|  |  |
| --- | --- |
| Институт (факультет) | Институт информационных технологий |
| Кафедра | Программного и математического обеспечения ЭВМ |

1. **КУРСОВАЯ РАБОТА**

|  |
| --- |
| по дисциплине Объектно-ориентированное программирование |

|  |  |
| --- | --- |
| на тему | Объектно-ориентированное программирование на языке С++ |
|  | |

|  |
| --- |
| Выполнил студент группы 1ПИб-01-21оп |
| *группа* |
| направления подготовки (специальности) |
| 09.03.04 "Программная инженерия" |
| *шифр, наименование* |
| Зайцев Вадим Александрович |
| *фамилия, имя, отчество* |

|  |
| --- |
| Руководитель |
| Ершов Евгений Валентинович |
| *фамилия, имя, отчество* |
| доцент |
| *должность* |

|  |
| --- |
| Дата представления работы |
| «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_ г. |
|  |
| Заключение о допуске к защите |
|  |
|  |
|  |
|  |
| Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| количество баллов |
| Подпись преподавателя\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Череповец, 2020 год

Аннотация

Данная расчётно-пояснительная записка содержит 43 страниц, в том числе 10 рисунков, 5 таблиц и 3 приложения.

Оглавление

[Введение 4](#_Toc43706474)

[1. Объектно-ориентированный анализ предметной области 5](#_Toc43706475)

[2. Проектирование классов 8](#_Toc43706476)

[3. Логическая структура программы 9](#_Toc43706477)

[4. Модульная структура программы 10](#_Toc43706478)

[5. Тестирование 12](#_Toc43706479)

[5.1. Тестирование классов 12](#_Toc43706480)

[5.2. Тестирование внешний функций 15](#_Toc43706481)

[5.3. Тестирование требований технического задания 15](#_Toc43706482)

[Заключение 16](#_Toc43706483)

[Список литературы 17](#_Toc43706484)

[Приложение 1. Техническое задание 18](#_Toc43706485)

[Приложение 2. Руководство пользователя 23](#_Toc43706486)

[Приложение 3. Код программы 26](#_Toc43706487)

Введение

Для выполнения курсовой работы требуется написать программу, с использованием механизмов объектно-ориентированного программирования. Объектно-ориентированное программирование - это парадигма разработки программных систем, в которой приложения состоят из объектов, то есть сущностей, со своими свойствами и поведением. Обычно объектами являются экземпляры классов, а связанные с этим объектом данные называют свойством. Поведение объекта определяется методами, которые можно вызывать из разных частей программы.

Использование этих свойств и методов, ускоряет разработку и делает код ассоциативно понятным. Так же это помогает программисту создавать код, основываясь на объекте и его отдельных компонентов.

**1. Объектно-ориентированный анализ предметной области**

Объектно-ориентированное программирование основано на представлении программы в виде совокупности объектов, каждый из которых реализовывается в определенном классе. В свою очередь классом является ссылочный тип данных, который описывает объект - конкретный экземпляр класса, хранящийся в памяти компьютера как переменная соответствующего типа. Классы имеют поля, свойства и методы. Метод – это процедура или функция, которая работает с полями данных. Поэтому с помощью методов можно управлять действиями классов. Свойства похожи на поля, но имеют дополнительные описатели, определяющие механизмы записи и считывания данных.

Класса называют ссылочным типом данных, потому что при копировании переменной через присваивание копируется интерфейс, а не данные. Поэтому классы разрабатывают таким образом, чтобы данные соответствовали объекту и решаемой задаче. Благодаря наследованию можно спроектировать и реализовать целостность природы объекта и его интерфейс.[2]

Объектно-ориентированное программирование состоим из четырёх основных принципов: наследование, абстракция, инкапсуляция и полиморфизм. Использование каждого из этих принципов помогает в разработке программ и экономит большое количество времени.

Наследование — механизм, который позволяет описать новый класс на основе существующего. При этом свойства и функциональность родительского класса частично или полностью заимствуется новым классом.

Класс, от которого производится наследование, называется родительским или базовым, а новый класс потомком или производным классом. Основными плюсами этого принципа, является повторное использование кода и возможность перенять поля и методы, описанные в родительских классах, наследником.

Абстракция означает выделение наиболее значимых характеристик предмета и отбрасывание незначительных. В ООП рассматривается только набор наиболее значимых характеристик объекта, доступных остальной программе.

Инкапсуляция означает ограничение доступа к данным и возможностям их изменения. Так же это позволяет объединить данные и методы, для использования их в классе. Основной целью инкапсуляции является обеспечение согласованности внутреннего состояния объекта.

Полиморфизм позволяет работать с объектами, у которых одинаковый интерфейс, но разное поведение. Поведение меняется в зависимости от типа, к которому принадлежит объект. [1]

При проектировании программы возникает необходимость в предварительной разработке информационной модели предметной области - концептуальной схемы, которая отражала бы взаимосвязи между объектами предметной области и поддержания целостности данных. В этом поможет объектно-ориентированный анализ. ООА – это методология анализа предметной области, при которой требования к проектируемой системе воспринимаются с точки зрения классов и объектов.

В объектно-ориентированном анализе основное внимание уделяется созданию моделей, более близких к реальности, с использованием объектно-ориентированного подхода.

Выделяют три этапа ООА:

* Построение информационной модели, абстрагирование реальных сущностей в терминах объектов и атрибутов.
* Построение модели состояний для формализации жизненных циклов объектов и отображение этой модели диаграммами и таблицами переходов, взаимодействие между объектами осуществляется путем передачи сообщений о происходящих с ними событиях.
* Разработка модели процессов, в которой действия в моделях состояний расчленяются на фундаментальные и многократно используемые процессы.

Предметная область - это мысленно ограниченная область реальной действительности, подлежащая описанию или моделированию и исследованию. Она состоит из объектов, различаемых по свойствам и отношениям между собой. В предметной области «Программное обеспечение» можно разделить три класса:

* Системное ПО - системные программы, выполняющие различные вспомогательные функции, например, управление ресурсами компьютера, проверка работоспособности устройств компьютера;
* Прикладное ПО - прикладные программы, непосредственно обеспечивающие выполнение необходимых пользователям работ;
* Системы программирования - инструментальные программные системы, облегчающие процесс создания новых программ для компьютера. [3]

Программное обеспечение - это совокупность программ обработки данных и необходимых для их эксплуатации документов. Программное обеспечение является очень широким понятием, оно включает в себя системное программное обеспечение, которое отвечает за работоспособность компьютеров, прикладное программное обеспечение, предназначенное для решения задач любой предметной области в виде пакетов прикладных программ и инструментарий технологии программирования. [4]

Все виды программного обеспечения имеют общие черты, такие как: режим эксплуатации, функционал, требования надежности, исходный язык, вычислительная система и среда, способ использования и распространения. Все эти критерии можно объединить в класс-интерфейс, который согласует работу с основными полями всех ПО для дальнейшей полиморфной обработки. Также нужен базовый класс, который будет содержать в себе основные поля для всего программного обеспечения.

Так как ПО делиться на три класса, а каждом классе есть свои программы, то с помощью наследования можно реализовать это.

В результате проведенного объектно-ориентированного анализа была построена контекстная диаграмма классов (рис. 1)

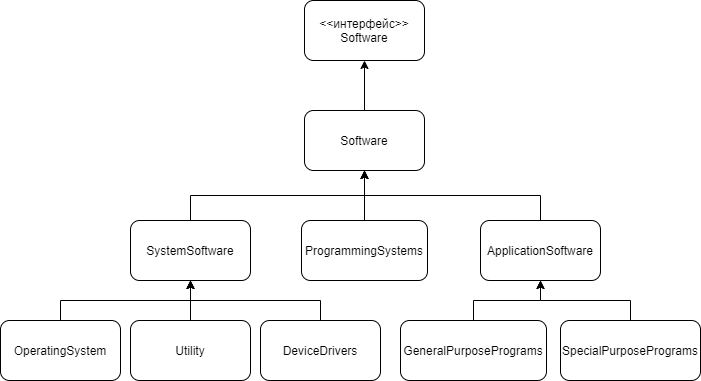


Рис. 1. Контекстная диаграмма классов

**2. Проектирование классов**

Проектирование — это процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или её части. Результатом проектирования является целостная совокупность моделей, свойств или характеристик, описанных в форме, пригодной для реализации системы. Поэтому следует установить взаимосвязи между ключевыми абстракциями, найденными в предметной области.

На основе объектно-ориентированного анализа была создана контекстная диаграмма классов из которой можно выделить:

* базовый класс «Software» (программное обеспечение), в котором были выделены следующие поля: название, версия, стоимость и размер занимаемого места программы;
* классы-наследники базового класса: «ProgrammingSystems» (системы программирования), который имеет поле «язык/среда программирования», «SystemSoftware» (системные программы) и «ApplicationSoftware» (прикладные программы) с полями «специализация».
* классы «OperatingSystem» (операционная система), «Utility» (утилита) и «DeviceDrivers» (драйвера) наследуются от класса «SystemSoftware», они имеют поля «обеспечение работы (всего/программ/устройств)». От класса «ApplicationSoftware» наследуются классы «GeneralPurposePrograms» (программы общего назначения) с полем «виды» и «SpecialPurposePrograms» (программы специального назначения), у которого есть поле «сфера применения».

Для хранения указателей на интерфейс был спроектирован обобщенный контейнерный класс и от него производный шаблонный класс. Этот класс нужен для более удобной полиморфной обработки данных. Он реализует статический вектор.

В завершение проектирования классов, становиться возможным составление диаграммы классов. Диаграмма классов - это структурная диаграмма, демонстрирующая общую структуру иерархии классов системы и взаимосвязей между ними.

**3. Логическая структура программы**

Логическая структура программы представлена на рис. 2.

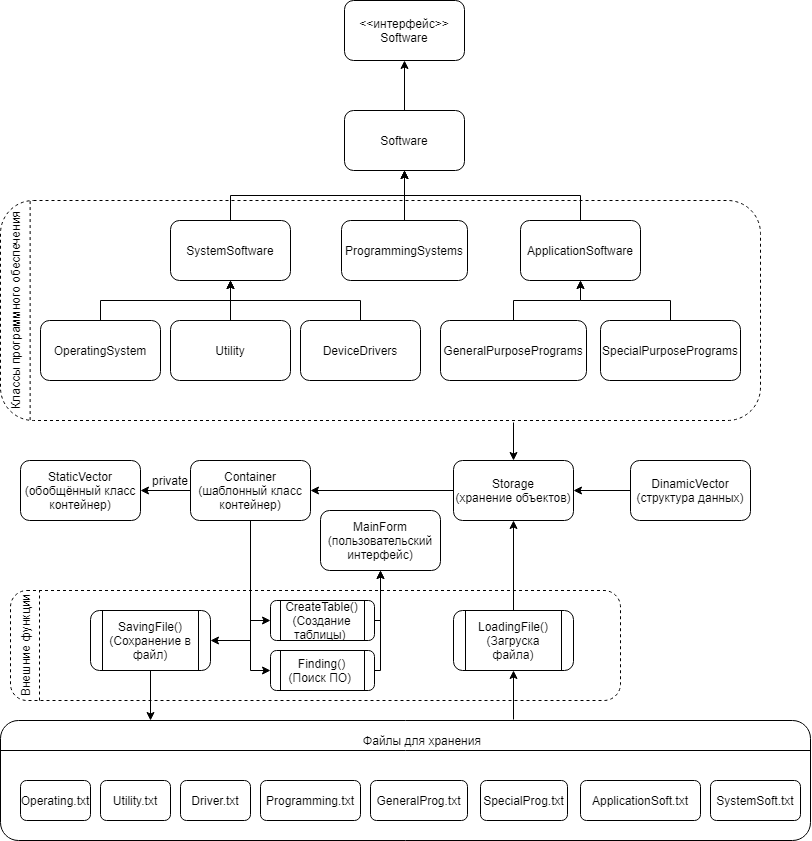


Рис. 2. Логическая структура программы.

Информация о каждом объекте хранится в файлах формата .txt. Внешняя функция LoadFromFile() загружает информацию об объектах из файлов, после чего создаётся таблица с этими данными в полях, которые помещаются в хранилище объектов(Storage) как динамический вектор. С помощью шаблонного контейнера связываются объекты. Из пользовательского интерфейса происходит взаимодействие логических модулей и функций, так же пользователь может добавлять новые объекты, сохранять их в файл и осуществлять поиск по модели.

# 4. Модульная структура программы

Модульная структура представляет собой иерархию процедур и функций, с помощью которых программа решает поставленную задачу. При этом программа является головным модулем в данной иерархии. Модульная структура программы представлена на рис.3.

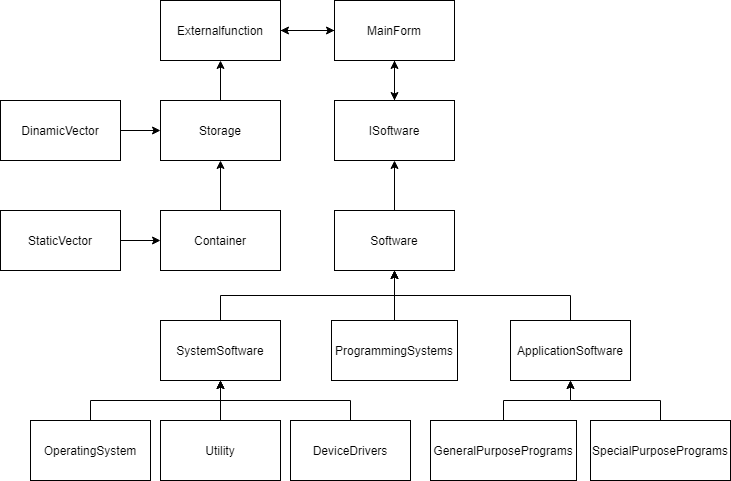


Рис.3. Модульная структура программы

Модули из схемы выше представлены в таблице 1:

Таблица 1. Модули

|  |  |  |
| --- | --- | --- |
| Наименование | Обозначение | Тип |
| MainForm.h | Интерфейс программы | Form |
| Storage.h | Реализация контейнеров для хранения объектов всех классов | Class |
| Container.h | Шаблонный класс для хранения указателей на интерфейс | Class |
| StaticVector.h | Реализация обобщённого контейнерного класса | Class |
| DinamicVector.h | Реализация класса vector для хранения объектов | Class |
| Externalfunction.h | Функции полиморфной обработки данных | Class |
| ISoftware.h | Класс-интерфейс с виртуальными методами для полиморфной обработки данных предметной области | Class |
| Software.h | Базовый класс с общими полями для всей предметной области | Class |
| SystemSoftware.h  ApplicationSoftware.h | Класс-наследник, базового класса | Class |
| ProgrammingSystems.h  OperatingSystem.h  Utility.h  DeviceDrivers.h  GeneralPurposePrograms.h  SpecialPurposePrograms.h | Нижний уровень иерархии родственных классов. Создание объектов будет происходить от этих классов | Class |

**5. Тестирование**

5.1. Тестирование классов

Тестирование классов программы представлено в таблице 2.

Таблица 2. Тестирование классов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Дата | Тестируемый класс | Тестируемая функция | Ответственное  лицо | Описание теста | Результат работы |
| 18.05.20 | Software | Software() | Тестировщик | Вызов конструктора по умолчанию | Успешно |
| 18.05.20 | Software | Software(…) | Тестировщик | Вызов конструктора с параметрами | Успешно |
| 18.05.20 | OperatingSystem | OperatingSystem () | Тестировщик | Вызов конструктора по умолчанию | Успешно |
| 18.05.20 | OperatingSystem | OperatingSystem (…) | Тестировщик | Вызов конструктора с параметрами | Успешно |
| 18.05.20 | OperatingSystem | Print | Тестировщик | Вывод информации об объекте в таблицу | Успешно |
| 18.05.20 | OperatingSystem | Saving | Тестировщик | Сохранение информации об объекте в файл | Успешно |
| 18.05.20 | OperatingSystem | Loading | Тестировщик | Загрузка из файла информации в объект | Успешно |
| 18.05.20 | Utility | Utility() | Тестировщик | Вызов конструктора по умолчанию | Успешно |
| 18.05.20 | Utility | Utility(…) | Тестировщик | Вызов конструктора с параметрами | Успешно |
| 18.05.20 | Utility | Print | Тестировщик | Вывод информации об объекте в таблицу | Успешно |
| 18.05.20 | Utility | Saving | Тестировщик | Сохранение информации об объекте в файл | Успешно |
| 18.05.20 | Utility | Loading | Тестировщик | Загрузка из файла информации в объект | Успешно |
| 18.05.20 | DeviceDrivers | DeviceDrivers() | Тестировщик | Вызов конструктора по умолчанию | Успешно |
| 18.05.20 | DeviceDrivers | DeviceDrivers (…) | Тестировщик | Вызов конструктора с параметрами | Успешно |
| 18.05.20 | DeviceDrivers | Print | Тестировщик | Вывод информации об объекте в таблицу | Успешно |
| 18.05.20 | DeviceDrivers | Saving | Тестировщик | Сохранение информации об объекте в файл | Успешно |
| 18.05.20 | DeviceDrivers | Loading | Тестировщик | Загрузка из файла информации в объект | Успешно |
| 19.05.20 | ProgrammingSystems | ProgrammingSystems() | Тестировщик | Вызов конструктора по умолчанию | Успешно |
| 19.05.20 | ProgrammingSystems | ProgrammingSystems(…) | Тестировщик | Вызов конструктора с параметрами | Успешно |
| 19.05.20 | ProgrammingSystems | Print | Тестировщик | Вывод информации об объекте в таблицу | Успешно |
| 19.05.20 | ProgrammingSystems | Saving | Тестировщик | Сохранение информации об объекте в файл | Успешно |
| 19.05.20 | ProgrammingSystems | Loading | Тестировщик | Загрузка из файла информации в объект | Успешно |
| 19.05.20 | GeneralPurposePrograms | GeneralPurposePrograms() | Тестировщик | Вызов конструктора по умолчанию | Успешно |
| 19.05.20 | GeneralPurposePrograms | GeneralPurposePrograms(…) | Тестировщик | Вызов конструктора с параметрами | Успешно |
| 19.05.20 | GeneralPurposePrograms | Print | Тестировщик | Вывод информации об объекте в таблицу | Успешно |
| 19.05.20 | GeneralPurposePrograms | Saving | Тестировщик | Сохранение информации об объекте в файл | Успешно |
| 19.05.20 | GeneralPurposePrograms | Loading | Тестировщик | Загрузка из файла информации в объект | Успешно |
| 19.05.20 | SpecialPurposePrograms | SpecialPurposePrograms() | Тестировщик | Вызов конструктора по умолчанию | Успешно |
| 19.05.20 | SpecialPurposePrograms | SpecialPurposePrograms(…) | Тестировщик | Вызов конструктора с параметрами | Успешно |
| 19.05.20 | SpecialPurposePrograms | Print | Тестировщик | Вывод информации об объекте в таблицу | Успешно |
| 19.05.20 | SpecialPurposePrograms | Saving | Тестировщик | Сохранение информации об объекте в файл | Успешно |
| 19.05.20 | SpecialPurposePrograms | Loading | Тестировщик | Загрузка из файла информации в объект | Успешно |
| 20.05.20 | Storage | Push | Тестировщик | Добавление элемента в массив | Успешно |
| 20.05.20 | Storage | Size | Тестировщик | Получение информации о размере массива | Успешно |
| 20.05.20 | Storage | Clear | Тестировщик | Очистка массива | Успешно |
| 21.05.20 | Container | Push | Тестировщик | Добавление элемента в массив | Успешно |
| 21.05.20 | Container | At | Тестировщик | Получение значения элемента массива | Успешно |
| 21.05.20 | Container | Delete | Тестировщик | Удаление элемента из массива | Успешно |
| 21.05.20 | Container | Empty | Тестировщик | Проверка на пустоту массива | Успешно |
| 21.05.20 | Container | Clear | Тестировщик | Очистка массива | Успешно |

5.2. Тестирование внешний функций

Тестирование внешних функций программы представлено в табл. 3.

Таблица 3. Тестирование внешних функций

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата | Тестируемая функция | Ответственное  лицо | Описание теста | Результат работы |
| 21.05.20 | CreateTable | Тестировщик | Создание таблицы из массива | Успешно |
| 21.05.20 | SavingFile | Тестировщик | Сохранение элементов в файл | Успешно |
| 21.05.20 | LoadingFile | Тестировщик | Загрузка файла в массив объектов | Успешно |

5.3. Тестирование требований технического задания

Тестирование требований технического задания представлено в табл. 4.

Таблица 4. Тестирование требований ТЗ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата | Тестируемая функция | Ответственное  лицо | Описание теста | Результат работы |
| 21.05.20 | Finding | Тестировщик | Поиск объекта в массиве и вывод на экран | Успешно |
| 21.05.20 | \_addButton | Тестировщик | Добавление объекта с клавиатуры | Успешно |
| 21.05.20 | MainForm.h | Тестировщик | Вызов исключительных ситуаций | Успешно |

Так же была разработана и протестирована иерархия родственных классов, реализованы и протестированы функции обработки данных, файловый ввод/вывод.

Заключение

В ходе выполнения данной курсовой работы была разработана программа, объектно-ориентированным методом по предметной области «Программное обеспечение». Для её написания были освоены новые компетенции в области объектно-ориентированного программирования и закреплены материалы, пройденные на практических и лекционных занятиях.

Разработанная программа отвечает всем требованиям технического задания, например, использование иерархии родственных типов и использование функции обработки данных.

Список литературы

1. Javarush [Электронный ресурс], статья «Принципы объектно-ориентированного программирования». URL: <https://javarush.ru/groups/posts/1966-principih-obhhektno-orientirovannogo-programmirovanija> (дата обращения: 23.05.2020)
2. Skillbox [Электронный ресурс], статья «Что такое классы и объекты». URL: <https://skillbox.ru/media/code/oop_chast_1_chto_takoe_klassy_i_obekty/> (дата обращения: 23.05.2020)
3. Клеверенс [Электронный ресурс], статья «Виды программного обеспечения». URL: <https://www.cleverence.ru/articles/auto-busines/vidy-programmnogo-obespecheniya-kakie-byvayut-tipy-klassifikatsiya-primery/> (дата обращения: 23.05.2020)
4. Википедия [Электронный ресурс], статья «Программное обеспечение». URL: <https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5> (дата обращения: 23.05.2020)

Приложение 1. Техническое задание

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное

образовательное учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт Информационных Технологий

наименование института (факультета)

Кафедра математического и программного обеспечения ЭВМ

наименование кафедры

Программирование, модуль: ООП

наименование дисциплины в соответствии с учебным планом

УТВЕРЖДАЮ

Зав. кафедрой МПО ЭВМ,

д.т.н., профессор \_\_\_\_\_\_\_\_\_\_ Ершов Е.В.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

Объектно-ориентированное программирование на языке с++

Техническое задание на курсовую работу

Листов 5

Руководитель: Ершов Е. В.

Ф.И.О преподавателя

Исполнитель:

студент 1ПИб-01-21оп

группа

Зайцев В.А.

Фамилия, Имя, Отчество

2020 год

Введение

Данная курсовая работа посвящена разработке программы на языке программирования С++. Предметная область - «Программное обеспечение вычислительной техники». Программа предназначена для подробного рассмотрения и применения механизмов объектно-ориентированного программирования, а также защиты курсовой работы.

1. Основания для разработки

Основанием для разработки данного программного продукта является задание на курсовую работу по дисциплине «Программирование», модуль: «Объектно-ориентированное программирование на языке С ++», выданное на кафедре МПО ЭВМ ИИТ ЧГУ.

Общие требования к программному обеспечению для разработки следующие:

* Разработать иерархии родственных типов, корневой класс которых класс-интерфейс, для моделирования и обработки данных предметных областей набором отложенных методов - полиморфная обработка родственных объектов (согласно вариантов А.х.х и А.у.у). Организовать взаимодействие иерархий типов через указатели на интерфейсы и виртуальные функции (отложенные методы).
* Создать обобщенный (void\*) контейнерный класс (базовый) и от него, используя закрытое наследование, производный класс – шаблон для хранения указателей на абстрактный базовый класс - интерфейс (согласно варианта В.х).
* Для хранения объектов каждого производного класса использовать структуру данных (согласно варианта С.х).
* Реализовать функции обработки данных (сортировка и поиск по выбранным полям и задаваемым диапазонам значений, другие функции, в том числе перегруженные).
* Реализовать файловый ввод/вывод и ввод данных с клавиатуры, вывод данных на дисплей.
* Предусмотреть обработку различных исключительных ситуаций.
* Работа всех функций должна быть проверена и результаты проверки оформлены протоколом тестирования.

2. Назначение разработки

Основной задачей программы является представление предметной области «Программное обеспечение вычислительной техники» и изучение объектно-ориентированного программирования на языке C++.

3. Требования к программе

3.1. Требования к функциональным характеристикам

* Запись данных в файл;
* Ввод данных с клавиатуры;
* Создание объектов классов и обмен данными между ними;
* Вывод данных из файла;
* Вывод данных и результатов на экран;
* Создание классов и демонстрация механизмов работы с ними;
* Обработка исключительных ситуаций при добавлении объектов с помощью пользовательской формы, и загрузке объектов из файлов;
* Реализация механизмов наследования;
* Создание иерархии родственных типов.

3.2. Требования к надёжности

* Выполнение функций программы без сбоев;
* Наличие обработчика исключительных ситуаций;
* Корректный вывод информации на экран.

3.3. Условия эксплуатации

Переносной накопитель хранить в сухом месте и не допускать механических воздействий.

3.4. Требования к составу и параметрам технических средств

Для правильной работы программы необходимы следующие минимальные системные требования:

• тип процессора – Pentium 4 и выше;

• операционная система Windows 7 и выше;

• оперативная память – 256 Mb и выше;

• объем жесткого диска – не менее 20 Гб;

3.5. Требования к информационной и программной совместимости

Программа будет работать только на операционной системе Windows. Исходный код программы реализуется при помощи языка C++ и встроенных в него функции на базе Visual Studio 2015. Необходимо наличие пакета .NET Framework 4.5 и выше.

3.6. Требования к маркировке и упаковке

Требования не предъявляются.

3.7. Требования к транспортировке и хранению

Для распространения и хранения приложения необходимо использовать такие носители, как диски и флэш накопители.

4. Требования к программной документации

Программная документация должна содержать расчетно-пояснительную записку с содержанием:

* Титульный лист;
* Оглавление;
* Введение;
* Основная часть;
* Заключение;
* Список литературы;
* Приложения (техническое задание, руководство пользователя, текст программы).

5. Стадии и этапы разработки

Стадии и этапы разработки представлены в таблице П1.1.

Таблица П1.1

Стадии и этапы разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапа разработки | Сроки разработки | Результат выполнения | Отметка о выполнении |
| Разработка техническое задания | 09.04.20 - 12.04.20 | Оформленное техническое задание |  |
| Анализ предметной области | 12.04.20 - 16.04.20 | Проведён анализ предметной области |  |
| Создание иерархии родственных классов | 16.04.20 - 21.04.20 | Созданная иерархия классов |  |
| Проектирование приложения | 21.04.20 - 05.05.20 | Готовое приложение |  |
| Тестирование приложения | 05.05.20 - 06.05.20 | Исправлены все ошибки |  |
| Оформление РПЗ | 06.05.20 - 20.05.20 | Оформленное РПЗ |  |

6. Порядок контроля и приёмки

Проверка контроля проходит в соответствии с графиком стадий и этапов разработки:

* сдача технического задания и иерархии родственных типов;
* обработка исключительных ситуаций и тестирование программы;
* предоставление программы в электронном виде;
* сдача расчетно-пояснительной записки.

Приложение 2. Руководство пользователя

1) Для запуска программы откройте файл «Программ.exe», который выделен на рисунке 4.



Рис. 4 Файл Программ.exe.

2) В открывшемся приложении выбираем программное обеспечение для добавления (см. рисунок 5). В ячейках (см. рисунок 6) нужно ввести параметры входных данных. После чего нажать на кнопку с надписью “Добавить” (см. рисунок 7). Которая добавит информацию в таблицу см. рисунок 8.

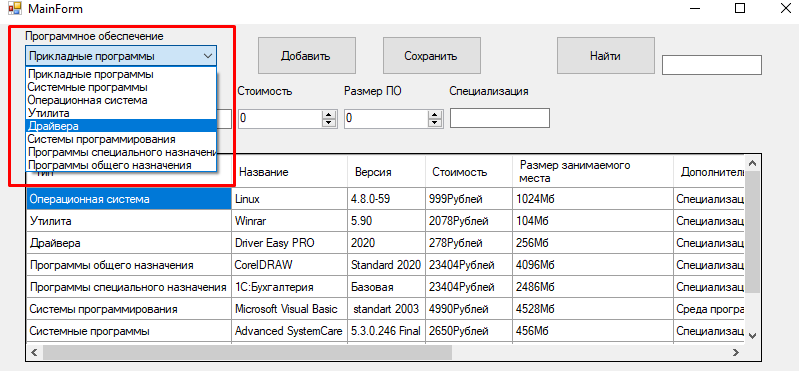


Рис. 5 Выбор программного обеспечения.

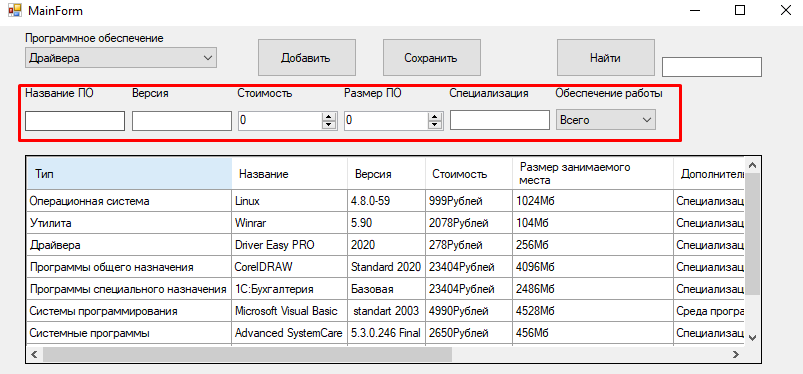


Рис. 6 Ввод параметров.

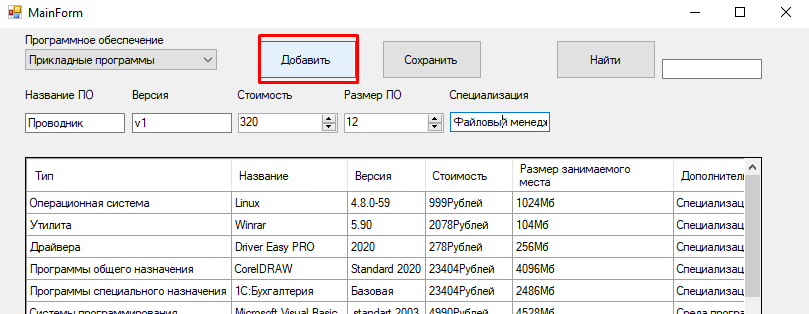


Рис. 7 Добавление ПО в таблицу.

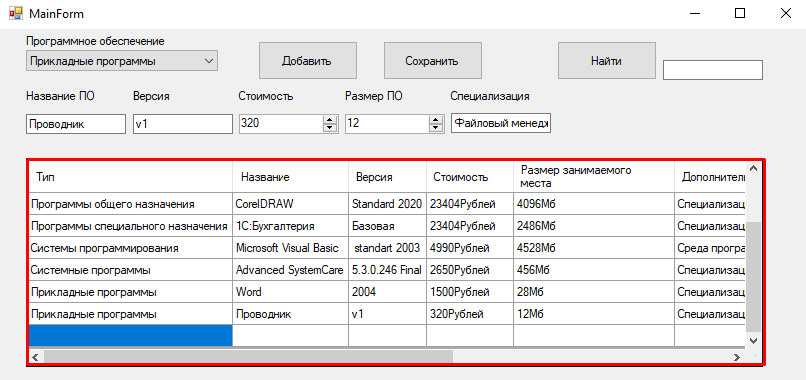


Рис. 8 Таблица с информацией о ПО.

3) В ячейку рядом с кнопкой с надписью “Найти” нужно ввести название искомого ПО. После чего нужно нажать на кнопку. Которая выделит нужную программу в таблице см. рисунок 9.

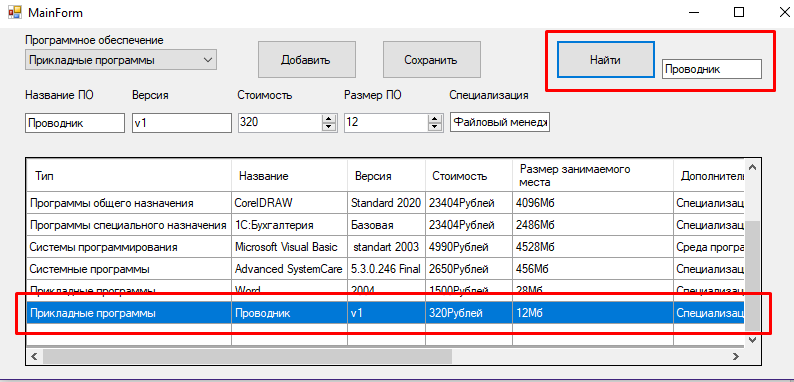


Рис. 9 Поиск.

4) Нажав на кнопку с надписью “Сохранить” в файлы сохранится информация из таблицы см. рисунок 10.

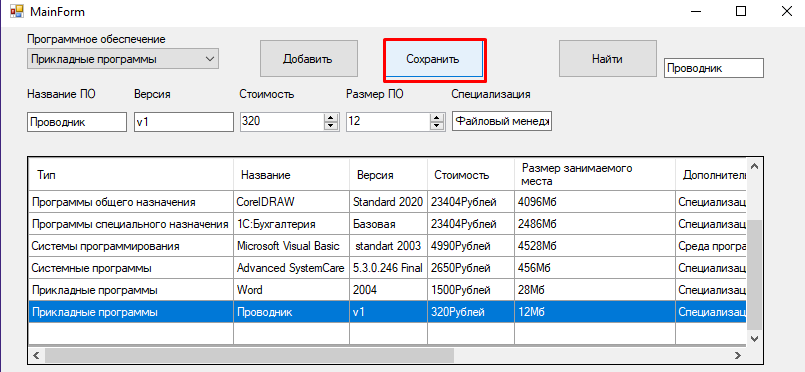


Рис. 10 Сохранение таблицы.

Приложение 3. Код программы

Модуль MainForm.cpp

#include "MainForm.h"

using namespace System;

using namespace System::Windows::Forms;

using namespace System::Threading;

using namespace System::Globalization;

[STAThreadAttribute]

int main(cli::array<String^>^ args)

{

setlocale(0, "RU");

CultureInfo^ ci = gcnew CultureInfo("en-US");

Thread::CurrentThread->CurrentCulture = ci;

Thread::CurrentThread->CurrentUICulture = ci;

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Project::MainForm form;

Application::Run(% form);

}

Модуль MainForm.h

#pragma once

#include "Externalfunction.h"

#include <iostream>

#include <vector>

#include <sstream>

namespace Project {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace System::IO;

/// <summary>

/// Сводка для MainForm

/// </summary>

public ref class MainForm : public System::Windows::Forms::Form

{

public:

MainForm(void)

{

InitializeComponent();

LoadingFile();

auto table = CreateTable();

\_table->DataSource = table;

\_table->ClearSelection();

\_softwareBox->SelectedIndex = 0;

\_systemBox->SelectedIndex = 0;

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~MainForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::ComboBox^ \_softwareBox;

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::TextBox^ text1;

private: System::Windows::Forms::NumericUpDown^ numeric1;

private: System::Windows::Forms::Label^ label2;

private: System::Windows::Forms::Label^ \_txt1;

private: System::Windows::Forms::Label^ \_txt2;

private: System::Windows::Forms::Label^ \_txt3;

private: System::Windows::Forms::Label^ \_txt4;

private: System::Windows::Forms::TextBox^ \_nameTxt;

private: System::Windows::Forms::TextBox^ \_versionTxt;

private: System::Windows::Forms::NumericUpDown^ \_costTxt;

private: System::Windows::Forms::NumericUpDown^ \_sizeTxt;

private: System::Windows::Forms::TextBox^ text2;

private: System::Windows::Forms::ComboBox^ \_systemBox;

private: System::Windows::Forms::Button^ \_addButton;

private: System::Windows::Forms::Label^ \_txt5;

private: System::Windows::Forms::Button^ \_saveButton;

private: System::Windows::Forms::TextBox^ \_findingTxt;

private: System::Windows::Forms::Button^ findButton;

private: System::Windows::Forms::DataGridView^ \_table;

private:

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

this->\_softwareBox = (gcnew System::Windows::Forms::ComboBox());

this->label1 = (gcnew System::Windows::Forms::Label());

this->text1 = (gcnew System::Windows::Forms::TextBox());

this->numeric1 = (gcnew System::Windows::Forms::NumericUpDown());

this->label2 = (gcnew System::Windows::Forms::Label());

this->\_txt1 = (gcnew System::Windows::Forms::Label());

this->\_txt2 = (gcnew System::Windows::Forms::Label());

this->\_txt3 = (gcnew System::Windows::Forms::Label());

this->\_txt4 = (gcnew System::Windows::Forms::Label());

this->\_nameTxt = (gcnew System::Windows::Forms::TextBox());

this->\_versionTxt = (gcnew System::Windows::Forms::TextBox());

this->\_costTxt = (gcnew System::Windows::Forms::NumericUpDown());

this->\_sizeTxt = (gcnew System::Windows::Forms::NumericUpDown());

this->text2 = (gcnew System::Windows::Forms::TextBox());

this->\_systemBox = (gcnew System::Windows::Forms::ComboBox());

this->\_addButton = (gcnew System::Windows::Forms::Button());

this->\_txt5 = (gcnew System::Windows::Forms::Label());

this->\_saveButton = (gcnew System::Windows::Forms::Button());

this->\_findingTxt = (gcnew System::Windows::Forms::TextBox());

this->findButton = (gcnew System::Windows::Forms::Button());

this->\_table = (gcnew System::Windows::Forms::DataGridView());

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numeric1))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->\_costTxt))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->\_sizeTxt))->BeginInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->\_table))->BeginInit();

this->SuspendLayout();

//

// \_softwareBox

//

this->\_softwareBox->DropDownStyle = System::Windows::Forms::ComboBoxStyle::DropDownList;

this->\_softwareBox->FormattingEnabled = true;

this->\_softwareBox->ImeMode = System::Windows::Forms::ImeMode::NoControl;

this->\_softwareBox->ItemHeight = 13;

this->\_softwareBox->Items->AddRange(gcnew cli::array< System::Object^ >(8) {

L"Прикладные программы", L"Cистемные программы",

L"Операционная система", L"Утилита", L"Драйвера", L"Cистемы программирования", L"Программы специального назначения", L"Программы общего назначения"

});

this->\_softwareBox->Location = System::Drawing::Point(26, 21);

this->\_softwareBox->MaximumSize = System::Drawing::Size(250, 0);

this->\_softwareBox->MaxLength = 250;

this->\_softwareBox->MinimumSize = System::Drawing::Size(170, 0);

this->\_softwareBox->Name = L"\_softwareBox";

this->\_softwareBox->Size = System::Drawing::Size(192, 21);

this->\_softwareBox->TabIndex = 1;

this->\_softwareBox->SelectedIndexChanged += gcnew System::EventHandler(this, &MainForm::\_softwareBox\_SelectedIndexChanged);

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(448, 60);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(35, 13);

this->label1->TabIndex = 2;

this->label1->Text = L"label1";

//

// text1

//

this->text1->Location = System::Drawing::Point(451, 84);

this->text1->Name = L"text1";

this->text1->Size = System::Drawing::Size(100, 20);

this->text1->TabIndex = 3;

//

// numeric1

//

this->numeric1->Location = System::Drawing::Point(451, 85);

this->numeric1->Name = L"numeric1";

this->numeric1->Size = System::Drawing::Size(100, 20);

this->numeric1->TabIndex = 4;

//

// label2

//

this->label2->AutoSize = true;

this->label2->Location = System::Drawing::Point(554, 60);

this->label2->Name = L"label2";

this->label2->Size = System::Drawing::Size(35, 13);

this->label2->TabIndex = 5;

this->label2->Text = L"label2";

//

// \_txt1

//

this->\_txt1->AutoSize = true;

this->\_txt1->Location = System::Drawing::Point(23, 60);

this->\_txt1->Name = L"\_txt1";

this->\_txt1->Size = System::Drawing::Size(76, 13);

this->\_txt1->TabIndex = 6;

this->\_txt1->Text = L"Название ПО";

//

// \_txt2

//

this->\_txt2->AutoSize = true;

this->\_txt2->Location = System::Drawing::Point(130, 60);

this->\_txt2->Name = L"\_txt2";

this->\_txt2->Size = System::Drawing::Size(44, 13);

this->\_txt2->TabIndex = 7;

this->\_txt2->Text = L"Версия";

//

// \_txt3

//

this->\_txt3->AutoSize = true;

this->\_txt3->Location = System::Drawing::Point(236, 60);

this->\_txt3->Name = L"\_txt3";

this->\_txt3->Size = System::Drawing::Size(62, 13);

this->\_txt3->TabIndex = 8;

this->\_txt3->Text = L"Стоимость";

//

// \_txt4

//

this->\_txt4->AutoSize = true;

this->\_txt4->Location = System::Drawing::Point(342, 60);

this->\_txt4->Name = L"\_txt4";

this->\_txt4->Size = System::Drawing::Size(65, 13);

this->\_txt4->TabIndex = 9;

this->\_txt4->Text = L"Размер ПО";

//

// \_nameTxt

//

this->\_nameTxt->Location = System::Drawing::Point(26, 85);

this->\_nameTxt->Name = L"\_nameTxt";

this->\_nameTxt->Size = System::Drawing::Size(100, 20);

this->\_nameTxt->TabIndex = 10;

//

// \_versionTxt

//

this->\_versionTxt->Location = System::Drawing::Point(133, 85);

this->\_versionTxt->Name = L"\_versionTxt";

this->\_versionTxt->Size = System::Drawing::Size(100, 20);

this->\_versionTxt->TabIndex = 11;

//

// \_costTxt

//

this->\_costTxt->Location = System::Drawing::Point(239, 85);

this->\_costTxt->Name = L"\_costTxt";

this->\_costTxt->Size = System::Drawing::Size(100, 20);

this->\_costTxt->TabIndex = 14;

//

// \_sizeTxt

//

this->\_sizeTxt->Location = System::Drawing::Point(345, 85);

this->\_sizeTxt->Name = L"\_sizeTxt";

this->\_sizeTxt->Size = System::Drawing::Size(100, 20);

this->\_sizeTxt->TabIndex = 15;

//

// text2

//

this->text2->Location = System::Drawing::Point(557, 84);

this->text2->Name = L"text2";

this->text2->Size = System::Drawing::Size(100, 20);

this->text2->TabIndex = 16;

//

// \_systemBox

//

this->\_systemBox->AutoCompleteCustomSource->AddRange(gcnew cli::array< System::String^ >(3) { L"Всего", L"Программ", L"Устройств" });

this->\_systemBox->DropDownStyle = System::Windows::Forms::ComboBoxStyle::DropDownList;

this->\_systemBox->FormattingEnabled = true;

this->\_systemBox->Items->AddRange(gcnew cli::array< System::Object^ >(3) { L"Всего", L"Программ", L"Устройств" });

this->\_systemBox->Location = System::Drawing::Point(557, 83);

this->\_systemBox->Name = L"\_systemBox";

this->\_systemBox->Size = System::Drawing::Size(100, 21);

this->\_systemBox->TabIndex = 17;

//

// \_addButton

//

this->\_addButton->Location = System::Drawing::Point(258, 12);

this->\_addButton->Name = L"\_addButton";

this->\_addButton->Size = System::Drawing::Size(100, 39);

this->\_addButton->TabIndex = 18;

this->\_addButton->Text = L"Добавить";

this->\_addButton->UseVisualStyleBackColor = true;

this->\_addButton->Click += gcnew System::EventHandler(this, &MainForm::\_addButton\_Click);

//

// \_txt5

//

this->\_txt5->AutoSize = true;

this->\_txt5->Location = System::Drawing::Point(23, 5);

this->\_txt5->Name = L"\_txt5";

this->\_txt5->Size = System::Drawing::Size(146, 13);

this->\_txt5->TabIndex = 19;

this->\_txt5->Text = L"Программное обеспечение";

//

// \_saveButton

//

this->\_saveButton->Location = System::Drawing::Point(383, 12);

this->\_saveButton->Name = L"\_saveButton";

this->\_saveButton->Size = System::Drawing::Size(100, 39);

this->\_saveButton->TabIndex = 20;

this->\_saveButton->Text = L"Сохранить";

this->\_saveButton->UseVisualStyleBackColor = true;

this->\_saveButton->Click += gcnew System::EventHandler(this, &MainForm::\_saveButton\_Click);

//

// \_findingTxt

//

this->\_findingTxt->Location = System::Drawing::Point(663, 31);

this->\_findingTxt->Name = L"\_findingTxt";

this->\_findingTxt->Size = System::Drawing::Size(100, 20);

this->\_findingTxt->TabIndex = 21;

//

// findButton

//

this->findButton->Location = System::Drawing::Point(557, 12);

this->findButton->Name = L"findButton";

this->findButton->Size = System::Drawing::Size(100, 39);

this->findButton->TabIndex = 22;

this->findButton->Text = L"Найти";

this->findButton->UseVisualStyleBackColor = true;

this->findButton->Click += gcnew System::EventHandler(this, &MainForm::findButton\_Click);

//

// \_table

//

this->\_table->Anchor = static\_cast<System::Windows::Forms::AnchorStyles>(((System::Windows::Forms::AnchorStyles::Top | System::Windows::Forms::AnchorStyles::Left)

| System::Windows::Forms::AnchorStyles::Right));

this->\_table->AutoSizeColumnsMode = System::Windows::Forms::DataGridViewAutoSizeColumnsMode::Fill;

this->\_table->ColumnHeadersHeightSizeMode = System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;

this->\_table->Location = System::Drawing::Point(26, 129);

this->\_table->Name = L"\_table";

this->\_table->Size = System::Drawing::Size(737, 209);

this->\_table->TabIndex = 23;

//

// MainForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(810, 350);

this->Controls->Add(this->\_table);

this->Controls->Add(this->findButton);

this->Controls->Add(this->\_findingTxt);

this->Controls->Add(this->\_saveButton);

this->Controls->Add(this->\_txt5);

this->Controls->Add(this->\_addButton);

this->Controls->Add(this->\_systemBox);

this->Controls->Add(this->text2);

this->Controls->Add(this->\_sizeTxt);

this->Controls->Add(this->\_costTxt);

this->Controls->Add(this->\_versionTxt);

this->Controls->Add(this->\_nameTxt);

this->Controls->Add(this->\_txt4);

this->Controls->Add(this->\_txt3);

this->Controls->Add(this->\_txt2);

this->Controls->Add(this->\_txt1);

this->Controls->Add(this->label2);

this->Controls->Add(this->numeric1);

this->Controls->Add(this->text1);

this->Controls->Add(this->label1);

this->Controls->Add(this->\_softwareBox);

this->Name = L"MainForm";

this->Text = L"MainForm";

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->numeric1))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->\_costTxt))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->\_sizeTxt))->EndInit();

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->\_table))->EndInit();

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private: System::Void \_softwareBox\_SelectedIndexChanged(System::Object^ sender, System::EventArgs^ e) {

numeric1->Visible = false;

text1->Visible = false;

text2->Visible = false;

label1->Visible = false;

label2->Visible = false;

\_systemBox->Visible = false;

switch (\_softwareBox->SelectedIndex)//просмотр выбранного ПО для заполнения параметров

{

case 0://Прикладные программы

label1->Visible = true;

text1->Visible = true;

label1->Text = "Специализация";

break;

case 1://Cистемные программы

label1->Visible = true;

text1->Visible = true;

label1->Text = "Специализация";

break;

case 2://Операционная система

label1->Visible = true;

label2->Visible = true;

text1->Visible = true;

label1->Text = "Специализация";

\_systemBox->Visible = true;

label2->Text = "Обеспечение работы";

break;

case 3://Утилита

label1->Visible = true;

label2->Visible = true;

text1->Visible = true;

label1->Text = "Специализация";

\_systemBox->Visible = true;

label2->Text = "Обеспечение работы";

break;

case 4://Драйвера

label1->Visible = true;

label2->Visible = true;

text1->Visible = true;

label1->Text = "Специализация";

\_systemBox->Visible = true;

label2->Text = "Обеспечение работы";

break;

case 5://Cистемы программирования

label1->Visible = true;

text1->Visible = true;

label1->Text = "Язык/Среда программирования";

break;

case 6://Программы специального назначения

label1->Visible = true;

label2->Visible = true;

text1->Visible = true;

label1->Text = "Специализация";

text2->Visible = true;

label2->Text = "Сфера применения";

break;

case 7://Программы общего назначения

label1->Visible = true;

label2->Visible = true;

text1->Visible = true;

label1->Text = "Специализация";

text2->Visible = true;

label2->Text = "Виды";

break;

default:

break;

auto table = CreateTable();

\_table->DataSource = table;

\_table->ClearSelection();

}

}

private: System::Void \_addButton\_Click(System::Object^ sender, System::EventArgs^ e) {

std::string name = "";

std::string version = "";

std::string txt = "";

std::string txt2 = "";

MarshalString(\_nameTxt->Text, name);

MarshalString(\_versionTxt->Text, version);

MarshalString(text1->Text, txt);

if (name == "" || version == "" || txt == "") {

MessageBox::Show("Не возможно добавить элемент!\n" +

"Пожалуйста, заполните все поля!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

switch (\_softwareBox->SelectedIndex)//просмотр выбранного ПО для заполнения параметров

{

case 0://Прикладные программы

Storage::ApplicationSoftwares.push\_back(\*(new ApplicationSoftware(name, version,

(int)\_costTxt->Value, (int)\_sizeTxt->Value,txt)));

break;

case 1://Cистемные программы

Storage::SystemSoftwares.push\_back(\*(new SystemSoftware(name, version,

(int)\_costTxt->Value, (int)\_sizeTxt->Value, txt)));

break;

case 2://Операционная система

MarshalString(\_systemBox->Text, txt2);

if (txt2 == "") {

MessageBox::Show("Не возможно добавить элемент!\n" +

"Пожалуйста, заполните все поля!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

Storage::OperatingSystem.push\_back(\*(new OperatingSystems(name, version,

(int)\_costTxt->Value, (int)\_sizeTxt->Value, txt, txt2)));

break;

case 3://Утилита

MarshalString(\_systemBox->Text, txt2);

if (txt2 == "") {

MessageBox::Show("Не возможно добавить элемент!\n" +

"Пожалуйста, заполните все поля!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

Storage::Utilities.push\_back(\*(new Utility(name, version,

(int)\_costTxt->Value, (int)\_sizeTxt->Value, txt, txt2)));

break;

case 4://Драйвера

MarshalString(\_systemBox->Text, txt2);

if (txt2 == "") {

MessageBox::Show("Не возможно добавить элемент!\n" +

"Пожалуйста, заполните все поля!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

Storage::DeviceDriver.push\_back(\*(new DeviceDrivers(name, version,

(int)\_costTxt->Value, (int)\_sizeTxt->Value, txt, txt2)));

break;

case 5://Cистемы программирования

Storage::ProgrammingSystem.push\_back(\*(new ProgrammingSystems(name, version,

(int)\_costTxt->Value, (int)\_sizeTxt->Value, txt)));

break;

case 6://Программы специального назначения

MarshalString(text2->Text, txt2);

if (txt2 == "") {

MessageBox::Show("Не возможно добавить элемент!\n" +

"Пожалуйста, заполните все поля!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

Storage::SpecialPurposeProgram.push\_back(\*(new SpecialPurposePrograms(name, version,

(int)\_costTxt->Value, (int)\_sizeTxt->Value, txt, txt2)));

break;

case 7://Программы общего назначения

MarshalString(text2->Text, txt2);

if (txt2 == "") {

MessageBox::Show("Не возможно добавить элемент!\n" +

"Пожалуйста, заполните все поля!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

return;

}

Storage::GeneralPurposeProgram.push\_back(\*(new GeneralPurposePrograms(name, version, (int)\_costTxt->Value, (int)\_sizeTxt->Value, txt, txt2)));

break;

auto table = CreateTable();

\_table->DataSource = table;

\_table->ClearSelection();

}

}

private: System::Void \_saveButton\_Click(System::Object^ sender, System::EventArgs^ e) {

SavingFile();

}

private: System::Void findButton\_Click(System::Object^ sender, System::EventArgs^ e) {

std::string txt = "";

MarshalString(\_findingTxt->Text, txt);

int number = Finding(txt);

\_table->ClearSelection();

if (number != -1) {

\_table->Rows[number]->Selected = true;

}

else {

MessageBox::Show("Ничего не найдено!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

};

}

Модуль ISoftware.h

#pragma once

#include <iostream>

#include <string>

#include <msclr/marshal\_cppstd.h>

using namespace System::Data;

using namespace System;

using namespace System::IO;

//using namespace std;

using namespace Runtime::InteropServices;

/\*

Интерфейс для базового класса

\*/

class ISoftware

{

public:

std::string virtual GetName() = 0; // Получение и установка названия ПО

void virtual SetName(std::string) = 0;

std::string virtual GetVersion() = 0; // Получение и установка версии ПО

void virtual SetVersion(std::string) = 0;

int virtual GetCost() = 0; // Получение и установка стоимости ПО

void virtual SetCost(int) = 0;

int virtual GetSize() = 0; // Получение и установка размера ПО

void virtual SetSize(int) = 0;

void virtual Print(DataTable^) = 0; // Вывод информации об объекте

void virtual SavingFile() = 0; //Сохранение в файл

void virtual LoadingFile(String^) = 0; //Загрузка из файла

};

Модуль Software.h

#include "Software.h"

Software::Software()

{

this->name = "";

this->version = "";

this->cost = 0;

this->size = 0;

}

Software::Software(std::string name, std::string version, int cost, int size)

{

this->name = name;

this->version = version;

this->cost = cost;

this->size = size;

}

std::string Software::GetName()

{

return this->name;

}

std::string Software::GetVersion()

{

return this->version;

}

int Software::GetCost()

{

return this->cost;

}

void Software::SetName(std::string name)

{

this->name = name;

}

void Software::SetVersion(std::string version)

{

this->version = version;

}

void Software::SetCost(int cost)

{

this->cost = cost;

}

void Software::SetSize(int size)

{

this->size = size;

}

int Software::GetSize()

{

return this->size;

}

void Software::Print(DataTable^ table)

{

DataRow^ row;

row = table->NewRow();

int indx = 0;

row[indx++] = "Неизвестно";

row[indx++] = gcnew String(GetName().c\_str());

row[indx++] = gcnew String(GetVersion().c\_str());

row[indx++] = GetCost() + "Рублей";

row[indx++] = GetSize() + "Мб";

table->Rows->Add(row);

}

Модуль DinamicVector.h

#pragma once

/\*

Класс динамический вектор

\*/

template<typename T>

class DinamicVector

{

public:

DinamicVector() {

\_size = 0; \_capacity = 2 \* \_size; \_arr = nullptr;

}

void resize(size\_t \_new\_size)

{

\_capacity = 2 \* \_new\_size;

T\* \_new\_arr = new T[\_capacity];

if(\_size > \_new\_size)\_size = \_new\_size;

for (size\_t i = 0; i != \_size; ++i)

\_new\_arr[i] = \_arr[i];

delete[] \_arr;

\_size = \_new\_size;

\_arr = \_new\_arr;

}

void push\_back(T data) {

if (\_size == \_capacity)

{

resize(\_size + 1);

--\_size;

}

\_arr[\_size++] = data;

}

void clear() {

delete[] \_arr;

\_arr = nullptr;

\_size = 0;

\_capacity = 0;

}

T\* at(size\_t index) {

if (index > size()) throw 0;

T\* temp;

temp = &\_arr[index];

return temp;

}

size\_t size() {

return \_size;

}

private:

size\_t \_size;

size\_t \_capacity;

T\* \_arr;

};

Модуль StaticVector.h

#pragma once

#include <array>

/\*

Контейнерный базовый класс, обобщённый

Для задания части B

\*/

class StaticVector

{

public:

StaticVector() {

\_statvector.fill(nullptr);

}

protected:

void \_push(void\* statvector) {

for (size\_t i = 0; i < \_statvector.size(); i++) {

if (\_statvector[i] == nullptr) {

\_statvector[i] = statvector;

break;

}

}

}

void \_deleteAt(size\_t index) {

\_statvector[index]= nullptr;

}

void \_clear() {

\_statvector.fill(nullptr);

}

bool \_empty() {

return \_statvector.empty();

}

void\* \_at(size\_t index) {

return \_statvector.at(index);

}

private:

std::array<void\*, 100> \_statvector;

};

Модуль Container.h

#pragma once

#include "StaticVector.h"

/\*

Шаблонный класс для хранения абстрактных классов

Задание части B

\*/

template<class T>

class Container: private StaticVector

{

public:

void Push(T value) {

\_push(value);

}

T At(size\_t i) {

return reinterpret\_cast<T>(\_at(i));//явное приведение указателей на неродственные типы

}

void Clear() {

\_clear();

}

bool Empty() {

return \_empty();

}

void DeleteAt(size\_t index) {

\_deleteAt(index);

}

};

Модуль Storage.h

#pragma once

#include <vector>

#include "Container.h"

#include "OperatingSystem.h"

#include "Utility.h"

#include "DinamicVector.h"

#include "DeviceDrivers.h"

#include "GeneralPurposePrograms.h"

#include "SpecialPurposePrograms.h"

#include "ProgrammingSystems.h"

#include "ApplicationSoftware.h"

#include "SystemSoftware.h"

/\*

Класс хранящий статические переменные всех контейнеров

\*/

class Storage

{

public:

// Контейнер с указателями на классы для части B

static Container<ISoftware\*> StaticContainer;

// Контейнер с объектами каждого типа для части C

static DinamicVector <OperatingSystems> OperatingSystem;

static DinamicVector <Utility> Utilities;

static DinamicVector <DeviceDrivers> DeviceDriver;

static DinamicVector <GeneralPurposePrograms> GeneralPurposeProgram;

static DinamicVector <SpecialPurposePrograms> SpecialPurposeProgram;

static DinamicVector <ProgrammingSystems> ProgrammingSystem;

static DinamicVector <SystemSoftware> SystemSoftwares;

static DinamicVector <ApplicationSoftware> ApplicationSoftwares;

};

Container<ISoftware\*> Storage::StaticContainer;

DinamicVector<OperatingSystems> Storage::OperatingSystem;

DinamicVector<Utility> Storage::Utilities;

DinamicVector<DeviceDrivers> Storage::DeviceDriver;

DinamicVector<GeneralPurposePrograms> Storage::GeneralPurposeProgram;

DinamicVector<SpecialPurposePrograms> Storage::SpecialPurposeProgram;

DinamicVector<ProgrammingSystems> Storage::ProgrammingSystem;

DinamicVector<SystemSoftware> Storage::SystemSoftwares;

DinamicVector<ApplicationSoftware> Storage::ApplicationSoftwares;

Модуль ExternalFunction.h

#pragma once

#include "Storage.h"

using namespace System::Data;

using namespace System::IO;

/\*

Запись содержимого динамических векторов в статический

\*/

template<typename T>

void InputContainer(DinamicVector<T> list) {

for (size\_t i = 0; i < list.size(); i++) {

Storage::StaticContainer.Push(list.at(i));

}

}

/\*

Генерирует таблицу запоминающих устройств для DataGridView

\*/

DataTable^ CreateTable() {

DataTable^ table = gcnew DataTable();

table->Columns->Add("Тип");

table->Columns->Add("Название");

table->Columns->Add("Версия");

table->Columns->Add("Стоимость");

table->Columns->Add("Размер занимаемого места");

table->Columns->Add("Дополнительно");

Storage::StaticContainer.Clear();

InputContainer(Storage::OperatingSystem);

InputContainer(Storage::Utilities);

InputContainer(Storage::DeviceDriver);

InputContainer(Storage::GeneralPurposeProgram);

InputContainer(Storage::SpecialPurposeProgram);

InputContainer(Storage::ProgrammingSystem);

InputContainer(Storage::SystemSoftwares);

InputContainer(Storage::ApplicationSoftwares);

/\*

Создание таблицы

\*/

for (size\_t i = 0; i < Storage::StaticContainer.Size(); i++)

{

if (Storage::StaticContainer.At(i) != nullptr) {

Storage::StaticContainer.At(i)->Print(table);

}

}

return table;

}

/\*

Поиск названия объекта в таблице

\*/

int Finding(std::string soft) {

for (size\_t i = 0; i < Storage::StaticContainer.Size(); i++) {

if (Storage::StaticContainer.At(i) != nullptr) {

if (Storage::StaticContainer.At(i)->GetName() == soft)

return i;

}

}

return -1;

}

/\*

Сохранение объектов в файл

\*/

void SavingFile() {

Storage::StaticContainer.Clear();

InputContainer(Storage::OperatingSystem);

InputContainer(Storage::Utilities);

InputContainer(Storage::DeviceDriver);

InputContainer(Storage::GeneralPurposeProgram);

InputContainer(Storage::SpecialPurposeProgram);

InputContainer(Storage::ProgrammingSystem);

InputContainer(Storage::SystemSoftwares);

InputContainer(Storage::ApplicationSoftwares);

File::Delete("Operating.txt");

File::Delete("Utility.txt");

File::Delete("Driver.txt");

File::Delete("Programming.txt");

File::Delete("GeneralProg.txt");

File::Delete("SpecialProg.txt");

File::Delete("SystemSoft.txt");

File::Delete("ApplicationSoft.txt");

for (size\_t i = 0; i < Storage::StaticContainer.Size(); i++) {

if (Storage::StaticContainer.At(i) != nullptr) {

Storage::StaticContainer.At(i)->SavingFile();

}

}

}

/\*

Загрузка объектов из файла

\*/

void LoadingFile()

{

String^ path = "Operating.txt";

StreamReader^ reader = gcnew StreamReader(path);

auto operat = std::make\_shared<OperatingSystems>();

while (!reader->EndOfStream) {

operat->LoadingFile(reader->ReadLine());

Storage::OperatingSystem.push\_back(\*operat);

}

reader->Close();

path = "Utility.txt";

reader = gcnew StreamReader(path);

auto util = std::make\_shared<Utility>();

while (!reader->EndOfStream) {

util->LoadingFile(reader->ReadLine());

Storage::Utilities.push\_back(\*util);

}

reader->Close();

path = "Driver.txt";

reader = gcnew StreamReader(path);

auto driv = std::make\_shared<DeviceDrivers>();

while (!reader->EndOfStream) {

driv->LoadingFile(reader->ReadLine());

Storage::DeviceDriver.push\_back(\*driv);

}

reader->Close();

path = "GeneralProg.txt";

reader = gcnew StreamReader(path);

auto prog = std::make\_shared<GeneralPurposePrograms>();

while (!reader->EndOfStream) {

prog->LoadingFile(reader->ReadLine());

Storage::GeneralPurposeProgram.push\_back(\*prog);

}

reader->Close();

path = "SpecialProg.txt";

reader = gcnew StreamReader(path);

auto gener = std::make\_shared<SpecialPurposePrograms>();

while (!reader->EndOfStream) {

gener->LoadingFile(reader->ReadLine());

Storage::SpecialPurposeProgram.push\_back(\*gener);

}

reader->Close();

path = "Programming.txt";

reader = gcnew StreamReader(path);

auto spec = std::make\_shared<ProgrammingSystems>();

while (!reader->EndOfStream) {

spec->LoadingFile(reader->ReadLine());

Storage::ProgrammingSystem.push\_back(\*spec);

}

reader->Close();

path = "SystemSoft.txt";

reader = gcnew StreamReader(path);

auto systs = std::make\_shared<SystemSoftware>();

while (!reader->EndOfStream) {

systs->LoadingFile(reader->ReadLine());

Storage::SystemSoftwares.push\_back(\*systs);

}

reader->Close();

path = "ApplicationSoft.txt";

reader = gcnew StreamReader(path);

auto app = std::make\_shared<ApplicationSoftware>();

while (!reader->EndOfStream) {

app->LoadingFile(reader->ReadLine());

Storage::ApplicationSoftwares.push\_back(\*app);

}

reader->Close();

}

Модуль DeviceDrivers.h

#pragma once

#include "SystemSoftware.h"

class DeviceDrivers : public SystemSoftware

{

public:

DeviceDrivers () : SystemSoftware() {

\_ensure = "Неизвестно";

}

DeviceDrivers (std::string name, std::string version, int cost, int size, std::string specialization) :

SystemSoftware(name, version, cost, size, specialization) {

\_ensure = "Неизвестно";

}

DeviceDrivers (std::string name, std::string version, int cost, int size, std::string specialization, std::string ensure) :

SystemSoftware(name, version, cost, size, specialization) {

\_ensure = ensure;

}

std::string GetDriverEnsure() {

return \_ensure;

}

void Print(DataTable^ table) override {

DataRow^ row;

row = table->NewRow();

int indx = 0;

row[indx++] = "Драйвера";

row[indx++] = gcnew String(GetName().c\_str());

row[indx++] = gcnew String(GetVersion().c\_str());

row[indx++] = GetCost() + "Рублей";

row[indx++] = GetSize() + "Мб";

row[indx++] = "Специализация: " + gcnew String(GetSystSpecialization().c\_str()) + ";Обеспечение работы: " + gcnew String(GetDriverEnsure().c\_str());

table->Rows->Add(row);

}

void LoadingFile(String^ string) override {

auto str = string->Split(';');

auto Name = str[0];

auto Version = str[1];

auto Cost = str[2];

auto Size = str[3];

auto ensure = str[4];

MarshalString(Name, name);

MarshalString(Version, version);

cost = Convert::ToInt32(Cost);

size = Convert::ToInt32(Size);

MarshalString(ensure, \_ensure);

}

void SavingFile() override {

String^ str = "";

str += gcnew String(GetName().c\_str()) + ";";

str += gcnew String(GetVersion().c\_str()) + ";";

str += GetCost() + ";";

str += GetSize() + ";";

str += gcnew String(GetSystSpecialization().c\_str()) + ";";

str += gcnew String(GetDriverEnsure().c\_str());

StreamWriter^ file = gcnew StreamWriter("Driver.txt", true);

file->WriteLine(str);

file->Close();

}

protected:

std::string \_ensure = "";

};

Модуль GeneralPurposePrograms.h

#pragma once

#include "ApplicationSoftware.h"

class GeneralPurposePrograms : public ApplicationSoftware

{

public:

GeneralPurposePrograms () : ApplicationSoftware() {

\_kind = "Неизвестно";

}

GeneralPurposePrograms (std::string name, std::string version, int cost, int size, std::string specialization) :

ApplicationSoftware(name, version, cost, size, specialization) {

\_kind = "Неизвестно";

}

GeneralPurposePrograms (std::string name, std::string version, int cost, int size, std::string specialization, std::string kind) :

ApplicationSoftware(name, version, cost, size, specialization) {

\_kind = kind;

}

std::string GetKind() {

return \_kind;

}

void Print(DataTable^ table) override {

DataRow^ row;

row = table->NewRow();

int indx = 0;

row[indx++] = "Программы общего назначения";

row[indx++] = gcnew String(GetName().c\_str());

row[indx++] = gcnew String(GetVersion().c\_str());

row[indx++] = GetCost() + "Рублей";

row[indx++] = GetSize() + "Мб";

row[indx++] = "Специализация: " + gcnew String(GetSpecialization().c\_str()) + ";Вид: " + gcnew String(GetKind().c\_str());

table->Rows->Add(row);

}

void LoadingFile(String^ string) override {

auto str = string->Split(';');

auto Name = str[0];

auto Version = str[1];

auto Cost = str[2];

auto Size = str[3];

auto kind = str[4];

MarshalString(Name, name);

MarshalString(Version, version);

cost = Convert::ToInt32(Cost);

size = Convert::ToInt32(Size);

MarshalString(kind, \_kind);

}

void SavingFile() override {

String^ str = "";

str += gcnew String(GetName().c\_str()) + ";";

str += gcnew String(GetVersion().c\_str()) + ";";

str += GetCost() + ";";

str += GetSize() + ";";

str += gcnew String(GetSpecialization().c\_str()) + ";";

str += gcnew String(GetKind().c\_str());

StreamWriter^ file = gcnew StreamWriter("GeneralProg.txt", true);

file->WriteLine(str);

file->Close();

}

protected:

std::string \_kind = "";

};

Модуль OperatingSystems.h

#include "MainForm.h"

#pragma once

#include "SystemSoftware.h"

class OperatingSystems : public SystemSoftware

{

public:

OperatingSystems() : SystemSoftware() {

\_ensure = "Неизвестно";

}

OperatingSystems(std::string name, std::string version, int cost, int size, std::string specialization) :

SystemSoftware(name, version, cost, size, specialization) {

\_ensure = "Неизвестно";

}

OperatingSystems(std::string name, std::string version, int cost, int size, std::string specialization, std::string ensure) :

SystemSoftware(name, version, cost, size, specialization) {

\_ensure = ensure;

}

std::string GetEnsure() {

return \_ensure;

}

void Print(DataTable^ table) override {

DataRow^ row;

row = table->NewRow();

int indx = 0;

row[indx++] = "Операционная система";

row[indx++] = gcnew String(GetName().c\_str());

row[indx++] = gcnew String(GetVersion().c\_str());

row[indx++] = GetCost() + "Рублей";

row[indx++] = GetSize() + "Мб";

row[indx++] = "Специализация: " + gcnew String(GetSystSpecialization().c\_str()) + ";Обеспечение работы: " + gcnew String(GetEnsure().c\_str());

table->Rows->Add(row);

}

void LoadingFile(String^ string) override {

auto str = string->Split(';');

auto Name = str[0];

auto Version = str[1];

auto Cost = str[2];

auto Size = str[3];

auto ensure = str[4];

MarshalString(Name, name);

MarshalString(Version, version);

cost = Convert::ToInt32(Cost);

size = Convert::ToInt32(Size);

MarshalString(ensure, \_ensure);

}

void SavingFile() override {

String^ str = "";

str += gcnew String(GetName().c\_str()) + ";";

str += gcnew String(GetVersion().c\_str()) + ";";

str += GetCost() + ";";

str += GetSize() + ";";

str += gcnew String(GetSystSpecialization().c\_str()) + ";";

str += gcnew String(GetEnsure().c\_str());

StreamWriter^ file = gcnew StreamWriter("Operating.txt", true);

file->WriteLine(str);

file->Close();

}

protected:

std::string \_ensure = "";

};

Модуль ProgrammingSystems.h

#pragma once

#include "Software.h"

class ProgrammingSystems : public Software

{

public:

ProgrammingSystems() : Software() {

\_environment = "Неизвестно";

}

ProgrammingSystems(std::string name, std::string version, int cost, int size) :

Software(name, version, cost, size) {

\_environment = "Неизвестно";

}

ProgrammingSystems(std::string name, std::string version, int cost, int size, std::string environment) :

Software(name, version, cost, size) {

\_environment = environment;

}

std::string GetProgEnvironment() {

return \_environment;

}

void Print(DataTable^ table) override {

DataRow^ row;

row = table->NewRow();

int indx = 0;

row[indx++] = "Cистемы программирования";

row[indx++] = gcnew String(GetName().c\_str());

row[indx++] = gcnew String(GetVersion().c\_str());

row[indx++] = GetCost() + "Рублей";

row[indx++] = GetSize() + "Мб";

row[indx++] = "Среда программирования: " + gcnew String(GetProgEnvironment().c\_str());

table->Rows->Add(row);

}

void LoadingFile(String^ string) override {

auto str = string->Split(';');

auto Name = str[0];

auto Version = str[1];

auto Cost = str[2];

auto Size = str[3];

auto Environment = str[4];

MarshalString(Name, name);

MarshalString(Version, version);

cost = Convert::ToInt32(Cost);

size = Convert::ToInt32(Size);

MarshalString(Environment, \_environment);

}

void SavingFile() override {

String^ str = "";

str += gcnew String(GetName().c\_str()) + ";";

str += gcnew String(GetVersion().c\_str()) + ";";

str += GetCost() + ";";

str += GetSize() + ";";

str += gcnew String(GetProgEnvironment().c\_str());

StreamWriter^ file = gcnew StreamWriter("Programming.txt", true);

file->WriteLine(str);

file->Close();

}

protected:

std::string \_environment = "";

};

Модуль SpecialPurposePrograms.h

#pragma once

#include "ApplicationSoftware.h"

class SpecialPurposePrograms : public ApplicationSoftware

{

public:

SpecialPurposePrograms() : ApplicationSoftware() {

\_scope = "Неизвестно";

}

SpecialPurposePrograms(std::string name, std::string version, int cost, int size, std::string specialization) :

ApplicationSoftware(name, version, cost, size, specialization) {

\_scope = "Неизвестно";

}

SpecialPurposePrograms(std::string name, std::string version, int cost, int size, std::string specialization, std::string scope) :

ApplicationSoftware(name, version, cost, size, specialization) {

\_scope = scope;

}

std::string GetScope() {

return \_scope;

}

void Print(DataTable^ table) override {

DataRow^ row;

row = table->NewRow();

int indx = 0;

row[indx++] = "Программы специального назначения";

row[indx++] = gcnew String(GetName().c\_str());

row[indx++] = gcnew String(GetVersion().c\_str());

row[indx++] = GetCost() + "Рублей";

row[indx++] = GetSize() + "Мб";

row[indx++] = "Специализация: " + gcnew String(GetSpecialization().c\_str()) + ";Cфера применения: " + gcnew String(GetScope().c\_str());

table->Rows->Add(row);

}

void LoadingFile(String^ string) override {

auto str = string->Split(';');

auto Name = str[0];

auto Version = str[1];

auto Cost = str[2];

auto Size = str[3];

auto scope = str[4];

MarshalString(Name, name);

MarshalString(Version, version);

cost = Convert::ToInt32(Cost);

size = Convert::ToInt32(Size);

MarshalString(scope, \_scope);

}

void SavingFile() override {

String^ str = "";

str += gcnew String(GetName().c\_str()) + ";";

str += gcnew String(GetVersion().c\_str()) + ";";

str += GetCost() + ";";

str += GetSize() + ";";

str += gcnew String(GetSpecialization().c\_str()) + ";";

str += gcnew String(GetScope().c\_str());

StreamWriter^ file = gcnew StreamWriter("SpecialProg.txt", true);

file->WriteLine(str);

file->Close();

}

protected:

std::string \_scope = "";

};

Модуль SystemSoftware.h

#pragma once

#include "Software.h"

class SystemSoftware : public Software

{

public:

SystemSoftware() : Software() {

\_specialization = "Неизвестно";

}

SystemSoftware(std::string name, std::string version, int cost, int size, std::string specialization) :

Software(name, version, cost, size) {

\_specialization = specialization;

}

std::string GetSystSpecialization() {

return \_specialization;

}

void Print(DataTable^ table) override {

DataRow^ row;

row = table->NewRow();

int indx = 0;

row[indx++] = "Cистемные программы";

row[indx++] = gcnew String(GetName().c\_str());

row[indx++] = gcnew String(GetVersion().c\_str());

row[indx++] = GetCost() + "Рублей";

row[indx++] = GetSize() + "Мб";

row[indx++] = "Специализация: " + gcnew String(GetSystSpecialization().c\_str());

table->Rows->Add(row);

}

void LoadingFile(String^ string) override {

auto str = string->Split(';');

auto Name = str[0];

auto Version = str[1];

auto Cost = str[2];

auto Size = str[3];

auto specialization = str[4];

MarshalString(Name, name);

MarshalString(Version, version);

cost = Convert::ToInt32(Cost);

size = Convert::ToInt32(Size);

MarshalString(specialization, \_specialization);

}

void SavingFile() override {

String^ str = "";

str += gcnew String(GetName().c\_str()) + ";";

str += gcnew String(GetVersion().c\_str()) + ";";

str += GetCost() + ";";

str += GetSize() + ";";

str += gcnew String(GetSystSpecialization().c\_str());

StreamWriter^ file = gcnew StreamWriter("SystemSoft.txt", true);

file->WriteLine(str);

file->Close();

}

protected:

std::string \_specialization = "";

};

Модуль Utility.h

#pragma once

#include "SystemSoftware.h"

class Utility : public SystemSoftware

{

public:

Utility () : SystemSoftware() {

\_ensure = "Неизвестно";

}

Utility (std::string name, std::string version, int cost, int size, std::string specialization) :

SystemSoftware(name, version, cost, size, specialization) {

\_ensure = "Неизвестно";

}

Utility (std::string name, std::string version, int cost, int size, std::string specialization, std::string ensure) :

SystemSoftware(name, version, cost, size, specialization) {

\_ensure = ensure;

}

std::string GetUtilEnsure() {

return \_ensure;

}

void Print(DataTable^ table) override {

DataRow^ row;

row = table->NewRow();

int indx = 0;

row[indx++] = "Утилита";

row[indx++] = gcnew String(GetName().c\_str());

row[indx++] = gcnew String(GetVersion().c\_str());

row[indx++] = GetCost() + "Рублей";

row[indx++] = GetSize() + "Мб";

row[indx++] = "Специализация: " + gcnew String(GetSystSpecialization().c\_str()) + ";Обеспечение работы: " + gcnew String(GetUtilEnsure().c\_str());

table->Rows->Add(row);

}

void LoadingFile(String^ string) override {

auto str = string->Split(';');

auto Name = str[0];

auto Version = str[1];

auto Cost = str[2];

auto Size = str[3];

auto ensure = str[4];

MarshalString(Name, name);

MarshalString(Version, version);

cost = Convert::ToInt32(Cost);

size = Convert::ToInt32(Size);

MarshalString(ensure, \_ensure);

}

void SavingFile() override {

String^ str = "";

str += gcnew String(GetName().c\_str()) + ";";

str += gcnew String(GetVersion().c\_str()) + ";";

str += GetCost() + ";";

str += GetSize() + ";";

str += gcnew String(GetSystSpecialization().c\_str()) + ";";

str += gcnew String(GetUtilEnsure().c\_str());

StreamWriter^ file = gcnew StreamWriter("Utility.txt", true);

file->WriteLine(str);

file->Close();

}

protected:

std::string \_ensure = "";

};