Частное образовательное учреждение высшего образования
«Международный Институт Дизайна и Сервиса»
(ЧОУВО МИДиС)

КУРСОВАЯ РАБОТА

по дисциплине ОП.05 Основы программирования
на тему: «Разработка программного комплекса Система тестирования»

специальности 09.02.03 Программирование в компьютерных системах

Руководитель курсовой работы:
Прилепина Е.В.

Выполнил: Кабанец Б.С.
обучающийся 2 курса группы П-22

Челябинск 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ **3**

ГЛАВА 1. ТЕСТИРОВАНИЕ **4**

1.1 История тестирования 4
1.2 Виды тестов 5
1.3 Преимущества и недостатки тестирования 6
1.4 Практика применения тестов 6

ГЛАВА 2. РАЗРАБОТКА ПРИЛОЖЕНИЯ **7**

2.1 Цель работы и требования 7
2.2 Обзор и история развития C++ 7
2.3 Обзор библиотеки FLTK 8
2.4 Архитектура 11
2.5 Классы 12
2.6 Точка входа и основной цикл 14

ЗАКЛЮЧЕНИЕ **17**

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ **18**

ВВЕДЕНИЕ

Актуальность данной работы заключается в том, что присутствует необходимость в средствах для тестирования различных субъектов на соответствие навыкам.

Объектом исследования является «тест».

Предмет исследования: программный комплекс «Система тестирования».

Цель работы: проектирование и разработка программного комплекса «Система тестирования» по теме «Операционные системы».

Задачи:

1. изучение предметной области
2. разработка технического задания
3. разработка десктопного приложения

ГЛАВА 1. ТЕСТИРОВАНИЕ

1.1 История тестирования

«История развития тестов своими корнями уходит в глубину веков, она связана с измерением различных способностей, знаний, умений и навыков. Эти измерения можно считать предысторией тестов. Уже в середине III тысячелетия до н. э. в Древнем Вавилоне проводились испытания выпускников в школах писцов на знание арифметических действий, умение измерять поля, распределять рационы, делить имущество, владение искусством пения и игры на музыкальных инструментах.»[5] Похожие практики также имели место в Древнем Египте, Древнем Китае, средневековом Вьетнамском государстве и многих других странах.

Периодом становления тестирования, как повсеместной практики, можно считать период с 80-х годов XIX века до 20-х годов XX века. Причиной распространения тестирования послужила потребность в исследовании индивидуальных различий.

Выделяют 4 этапа развития тестирования:

- Обозначенный выше период зарождения.
- 20–60-е годы XX века. Происходит разделение тестов на психологические и педагогические.
- с начала 60-х по конец 70-х годов XX века. Тесты начинают разрабатывать в рамках биосоциологизаторской концепции развития личности.
- с 80-х годов XX века по настоящее время. Характеризуется широким использованием компьютеров, математических моделей и средств программирования.

«Таким образом, история возникновения и развития тестирования показывает, что этот метод контроля стал важным методом оценки, с помощью которого можно достаточно объективно,надежно измерить, обработать, интерпретировать результаты учебного процесса, диагностировать выраженность у индивида определенных психологических качеств, свойств и состояний, а также измерить уровень их развития.»[5]

1.2 Виды тестов

1. Дидактические тесты – это тесты, которые направлены на диагностическое измерение уровня знаний в определенной образовательной сфере.

2. Психологические тесты – это процедура установления и измерения индивидуально-психологических отличий.

   В качестве примеров психологических тестов, можно привести

   (a) Тест Сонди – это тест, выявляющий психологические отклонения, путём портретных выборов.

   (b) Тест Роршаха – это тест для определения особенностей личности, путём выявления тестируемым явных образов в разрозненных пятнах.

   (c) Тест руки – это тест для исследования личности, основывающийся на толковании испытуемым изображения руки.

3. Тестирование в IT

   (a) Модульное тестирование – процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы. Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода.

   (b) Интеграционные тесты – это тип тестирования, при котором программные модули объединяются логически и тестируются как группа.

   (c) Системные тесты – это тестирование, выполняемое на полной интегрированной системе, с целью проверки соответствия системы исходным требованиям.

   (d) Тестирование безопасности – это стратегия тестирования, используемая для проверки системы на безопасность, а также для анализа рисков.

1.3 Преимущества и недостатки тестирования

Преимущества

- Объективность, достигаемая путём стандартизации тестов
- Тестирование справедливо, т.к. ставит всех учащихся в равные условия
- Тестирование эффективно с экономической точки зрения

Недостатки

- Разработка качественных тестов – длительный и трудоёмкий процесс
- Тест не позволяет оценивать вероятностные, абстрактные и методологические знания

1.4 Практика применения тестов

По настоящему эффективные тесты должны разрабатываться только специалистами – людьми, обладающими профессиональными знаниями и опытом. Отличить стоящий тест можно очень просто: он либо снабжён кратким обзором, в котором приводятся обосновывающие его актуальность данные, либо хотя бы ссылкой на своего автора, который является специалистом в конкретной области. Но, даже используя только качественные и грамотные тесты, нельзя рассчитывать на то, что оценка личности человека и его способностей будет на 100% достоверной. Единственным способом оценить человека по каким бы то ни было параметрам всегда являлось и является непосредственное общение с ним.

Исходя из этого, можно сделать вывод, что тесты – это не серебряная пуля и пользоваться ими стоит по необходимости, совмещая с другими средствами контроля.

Вывод по первой главе

В данной главе была изучена тема тестирования. Получено представление, в каких областях применяют тестирование, для чего применяют тестирующие системы. Получены знания о классификации тестов.

ГЛАВА 2. РАЗРАБОТКА ПРИЛОЖЕНИЯ

2.1 Цель работы и требования

Разработать программу для проведения тестирования по теме «Операционные системы». Программа должна соответствовать следующим требованиям и включать следующие возможности:

- возможность работы с несколькими типами вопросов (один вариант ответа и много)
- уведомление о результате ответа на вопрос сразу
- возможность получать подсказку
- подведение итогов, включающее в себя кол-во верных ответов и кол-во использованных подсказок

2.2 Обзор и история развития C++

Изначально, когда язык C++ только появился в начале 1980-х, он представлял собой расширения языка Си, добавляющее в него прежде всего **классы**, **шаблоны** (*generic types*) и поддержку обработки **исключительных ситуаций**, что обеспечило поддержку языком **объектно-ориентированного**, **обобщенного** и не только стилей программирования.

Не стоит забывать, что являясь прямым наследником Си, C++ сохраняет с ним обратную совместимость, что требует тянуть за языком всё наследие языка Си, как хорошее, так и крайне спорное. Одна из основных черт языка C++, унаследованная от Си и отличающая его от большинства других языков – это **ручное управление памятью**.

Начиная с 1998 года, C++ начинают стандартизировать. Первым стандартом был C++98, далее вышел C++03, лишь исправляющий ошибки стандарта C++98. Стандарт C++11 привнёс в язык следующие существенные изменения:

- ключевое слово `auto` для автоматического вывода типа
- range–based циклы

- универсальная инициализация
- шаблоны с переменным числом аргументов и `std::tuple` (т. е. кортеж)
- `std::tie`
- лямбда–функции
- `nullptr`
- типы данных фиксированного размера, определённые в файле `cstdint`

2.3 Обзор библиотеки FLTK

FLTK(произносится «фултик») – это кроссплатформенная библиотека для построения GUI. Она поддерживает Linux, Windows и MacOS. Аббревиатура FLTK расшифровывается, как Fast Light Toolkit.

Рассмотрим простейший пример программы «Hello, World!», написанной с использованием FLTK. Она просто создаст пустое окно, с заголовком «Hello, World!», разрешением 640 на 480 и выведет в центре окна «Hello, World!».

```
#include <Fl/Fl.H>
#include <Fl/Fl_Window.H>
#include <Fl/Fl_Box.H>

enum {
    win_w = 640,
    win_h = 480,
    text_w = 120,
    text_h = 20
};

int main(int argc, char *argv[])
{
```

```
    Fl_Window *window = new Fl_Window(
                        win_w,
                        win_h,
                        "Hello,World!");
    Fl_Box *text = new Fl_Box(
                   win_w / 2 - text_w / 2,
                   win_h / 2 - text_h / 2,
                   text_w,
                   text_h,
                   "Hello, World!");
    window->end();
    window->show();
    return Fl::run();
};
```

Разберём этот код построчно и вместе с этим затронем саму архитектуру FLTK. В начале мы, как и в большинстве программ, подключаем заголовочные файлы:

```
#include <Fl/Fl.H>
#include <Fl/Fl_Window.H>
#include <Fl/Fl_Box.H>
```

Далее, мы объявляем глобальные константы, используя которые, мы будем задавать размерность и координаты виджетов:

```
enum {
    win_w = 640,
    win_h = 480,
    text_w = 120,
    text_h = 20
};
```

Следующими строчками, уже внутри `main`, мы создадим окно и текст,

используя для этого, заданные нами константы:

```
int main(int argc, char *argv[])
{
    Fl_Window *window = new Fl_Window(
                            win_w,
                            win_h,
                            "Hello,World!");
    Fl_Box *text = new Fl_Box(
                        win_w / 2 - text_w / 2,
                        win_h / 2 - text_h / 2,
                        text_w,
                        text_h,
                        "Hello, World!");
```

На этом моменте остановимся поподробнее и поясним, как это работает. Очевидно, что любое приложение с GUI должно иметь окно, в котором оно запускается, его мы и создаём в первой строчке. В качестве параметров конструктора окна, мы передаём ширину окна (`win_w`), высоту окна (`win_h`) и заголовок окна.

Для дальнейшего понимания, потребуется чуть рассказать о двух важнейших классах в FLTK, это классы `Fl_Widget` и `Fl_Group`.

Начнём с класса `Fl_Widget`, это базовый класс для большинства классов FLTK, как понятно из названия, он определяет виджет. **Виджет – это примитив графического интерфейса**, а к тому же, одна из основополагающих концепций, как FLTK, так и программирования GUI в принципе. В FLTK существует множество примитивов графического интерфейса, все они наследуются от `Fl_Widget` и имеют одинаковый конструктор, который требует задать координаты X/Y, ширину/высоту и текст (опционально).

Класс `Fl_Group` служит для объединения виджетов в группы (контейнеры). Работает он крайне просто, достаточно лишь создать экземпляр этого класса и все последующие созданные виджеты будут включены в группу, чтобы прекратить включение, нужно вызвать метод `end()` у

объекта группы.

Теперь, имея представление о работе FLTK, перейдём к разбору нашего примера. В первой строчке мы создали окно, задав его размер и заголовок. Во второй строчке, мы создали объект `text` типа `Fl_Box`, который представляет из себя виджет (то есть является потомком `Fl_Widget`) и реализует отрисовку бокса и надписи внутри него. По дефолту, бокс прозрачный. Объект `text` здесь находится внутри окна, потому что `window` является потомком класса `Fl_Group`, а значит реализует всю его функциональность, описанную выше.

Теперь нам нужно остановить добавление элементов в окно и показать его. Это делается следующим образом:

```
    window->end();
    window->show();
```

Программирование GUI основано на применение событийно - ориентированного подхода. Это означает, что нам понадобится бесконечный цикл, который будет мониторить действия пользователя, например закрытие приложения или нажатия клавиш (как виртуальных, так и реальных), и обрабатывать их, вызывая **обработчики событий**. Для вызова этого цикла, в конце приложения мы прописываем:

```
    return Fl::run();
};
```

2.4 Архитектура

Приступим к рассмотрению реализации программы. В отличие от рассмотренного выше примера, эта программа гораздо больше и рассматривать её построчно, было бы крайне не прагматично. Гораздо правильней будет обозреть её архитектуру, то есть довольно высокоуровневые вещи, а именно классы и их взаимосвязь.

Как говорилось выше, FLTK предоставляет нам возможность работы с графическими примитивами (кнопки, поля ввода, чекбоксы и т.д.). В приложении более-менее приличного размера оперировать примитивами

довольно неудобно, поэтому мы сгруппируем их в классы. Для этого, разделим окно приложение на 3 части. Сверху окна мы разместим текст вопроса, чуть ниже – варианты ответов и в самом низу – элементы управления в виде кнопок. Часть логики будет представлена непосредственно в файле `main`, часть мы разобьём по классам. Из трёх выделенных выше частей, в `main` будет представлены только элементы управления. Они будут вызывать callback – функции при нажатии и обращаться к классам, которые мы опишем ниже.

Наше приложение представляет из себя тест, проще говоря, сборник вопросов. Для их представления выделим 3 класса: виртуальный `Fl_Question` и 2 его потомка: `Fl_Single_Choice_Question` для вопроса с одним вариантом ответа и `Fl_Many_Choice_Question` для вопроса с несколькими вариантами ответа.

2.5 Классы

Рассмотрим для начала виртуальный `Fl_Question`. Данный класс, являясь виртуальным, берёт на себя минимум ответственности, однако служит своеобразным образцом для своих потомков. Прежде всего, он задаёт то, что имеется у любого вопроса, а именно **текст вопроса**, а также его тип. У нас имеется 3 типа вопроса, в виртуальном классе они определяется через `enum` переменную типа:

```
enum class QuestionType
{
    SINGLE_CHOICE,
    MANY_CHOICE,
    TEXT_FIELD
};
```

Как итог, получаем следующие `protected` (НЕ `private`, т.к. нам необходимо наследование) поля класса:

```
protected:
    Fl_Box *question; // Бокс для самого вопроса
    QuestionType question_type; // Тип вопроса
```

В классе будут определенны следующие **чисто виртуальные** методы, реализация которых зависит от потомка:

```
// Проверить правильность выбранного ответа
virtual int check_answer() = 0;

// Сбросить выбранные варианты
virtual void reset_answers() = 0;

// Удаление виджетов
virtual void delete_widgets() = 0;

// Инициализация виджетов
virtual void init_widgets() = 0;
```

Отличие *чисто* виртуального метода от виртуального, состоит в том, что его можно не реализовывать в базовом классе, ограничившись реализацией в классе–потомке.

Также мы определим в классе 2 обычных метода, один будет устанавливать текст вопроса, другой возвращать тип вопроса:

```
// Установить вопрос
void set_question(const char* q);

// Вернуть тип вопроса
QuestionType get_type();
```

Опустим их реализацию, она крайне проста.

Продолжим рассмотрением потомков класса `Fl_Question` и начнём с `Fl_Single_Choice_Question`, отвечающего за вопросы с одним вариантом ответа.

Вот его члены, находящиеся в `private` секции:

```
private:
```

```
    Fl_Radio_Round_Button *answers[4];
    int right_answ; // from 0 to 3
```

Здесь, массив `answers` представляет собой варианты ответов, а `right_answers` позицию в массиве `answers` с правильным ответом. Чуть ниже, мы подробнее рассмотрим, что представляет собой `Fl_Radio_Round_Button`, хотя вы уже можете догадаться по названию.

В нашем классе обретают свою реализацию все 4 чисто виртуальных метода класса–родителя `Fl_Question` и 2 обычных, `set_answers()` для установки вариантов ответа и `set_right_answer()` для установки номера верного варианта. Эти методы не определены как виртуальные в `Fl_Question`, потому что в зависимости от типа вопроса, они принимают разные аргументы.

2.6 Точка входа и основной цикл

Теперь давайте наконец скомпонуем наши классы в рабочее приложение. Надо сказать, что специфика GUI приложений плоха тем, что в процессе разработки, так или иначе, образуются глобальные переменные и функции.

И начнём мы рассмотрение не с функции `main`, а как раз таки с этих глобальных переменных и функций. Все данные, которыми оперирует наше приложение, а именно, вопросы, варианты ответов, верные ответы и прочее, заданы в коде, как константы. Вот так, например, представлены вопросы:

```
const char *questions[]
{
    "Какая файловая система используется в современных\
    версиях Windows?",
    "Что такое системный вызов?",
    "Что из этого, является интерфейсами системных вызовов?
  ",
    "Как называется абстракция, используемая для работы\
    с сетью в Linux?",
```

```
    "Как называется \" файловая система\" используемая\
    для получения информации о системе в Linux?",
    "Чем выражается файловый дескриптор в Linux?"
};
```

Примерно так же заданы подсказки (`const char *prompts[]`), варианты ответов (`const char* answers[][4]`), номера верных ответов для вопросов с одним вариантом ответа (`int right_answers[]`) и для вопросов с несколькими вариантами (`int right_answers_many[][4]`), а также массив, задающий типы этих вопросов
(`QuestionType question_types[]`).

Помимо данных, глобально представлены виджеты и константы размерности:

```
struct widgets
{
        Fl_Button *check_button;
        Fl_Button *next_button;
        Fl_Button *prompt_button;
        Fl_Text_Display *prompt_display;
}elements;

enum
{
    spacing_w = 20,
    spacing_h = 20,
    text_display_w = 940,
    text_display_h = 50,
    prompt_h = 240,
    radio_round_button_w = 20,
    radio_round_button_h = 20,
    button_h = 30,
    button_w = 90
};
```

В функции `main()`, которая является точкой входа в программу, происходит следующее:

1. Создание главного окна, используя константы размерности
2. Инициализация элементов структуры `elements`, используя константы размерности
3. Создание первого вопроса
4. Привязка к кнопкам callback–функций
5. Запуск главного цикла программы

ЗАКЛЮЧЕНИЕ

В ходе работы над курсовым проектом была рассмотрена история тестирования и понятие теста. Также была реализована программа для тестирования студентов на знание устройства операционных систем. В качестве её дальнейшего развития, логичнее всего было бы перенести вопросы из кода либо в файл, либо в БД.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Уорд Б., Внутреннее устройство Linux – СПб: Питер, 2018. – 384 с.

2. Руссинович М., Ионеску А., Соломон Д., Йосифович П., Внутреннее устройство Windows / М. Руссинович, А. Ионеску, Д. Соломон, П. Йосифович. – СПб: Питер, 2018. – 944 с.

3. Документация FLTK [Электронный ресурс] – https://www.fltk.org/doc-1.3/index.html

4. Каплун О.А., История возникновения и развития тестирования [Электронный ресурс] // Ученые записки Орловского государственного университета. Серия: Гуманитарные и социальные науки: Электрон. научн. ж. – 2008. – URL: https://cyberleninka.ru/article/n/istoriya-vozniknoveniya-i-razvitiya-testirovaniya (Дата обращения 24.04.2021)

5. Епанчинцева Г., Из истории тестов [Электронный ресурс] // Высшее образование в России: Электрон. науч. ж. – 2003. – URL: https://cyberleninka.ru/article/n/iz-istorii-testov (Дата обращения: 24.04.2021)

6. Саенко Н.М., К вопросу о типологиях тестов [Электронный ресурс] // Вестник Таганрогского института имени А.П. Чехова: Электрон. науч. ж. – 2009. – URL: https://cyberleninka.ru/article/n/k-voprosu-o-tipologiyah-testov (Дата обращения: 24.04.2021)

7. Столяров А.В., Программирование: введение в профессию. Том III: системы и сети – М.: МАКС Пресс, 2017. – 403 с.

8. Столяров А.В., Программирование: введение в профессию. Том IV: парадигмы – М.: МАКС Пресс, 2020. – 659 с.

9. Олифер В., Олифер Н., Сетевые операционные системы: Учебник для вузов / Н. Олифер, В. Олифер. – СПб: Питер, 2009. – 669 с.

10. Таненбаум Э., Современные операционные системы – СПб: Питер, 2015. – 1120 с.

11. Евдокимова Л.А., Составление тестов как метод закрепления учебного материала [Электронный ресурс] // Вопросы методики преподавания в вузе: Электрон. науч. ж. – 2004. – URL: https://cyberleninka.ru/article/n/sostavlenie-testov-kak-metod-zakrepleniya-uchebnogo-materiala (Дата обращения: 24.04.2021)

12. Бертрам Д., Стандарты тестов, квалификация и сертификация пользователей тестов [Электронный ресурс] // Психология. Психофизиология: Электрон. науч. ж. – 2014. – URL: https://cyberleninka.ru/article/n/standarty-testov-kvalifikatsiya-i-sertifikatsiya-polzovateley-testov (Дата обращения: 24.04.2021)

13. Страуструп Б., Программирование. Принципы и практика с использованием C++ – СПб: Диалектика, 2019. – 1328 с.

14. Лав Р., Ядро Linux. Описание процесса разработки – М: Вильямс, 2013. – 496 с.

15. Страуструп Б., Язык программирования C++. Краткий курс – СПб: Диалектика, 2019. – 320 с.