostream>

#include <string>

using namespace std;

class Question{

private:

string question;

string answer1;

string answer2;

string answer3;

string answer4;

int trueAnswer;

public:

Question (string, string, string, string, string, int);

string getQuestion ();

string getAnswer1 ();

string getAnswer2 ();

string getAnswer3 ();

string getAnswer4 ();

int getTrueAnswer ();};

Листинг 2 – Файл «Question.cpp»

#include <iostream>

#include <string>

#include "Question.h"

Question::Question(string Q, string A1, string A2, string A3, string A4, int TA){

question = Q;

answer1 = A1;

answer2 = A2;

answer3 = A3;

answer4 = A4;

trueAnswer = TA;

}

string Question::getQuestion (){return question;}

string Question::getAnswer1 (){return answer1;}

string Question::getAnswer2 (){return answer2;}

string Question::getAnswer3 (){return answer3;}

string Question::getAnswer4 (){return answer4;}

int Question::getTrueAnswer (){return trueAnswer;}

Листинг 3 – Файл «TestingApplication.h»

#ifndef TestingApplicationH

#define TestingApplicationH

#include <System.Classes.hpp>

#include <Vcl.Controls.hpp>

#include <Vcl.StdCtrls.hpp>

#include <Vcl.Forms.hpp>

class TForm1 : public TForm{

\_\_published:

TLabel \*Label1;

TEdit \*Edit1;

TButton \*Button1;

TButton \*Button2;

TButton \*Button3;

void \_\_fastcall Button2Click(TObject \*Sender);

void \_\_fastcall Button3Click(TObject \*Sender);

void \_\_fastcall Button1Click(TObject \*Sender);

void \_\_fastcall Edit1KeyPress(TObject \*Sender, System::WideChar &Key);

public: // User declarations

\_\_fastcall TForm1(TComponent\* Owner);

UnicodeString usersName;

};

extern PACKAGE TForm1 \*Form1;

#endif

Листинг 4 – Файл «TestingApplication.cpp»

#include <vcl.h>

#include <iostream>

#include <string>

#pragma hdrstop

#include "TestingApplication.h"

#include "TestMenu.h"

#include "RezultMenu.h"

#pragma package(smart\_init)

#pragma resource "\*.dfm"

using namespace std;

TForm1 \*Form1;

//---------------------------------------------------------------------------

\_\_fastcall TForm1::TForm1(TComponent\* Owner): TForm(Owner){}

//---------------------------------------------------------------------------

void \_\_fastcall TForm1::Button2Click(TObject \*Sender){Close();}

//---------------------------------------------------------------------------

void \_\_fastcall TForm1::Button3Click(TObject \*Sender){Form2->ShowModal();}

//---------------------------------------------------------------------------

void \_\_fastcall TForm1::Button1Click(TObject \*Sender){

usersName = Edit1->Text;

}

//---------------------------------------------------------------------------

void \_\_fastcall TForm1::Edit1KeyPress(TObject \*Sender, System::WideChar &Key){

if ( (( Key >= '0') && ( Key <= '9')) || Key == '-' || Key == '+' || Key == '\*' || Key == '/' || Key == '=' || Key == '.' || Key == ',' || Key == '?' ){

ShowMessage("В имени не допускаются числа");

Key='\0';

Edit1->SetFocus();

return;

}

}

Листинг 5 – Файл «TestMenu.h»

#ifndef TestMenuH

#define TestMenuH

#include <System.Classes.hpp>

#include <Vcl.Controls.hpp>

#include <Vcl.StdCtrls.hpp>

#include <Vcl.Forms.hpp>

class TForm2 : public TForm{

\_\_published:

TLabel \*Label1;

TButton \*Button1;

TButton \*Button2;

TButton \*Button3;

TButton \*Button4;

TButton \*Button5;

void \_\_fastcall Button5Click(TObject \*Sender);

void \_\_fastcall Button1Click(TObject \*Sender);

void \_\_fastcall Button2Click(TObject \*Sender);

void \_\_fastcall Button3Click(TObject \*Sender);

void \_\_fastcall Button4Click(TObject \*Sender);

public: // User declarations

int way;

\_\_fastcall TForm2(TComponent\* Owner);

};

extern PACKAGE TForm2 \*Form2;#endif

Листинг 6 – Файл «TestMenu.cpp»

#include <vcl.h>

#pragma hdrstop

#include "TestMenu.h"

#include "Test1.h"

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm2 \*Form2;

//---------------------------------------------------------------------------

\_\_fastcall TForm2::TForm2(TComponent\* Owner): TForm(Owner){}

//---------------------------------------------------------------------------

void \_\_fastcall TForm2::Button5Click(TObject \*Sender){Close();}

//---------------------------------------------------------------------------

void \_\_fastcall TForm2::Button1Click(TObject \*Sender){

way = 1;

Application->CreateForm(\_\_classid(TForm3), &Form3);

Form3->ShowModal();

Form3->Free();

}

//---------------------------------------------------------------------------

void \_\_fastcall TForm2::Button2Click(TObject \*Sender){

way = 2;

Application->CreateForm(\_\_classid(TForm3), &Form3);

Form3->ShowModal();

Form3->Free();

}

//---------------------------------------------------------------------------

void \_\_fastcall TForm2::Button3Click(TObject \*Sender){

way = 3;

Application->CreateForm(\_\_classid(TForm3), &Form3);

Form3->ShowModal();

Form3->Free();

}

//---------------------------------------------------------------------------

void \_\_fastcall TForm2::Button4Click(TObject \*Sender){

way = 4;

Application->CreateForm(\_\_classid(TForm3), &Form3);

Form3->ShowModal();

Form3->Free();

}

Листинг 7 – Файл «Test1.h»

#ifndef Test1H

#define Test1H

#include <System.Classes.hpp>

#include <Vcl.Controls.hpp>

#include <Vcl.StdCtrls.hpp>

#include <Vcl.Forms.hpp>

#include <Vcl.ExtCtrls.hpp>

#include <Vcl.Buttons.hpp>

class TForm3 : public TForm{

\_\_published:

TButton \*Button1;

TRadioGroup \*RadioGroup1;

TButton \*Button2;

TLabel \*Label3;

void \_\_fastcall Button1Click(TObject \*Sender);

void \_\_fastcall Button2Click(TObject \*Sender);

void \_\_fastcall FormCreate(TObject \*Sender);

public: // User declarations

int countOfQues;

int countOfTrueAns;

\_\_fastcall TForm3(TComponent\* Owner);

};

extern PACKAGE TForm3 \*Form3;

#endif

Листинг 8 – Файл «Test1.cpp»

#include "Test1.h"

#include "TestMenu.h"

#include <iostream>

#include <string>

#include "RezultMenu.h"

using namespace std;

#include <vector>

#include "Question.h"

#include <vcl.h>

#include <fstream>

#pragma hdrstop

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm3 \*Form3;

vector <Question> quesVect;

int countTrueAns = 0;

int i = 0;

int countOfTrueAns = 0;

\_\_fastcall TForm3::TForm3(TComponent\* Owner): TForm(Owner){}

//---------------------------------------------------------------------------

void \_\_fastcall TForm3::Button1Click(TObject \*Sender){

quesVect.clear();

countOfTrueAns = 0;

i = 0;

Close();

}

//---------------------------------------------------------------------------

void \_\_fastcall TForm3::FormCreate(TObject \*Sender){

RadioGroup1->ItemIndex = -1;

string ques;

string ans1;

string ans2;

string ans3;

string ans4;

int trueAns;

fstream file;

switch(Form2->way) {

case 1: {

file.open("C:\\Kyrsach RAD Studio\\Marvel.txt", ios::in );

Form3->Caption = "Тест \"Marvel\"";

Label3->Caption = "Marvel";

break;

}

case 2: {

file.open("C:\\Kyrsach RAD Studio\\DC.txt", ios::in );

Form3->Caption = "Тест \"DC\"";

Label3->Caption = "DC";

break;

}

case 3: {

file.open("C:\\Kyrsach RAD Studio\\Literacy.txt", ios::in );

Form3->Caption = "Тест \"Грамотность\"";

Label3->Caption = "Грамотность";

break;

}

case 4: {

file.open("C:\\Kyrsach RAD Studio\\School.txt", ios::in );

Form3->Caption = "Тест \"Школьная программа\"";

Label3->Caption = "Школьная программа";

break;

}

file >> countOfQues;

for (int j = 0; j < countOfQues; j++) {

getline(file, ques);

getline(file, ques);

getline(file, ans1);

getline(file, ans2);

getline(file, ans3);

getline(file, ans4);

file >> trueAns;

quesVect.push\_back(Question::Question(ques, ans1, ans2, ans3, ans4, trueAns));

}

i = 0;

file.close();

if(quesVect.size() != 0){

i = rand() % quesVect.size();

RadioGroup1->Items->Clear();

RadioGroup1->Caption = quesVect[i].getQuestion().c\_str() ;

RadioGroup1->Items->Add(quesVect[i].getAnswer1().c\_str());

RadioGroup1->Items->Add(quesVect[i].getAnswer2().c\_str());

RadioGroup1->Items->Add(quesVect[i].getAnswer3().c\_str());

RadioGroup1->Items->Add(quesVect[i].getAnswer4().c\_str());

}

}

void \_\_fastcall TForm3::Button2Click(TObject \*Sender){

Button2->Caption = "Следующий вопрос";

if(RadioGroup1->ItemIndex == quesVect[i].getTrueAnswer() - 1){

countOfTrueAns++;

}

quesVect.erase(quesVect.begin() + i);

if(quesVect.size() != 0){

i = rand() % quesVect.size();

RadioGroup1->Items->Clear();

RadioGroup1->Caption = quesVect[i].getQuestion().c\_str() ;

RadioGroup1->Items->Add(quesVect[i].getAnswer1().c\_str());

RadioGroup1->Items->Add(quesVect[i].getAnswer2().c\_str());

RadioGroup1->Items->Add(quesVect[i].getAnswer3().c\_str());

RadioGroup1->Items->Add(quesVect[i].getAnswer4().c\_str());

}

else if (quesVect.size() == 0){

Button2->Caption = "Закончить тестирование";

Application->CreateForm(\_\_classid(TForm4), &Form4);

Form4->ShowModal();

Form4->Free();

Form3->Close();

}

}

Листинг 9 – Файл «RezultMenu.h»

#ifndef RezultMenuH

#define RezultMenuH

#include <System.Classes.hpp>

#include <Vcl.Controls.hpp>

#include <Vcl.StdCtrls.hpp>

#include <Vcl.Forms.hpp>

class TForm4 : public TForm{

\_\_published:

TLabel \*Label1;

TButton \*Button1;

TLabel \*Label2;

void \_\_fastcall Button1Click(TObject \*Sender);

void \_\_fastcall FormShow(TObject \*Sender);

public: // User declarations

\_\_fastcall TForm4(TComponent\* Owner);

};

extern PACKAGE TForm4 \*Form4;

#endif

Листинг 10 – Файл «RezultMenu.cpp»

#include <vcl.h>

#pragma hdrstop

#include "RezultMenu.h"

#include "Test1.h"

#include "TestingApplication.h"

#pragma package(smart\_init)

#pragma resource "\*.dfm"

TForm4 \*Form4;

\_\_fastcall TForm4::TForm4(TComponent\* Owner): TForm(Owner){}

void \_\_fastcall TForm4::Button1Click(TObject \*Sender){

Close();

Form3->Close();

}

void \_\_fastcall TForm4::FormShow(TObject \*Sender){

Label1->Caption = "Ваш результат, " + Form1->usersName;

float rez = (float)Form3->countOfTrueAns / Form3->countOfQues;

if(rez >0.85){Label2->Caption = "Отлично, вы ответили правильно более чем на 80% вопросов!";}

else if (rez >= 0.71 && rez <= 0.85) {Label2->Caption = "Хорошо, вы ответили правильно более чем на 70% вопросов!"; }

else if (rez >= 0.6 && rez < 0.71) {Label2->Caption = "Удовлетворительно, вы ответили правильно более чем на 60% вопросов!";}

else if (rez < 0.6 && rez > 0) { Label2->Caption = "Плохо, вы ответили правильно менее чем на 60% вопросов!";}

else{Label2->Caption = "Странно, это не должно выскочить. Проблема: либо ответ отрицательный, либо ещё что-то";}

}