МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное

образовательное учреждение высшего образования

**«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Институт информационных технологий

Кафедра математического и программного обеспечения ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Объектно-ориентированное программирование» в

на тему Объектно-ориентированное программирование на языке С++

Выполнила студентка группы

  1ПИб-01-1оп-21

*группа*

направления подготовки

  09.03.04 Программная инженерия

*шифр, наименование*

  Пикалова А.С.

*фамилия, имя, отчество*

Руководитель

\_\_\_\_\_\_ Шаханов Никита Иванович \_

*фамилия, имя, отчество*

\_\_\_\_\_\_\_доцент \_\_\_\_\_\_\_

*должность*

Дата представления работы

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_ г.

Заключение о допуске к защите

Оценка \_\_\_\_\_\_\_\_\_\_\_\_, \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*количество баллов*

Подпись преподавателя \_\_\_\_\_\_\_\_\_ \_\_

Череповец, 2023

Оглавление

[Введение 3](#_Toc138416558)

[1. Объектно-ориентированный анализ предметной области 8](#_Toc138416559)

[2. Проектирование классов 13](#_Toc138416560)

[3. Логическая структура программы 18](#_Toc138416561)

[4. Модульная структура программы 19](#_Toc138416562)

[5. Тестирование программы 20](#_Toc138416563)

[Заключение 22](#_Toc138416564)

[Список литературы 23](#_Toc138416565)

[Приложение 1. Техническое задание 24](#_Toc138416566)

[Приложение 2. Текст программы 29](#_Toc138416567)

[Приложение 3. Руководство пользователя 40](#_Toc138416568)

[Приложение 4. Детальная диаграмма классов 43](#_Toc138416569)

# Введение

ООП создавалось, как развитие концепции процедурного программирования, с целью решения некоторых проблем, присущих процессу разработки сложных программных систем при использовании процедурного подхода к программированию и алгоритмического подхода к декомпозиции. Разумеется, процесс планирования при использовании ООП упрощается, так как объекты программы соответствуют объектам реального мира. Согласно, первые программисты писали программы посредством использования машинных кодов. В связи со сложностью подобного подхода возникла необходимость создать программу, которая преобразовывала бы написанный человеком и понятный ему текст в машинные коды. Такая программа получила название «компилятор», а используемый для написания исходного текста программы язык — «языком программирования». Первым компилятором был «Assembler». [1]

После процедурного подхода к программированию, следующей ступенью в развитии технологий программирования стало появление ООП — программный код теперь не представляется «плоским», а программисту подвластны не только процедуры и функции, но и целые классы. На сегодняшний день существуют более 2500 языков программирования высокого уровня. Это объясняется направленностью конкретных языков на определенные предметные области, а также тем, что появление новых языков дает возможность разработчикам решать все более сложные задачи.

Основная идея объектно-ориентированного подхода-объединение данных и действий, выполняемых с этими данными, в единое целое, который называется объектом. Объектные функции, называемые методами или функциями-членами. В с++, обычно они предназначены для доступа к данным объекта. При необходимости подсчета любые объектные данные, необходимо вызвать соответствующий метод, который он выполнит считывание и вернет требуемое значение. Прямой доступ к данным невозможно. Данные скрыты от внешнего влияния, что защищает их от случайного изменения. Говорят, что данные и методы инкапсулированы. Даты скрытие и инкапсуляция данных являются ключевыми в описании объектно-ориентированных языков. Если необходимо изменить данные объекта, то это действие очевидно он будет доверен методам объекта. Никакая другая функция не может изменять данные класса. Такой подход упрощает написание, отладку и использование программы.

Разработка объектно-ориентированного метода обусловлена ограничениями других методов программирования, разработанных ранее. Чтобы лучше понять и оценить значение ООП, необходимо разобраться, что это за ограничение и как оно проявляется в традиционных языках программирования.

Любая программа, использующая ООП, отражает в своих данных состояния физических предметов либо абстрактных понятий – объектов программирования, для работы с которыми она предназначена.

Все данные об объекте программирования и его связях с другими объектами можно объединить в одну структурированную переменную. В первом приближении ее можно назвать объектом.

С объектом связывается набор действий, иначе называемых методами. С точки зрения языка программирования набор действий или методов – это функции, получающие в качестве обязательного параметра указатель на объект и выполняющие определенные действия с данными объекта программирования. Технология ООП запрещает работать с объектом иначе, чем через методы, таким образом, внутренняя структура объекта закрыта от внешнего пользователя.

В ООП существует две ключевые концепции —объекты и классы.

Объект – это сущность, способная сохранять свое состояние (информацию) и обеспечивающая набор операций (поведение) для проверки и изменения этого состояния.

Класс – своего рода форма, определяющая, какие данные и функции будут включены в объект класса. При объявлении класса не создаются никакие объекты этого класса, по аналогии с тем, что существование типа int еще не означает существование переменных этого типа. [2]

Класс содержит константы и переменные, называемые полями, а также выполняемые над ними операции и функции. Функции класса называются методами. Предполагается, что доступ к полям класса возможен только через вызов соответствующих методов. Поля и методы являются элементами (членами) класса. Методы, расположенные в открытой части, формируют интерфейс класса и могут свободно вызываться клиентом через соответствующий объект класса. Доступ к закрытой секции класса возможен только из его собственных методов, а к защищенной — из его собственных методов, а также из методов классов-потомков.

Программирование «от класса к классу» включает в себя ряд новых понятий. Основными понятиями ООП являются:

* инкапсуляция;
* наследование;
* полиморфизм.

Инкапсуляция – это процесс отделения друг от друга элементов объекта, определяющих его устройство и поведение; инкапсуляция служит для того, чтобы изолировать контрактные обязательства абстракции от их реализации. Важность инкапсуляции, т.е. заключения элементов в защитную оболочку, резко возрастает с ростом размеров программы и увеличивающимся разбросом областей приложения. Инкапсуляция – это «свойство, благодаря которому разработчику, использующему определенный строительный блок (код), не нужно знать, как он на самом деле реализован и работает для корректного использования этого строительного блока». Другое преимущество инкапсуляции заключается в том, что разработанную программу легче модифицировать, поскольку при сохранении интерфейса класса можно менять его реализацию, и это не затронет внешний программный код (код клиента). [3]

Наследование — определение объекта и его дальнейшее использование для построения иерархии порожденных объектов с возможностью для каждого порожденного объекта, относящегося к иерархии, доступа к коду и данным всех порождающих объектов. Наследственность в ООП — это его основа. Следует понимать, что потомок — класс, от которого происходят другие классы, наследует свойства предка — класса, который происходит либо порожден из другого класса. Следовательно, потомок всегда «знает», какими он обладает свойствами, а предок не может «знать» свойства своего потомка, поскольку не может «знать» те свойства, которые будут добавлены в новый класс.

Полиморфизм – присваивание действию одного имени, которое затем совместно используется вниз и вверх по иерархии объектов, причем каждый объект иерархии выполняет это действие способом, именно ему подходящим. «Полиморфизм» представляет собой способность обладать несколькими формами. В объектно-ориентированной разработке это относится к сущностям (элементам структур данных), способным в процессе выполнения присоединяться к объектам разных типов. Полиморфизм дает возможность применения одних и тех же функций для решения различных задач и находит свое отражение в том, что под одним именем содержатся разные действия, наполнение которых зависит от типа объекта. Производные одного базового класса считаются полиморфными. Это подразумевает, что им присущи как общие характеристики, так и собственные свойства.

Абстрактный класс – это класс, который может использоваться лишь в качестве базового класса для производных классов. Класс является абстрактным, если он содержит хотя бы одну абстрактную функцию.

Из всех объектно-ориентированных языков C++ является наиболее широкоупотребительным. Язык Java, представляющий собой последнюю разработку в области объектно-ориентированных языков, лишен таких составляющих, как указатели, шаблоны и множественное наследование, что сделало его менее мощным и гибким по сравнению с C++.

Целью данной курсовой работы является создания программного продукта, предназначенного для работы с объектами предметной области " Программное обеспечение вычислительной техники ". В ходе разработки были использованы принципы и возможности объектно-ориентированного программирования. При проектировании была создана иерархия родственных классов предметной области и были использованы механизмы наследования и принципы инкапсуляции, произведена полиморфная обработка объектов данных классов.

В расчетно-пояснительной записке описано функциональное назначение данного программного продукта, его логическая структура, а также принцип работы программы.

# 1. Объектно-ориентированный анализ предметной области

Цель курсовой работы: создание программы, которая будет содержать иерархию родственных типов, а также контейнерные классы для их хранения.

В ходе выполнения курсовой работы необходимо разработать иерархию родственных типов, относящихся к области программного обеспечения.

Наследование – механизм, который позволяет описать новый класс (дочерний) на основе существующего (родительского). Свойства и функциональность родительского класса заимствуются новым классом. При этом каждый производный класс тоже может стать базовым.

Такое множество связанных между собой классов называют иерархией классов. Обычно она представляется деревом, но бывают иерархии с более общей структурой в виде графа.

Суперклассом в нашей иерархии является класс-интерфейс, абстрактный базовый класс (класс, от которого пойдут все остальные).

Класс-интерфейс – представляет собой не более чем просто именованный набор абстрактных членов (функций). Абстрактные методы являются чистым протоколом, поскольку не имеют никакой стандартной реализации.

Виртуальная функция – это функция-член, которую предполагается переопределить в производных классах.

Для создания иерархии родственных классов была взята предметная область, предписанная индивидуальным вариантом, а именно «Программное обеспечение вычислительной техники». [4]

Программное обеспечение (ПО) – это совокупность программных и документальных средств для создания и эксплуатации систем обработки данных средствами вычислительной техники. В зависимости от функций, выполняемых программным обеспечением, его можно разделить на:

1.системное ПО (базовое программное обеспечение);

2.прикладное ПО;

3.инструментальное ПО.

В ходе работе были выбраны такие области, как системное ПО и Прикладное ПО, т.к. они наиболее популярные представители этой области.

Системное ПО – программы, управляющие работой компьютера и выполняющие различные вспомогательные функции, например, управление ресурсами компьютера, создание копий информации, проверка работоспособности устройств компьютера, выдача справочной информации о компьютере и др. Они предназначены для всех категорий пользователей, используются для эффективной работы компьютера и пользователя, а также эффективного выполнения прикладных программ.

К прикладному ПО относятся программы, предназначенные для решения задач в различных сферах деятельности человека (бухгалтерские программы, текстовые и графические редакторы, базы данных, экспертные системы, переводчики, энциклопедии, обучающие, тестовые и игровые программы и т.д.).

Выделяют следующие программы этих областей:

Операционная система (ОС) — системный комплекс взаимосвязанных программ, который служит посредником при организации диалога пользователя с компьютером, управляет распределением и использованием компьютерных ресурсов, руководит работой всех аппаратных средств компьютера.

Утилиты — программы вспомогательного назначения, обеспечивающие дополнительный сервис (форматирование дискет, восстановление ошибочно удаленных файлов, дефрагментация файлов на диске и т. п.). Часть утилит входит в состав операционной системы, а другая часть функционирует независимо от неё, т.е. автономно). Утилиты могут распространятся как по одиночке, так и в составе больших и мощных утилитных комплексов, например, Norton Utilities.

Графические редакторы — программы, предназначенные для создания и обработки графических изображений. В данном классе различают растровые редакторы, векторные редакторы и программные средства для создания и обработки трехмерной графики.

Редактор текстов — программа, предназначенная для создания, редактирования, вывода на экран и печать, а также сохранения в виде файлов различного рода документов и данных. Текстовые редакторы различаются по своему назначению, сложности или способа оформления и, соответственно, их функциональным возможностям. По второму признаку, в частности, можно выделить встроенные текстовые редакторы, которые являются составной частью других программных продуктов, например, систем программирования. Более развитые текстовые редакторы, например, Лексикон 5.0, Microsoft Word, Microsoft Word for Windows и др., представляют собой пакеты прикладных программ, которые наряду с перечисленными выше операциями позволяют производить форматирование текста, формировать различные стили оформления документов, пользоваться большим количеством шрифтов, выделять участки текста, набирать текст в виде колонок, включать в тексты иллюстрации, формировать различного рода указатели и ссылки, вводить верхние и нижние колонтитулы страниц, производить автоматизированный поиск элементов текста и исправление ошибок, копировать и переносить в другой документ любые участки текста а также многое другое, что делает их близкими к настольным издательским системам и т.д.

Класс-интерфейс реализуется классом «программное обеспечение», у которого будет 2 класса наследника это «прикладное ПО» и «Системное ПО». Каждый из классов наследников также будет обладать наследниками. Для класса «прикладное ПО» это классы «Редактор текстов» и «Графические редакторы», а для класса «Системное ПО» это классы «Утилиты» и «Операционная система».

Также необходимо создать обобщенный (**void\***) контейнерный класс. Контейнерный класс – это класс, предназначенный для хранения и организации нескольких экземпляров другого типа (либо другого класса, либо базового типа). В нашем случае такой класс будет хранить указатель типа void, также известный как обобщенный указатель. Представляет собой специальный тип указателя, который может указывать на объекты любого типа данных, а значит все наши классы, которые являются разными типами данных, можно хранить в одном контейнере приведя их тип к void\*. Таким классом является Void\_Dynamic\_Vector. Однако для выполнения преобразования из void\* к типу класса-интерфейса и наоборот будет использоваться его наследник класс Dynamic\_Vector.

Для хранения объектов каждого производного от программного обеспечения и их сортировки будет использоваться стек. Реализовано в классе Stack и имеет ассоциативную связь с классом-интерфейсом.

В данной работе установки были классифицированы по принципу их функций, разделение представлено на контекстной диаграмме классов(рис.1).

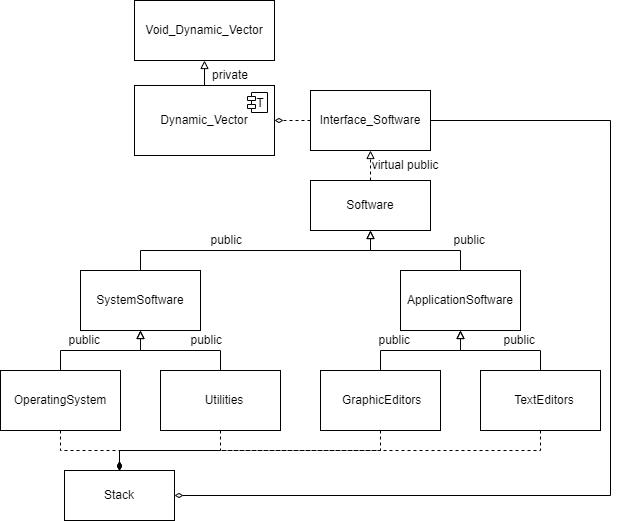


Рис. 1. Диаграмма классов

Класс-интерфейс реализуется классом «программное обеспечение», у которого будет 2 класса наследника это «прикладное ПО» и «Системное ПО». Каждый из классов наследников также будет обладать наследниками. Для класса «прикладное ПО» это классы «Редактор текстов» и «Графические редакторы», а для класса «Системное ПО» это классы «Утилиты» и «Операционная система».

Также необходимо создать обобщенный (**void\***) контейнерный класс. Контейнерный класс – это класс, предназначенный для хранения и организации нескольких экземпляров другого типа (либо другого класса, либо базового типа). В нашем случае такой класс будет хранить указатель типа void, также известный как обобщенный указатель. Представляет собой специальный тип указателя, который может указывать на объекты любого типа данных, а значит все наши классы, которые являются разными типами данных, можно хранить в одном контейнере приведя их тип к void\*. Таким классом является Void\_Dynamic\_Vector. Однако для выполнения преобразования из void\* к типу класса-интерфейса и наоборот будет использоваться его наследник класс Dynamic\_Vector.

Для хранения объектов каждого производного от программного обеспечения и их сортировки будет использоваться стек. Реализовано в классе Stack и имеет ассоциативную связь с классом-интерфейсом.

# 2. Проектирование классов

Собственно проектирование классов предполагает окончательное определение структуры и поведения его объектов. Структура объектов определяется совокупностью атрибутов и операций класса. Каждый атрибут — это поле или совокупность полей данных, содержащихся в объекте класса.

Поведение объектов класса определяется реализуемыми обязанностями. Обязанности выполняются посредством операций класса. Каждый класс должен обладать именем, и опционально атрибутами и операциями класса для обработки атрибутов, которые он содержит.

Атрибуты класса - содержательная характеристика класса, описывающая множество значений, которые могут принимать отдельные объекты этого класса. Класс может иметь любое число атрибутов или не иметь ни одного. На практике для именования атрибута используются одно или несколько коротких существительных, выражающих некое свойство класса, к которому относится атрибут. [5]

Ответственностью класса называют краткое неформальное перечисление основных функций объектов класса. Ответственность класса обычно определяют на начальных этапах проектирования, когда атрибуты и операции класса еще не определены.

Операция – это реализация метода класса. Класс может иметь любое число операций либо не иметь ни одной. Графически операции представлены в нижнем блоке описания класса. Допускается указание только имен операций. Абстрактные методы класса обозначаются курсивным шрифтом.

Достаточно существенную информацию о действиях, которые должны. выполняться методами класса, можно получить, анализируя диаграммы последовательности действий. Однако алгоритмы всех сколько-нибудь сложных методов необходимо проработать детально. При этом можно использовать как уже известные нотации, так и диаграммы деятельностей.

Класс «Interface\_Software» является интерфейсом, в нем инициализированы виртуальные функции print, FileOutput и get\_price, которые будут переопределяться в каждом классе наследнике.

Был выделен базовый класс «Software» (Программное обеспечение), в котором были выделены следующие поля: name (название), device (вид платформы), price (цена), year (год выпуска), manufacturer (производитель).

От базового класса были выделены 2 класса-наследника: «SystemSoftware» (системное программное обеспечение), и «Аpplicationsoftware» (прикладное программное обеспечение). Класс «SystemSoftware» имеет поля: management (назначение управление), limited (ограниченный доступ). Класс «Аpplicationsoftware» имеет поля: creature (назначение – создание), free (свободный доступ).

От класса системное программное обеспечение был спроектирован класс «OperatingSystem» (операционные системы), он имеет поле «functioning\_and\_access» (назначение – доступ и функциональность) и OS (тип операционной системы), и класс «Utilities» (утилиты), он имеет поле «checking\_and\_setting\_up» (проверка и настройка) и Uti (тип утилиты). В свою очередь от класса прикладное программное обеспечение был спроектирован класс «TextEditor» (текстовый редактор), у которого есть поле «format1» (формат), «tip\_info» (тип информации) и «type1» (тип представления информации), класс «GraphicEditor» (графический редактор), который имеет поле «format2» (формат), «tip\_info» (тип информации) и «type2» (тип представления информации).

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 динамический вектор. Для хранения объектов производных классов был спроектирован класс стека.

После проектирования классов и их компонентов, можно составить диаграмму классов. Диаграмма классов на унифицированном языке моделирования (UML) — это диаграмма статической структуры, которая описывает структуру системы, показывая ее классы, их атрибуты, операции (или методы) и отношения между объектами. Диаграмма классов — это план системы или подсистемы. Диаграммы классов можно использовать для моделирования объектов, составляющих систему, для демонстрации отношений между объектами и для описания ролей этих объектов и предоставляемых ими услуг. В проектах объектно-ориентированного программного обеспечения диаграмма классов, созданная на ранних стадиях проекта, содержит классы, которые часто преобразуются в реальные классы и объекты программного обеспечения при написании кода. [6]

Графически класс изображается в виде прямоугольника, разделенного на 3 блока горизонтальными линиями.

В объектно-ориентированном дизайне есть нотация видимости атрибутов и операций. UML определяет четыре типа видимости: public, protected, private и package.

Символы +, -, # и ~ перед именем атрибута и операции в классе обозначают видимость атрибута и операции:

* + обозначает общедоступные атрибуты или операции;
* – обозначает частные атрибуты или операции;
* # обозначает защищенные атрибуты или операции;
* ~ обозначает атрибуты пакета или операции.

Помимо хранения данных, классы могут содержать и функции. Функции, определенные внутри класса, называются методами. C++ требует, чтобы каждый метод структуры или класса был упомянут в определении этой структуры или класса. Методы описаны в Таблице 1.

Таблица 1

Описание методов

|  |  |  |
| --- | --- | --- |
| Название метода | Класс | Функционал |
| getCountingByIndex | Stack | Чтение элемента |
| empty | Stack | Проверка на пустоту |
| pop | Stack | Удаление элемента |
| push | Stack | Добавление элементов |
| find\_by\_price | Stack | Вывод больше цены 5000 |
| sorting | Stack | Сортировка по возрастанию |
| get\_count | Stack | Размер списка |
| pop | Dynamic\_Vector <T> | Удаление элемента списка |
| empty | Dynamic\_Vector <T> | Проверка на путоту списка |
| Size | Dynamic\_Vector <T> | Получить размер списка |
| Push | Dynamic\_Vector <T> | Добавление узла в список |
| print | Interface\_Software | Объявление функции вывода |
| FileOutput | Interface\_Software | Объявление загрузки данных об объекте из файла |
| get\_price | Interface\_Software | Объявление функции получения цены |
| FileOutput | Software | Определение загрузки данных об объекте из файла |
| GetProperties | Software | Получить информацию |
| print | Software | Функция вывода информации |
| get\_manufacturer | Software | Получить производителя |
| get\_price | Software | Получить цену |
| get\_year | Software | Получить год выпуска |
| get\_Name | Software | Получить название пр. обеспечения |
| get\_device | Software | Получить тип платформы |
| print | SystemSoftware | Вызов функции базового класса |
| GetProperties | SystemSoftware | Получить информацию |
| get\_management | SystemSoftware | Получить назначение |
| get\_limited | SystemSoftware | Получить тип доступа |
| GetProperties | ApplicationSoftware | Получение всей информации об объекте |
| print | ApplicationSoftware | Вызов функции базового класса |
| get\_creature | ApplicationSoftware | Получить назначение |
| get\_free | ApplicationSoftware | Получить тип доступа |
| print\_name | OperatingSystem | Установить вид пр. обеспечения |
| print | OperatingSystem | Вызов функции базового класса |
| GetProperties | OperatingSystem | Получение всей информации об объекте |
| get\_functioning\_  and\_access | OperatingSystem | Установить назначение |
| print\_name | Utilities | Установить вид пр. обеспечения |
| print | Utilities | Вызов функции базового класса |
| GetProperties | Utilities | Получение всей информации об объекте |
| get\_checking\_and \_setting\_up | Utilities | Установить назначение |
| print\_name | TextEditors | Установить вид пр. обеспечения |
| print | TextEditors | Вызов функции базового класса |
| GetProperties | TextEditors | Получение всей информации об объекте |
| get\_tip\_info | TextEditors | Установить тип информации |
| print\_name | GraphicEditors | Установить вид пр. обеспечения |
| print | GraphicEditors | Вызов функции базового класса |
| GetProperties | GraphicEditors | Получение всей информации об объекте |
| get\_tip\_info | GraphicEditors | Установить тип информации |

После разработки контекстной диаграммы родственных классов (рис.1) была создана Детальная диаграмма классов, полученная в ходе проектирования с учётом контейнеров, представлена в приложении 4.

# 3. Логическая структура программы

Логическая структура программы представлена на Рис. 2.

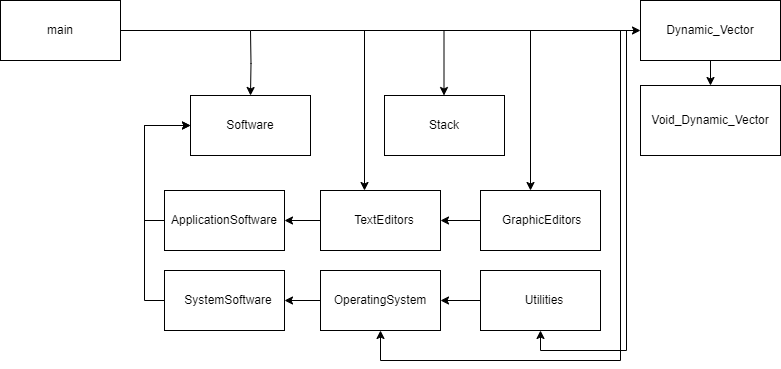


Рис. 2. Логическая структура программы

Сначала вводятся свойства объекта, которые затем сохраняются. При сохранении эти объекты заносятся в динамический вектор. Затем объекты из стека связываются с соответствующими указателями в шаблонном контейнере.

Блок main может вызвать любой из блоков, кроме блока Void\_Dynamic\_Vector, так как его вызов осуществляется только через блок Dynamic\_Vector. Также Stack не может вызвать другие блоки.

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 динамический вектор.

Данные о каждом объекте хранятся в файле input.txt, загрузка из файлов осуществляется посредством функции FileOutput ().

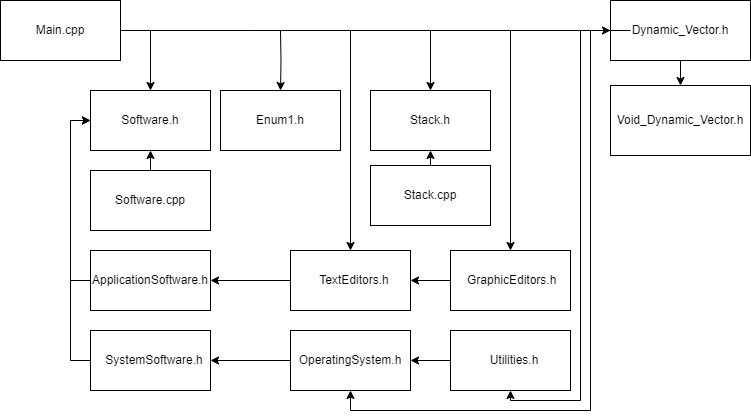
# 4. Модульная структура программы

Для представленной выше логической структуры программы были разработаны следующие модули (табл. 2).

Таблица 2

Модули программы

|  |  |
| --- | --- |
| Наименование | Обозначение |
| Main.h | Интерфейс программы |
| Software.h | Базовый класс с общими полями для всей предметной области будущих дочерних классов |
| Software.cpp | Реализация базового класса |
| SystemSoftware.h | Класс-наследник, базового класса Программное обеспечение, «Системное программное обеспечение» |
| ApplicationSoftware.h | Класс-наследник, базового класса Программное обеспечение, «Прикладное программное обеспечение» |
| OperatingSystem.h | Класс-наследник «Операционная система» класса «Системное программное обеспечение» |
| Utilities.h | Класс-наследник «Утилиты» класса «Системное программное обеспечение» |
| TextEditors.h | Класс-наследник «Текстовый редактор» класса «Прикладное программное обеспечение» |
| GraphicEditors.h | Класс-наследник «Графический редактор» класса «Прикладное программное обеспечение» |
| Void\_Dynamic\_Vector.h | Класс для хранения указателей |
| Stack.h | Класс контейнер для объектов, представляющий собой стек |
| Dynamic\_Vector.h | Шаблонный класс-наследник класса для хранения указателей на интерфейс |
| Enum1.h | Модуль, содержащий в себе наборы «перечислителей» |

  
Рис. 3. Модульная структура

# 5. Тестирование программы

В приведенных ниже таблицах представлены результаты тестирования. В таблице 3 тестирование классов, во 4 таблице тестирование внешних функций, а в таблице 5 тестирование всей программы.

Таблица 3

Протокол тестирования классов

| Дата тестирования | Класс | Кто проводил тестирование | Описание теста | Результаты тестирования |
| --- | --- | --- | --- | --- |
| 15.05.22 | OperatingSystem | Пикалова А.С. | Создание объекта класса | Успех |
| Тест метода print |
| Тест метода FileOutput |
| 15.05.22 | Utilities | Пикалова А.С. | Создание объекта класса | Успех |
| Тест метода print |
| Тест метода FileOutput |
| 26.05.22 | TextEditors | Пикалова А.С. | Создание объекта класса | Успех |
| Тест метода print |
| Тест метода FileOutput |
| 26.05.22 | GraphicEditors | Пикалова А.С. | Создание объекта класса | Успех |
| Тест метода print |
| Тест метода FileOutput |
| 02.06.22 | Void\_Dynamic\_Vector | Пикалова А.С. | Вывод | Успех |
| Запись |
| Чтение |
| 03.06.22 | Stack | Пикалова А.С. | Добавление в конец | Успех |
| Удаление из начала |
| Вставка по индексу |
| Тест print |
| Тест FileOutput |

Таблица 4

Протокол тестирования внешних функций

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата тестирования | Функция | Кто проводил тестирование | Описание теста | Результаты тестирования |
| 29.05.22 | main | Пикалова А.С. | Сортировка данных | Успех |
| Заполнение вектора и стека | Успех |
| Вывод данных | Успех |
| Запись данных | Успех |
| Тест вызова функций классов | Успех |

Таблица 5

Протокол тестирования по техническому заданию

|  |  |  |  |
| --- | --- | --- | --- |
| Дата тестирования | Тестируемый модуль | Описание теста | Результаты тестирования |
| 29.05.22 | Project140423.sln | Проверка на правильность заполнения данными вручную | Непредусмотренный ввод в файл |
| 29.05.22 | Project140423.sln | Проверка на правильность заполнения данными вручную | Успех |
| 03.06.22 | Project140423.sln | Проверка на правильность заполнения стека объектами | Успех |
| 03.06.22 | Project140423.sln | Проверка на правильность сортировки | Успех |
| 03.06.22 | Project140423.sln | Проверка всей программы на конкретную работу | Успех |

# Заключение

В результате выполнения данной курсовой работы была написана программа, со своей реализацией динамического вектора с возможностью проводить обработку данных, хранимых в стеке, реализована возможность введения данных из файла или вручную. Инструкция по работе с программой написана в приложении 3.

# Список литературы

|  |  |
| --- | --- |
| [1] | Ершов, Методика и организация самостоятельной работы: учебное пособие / Ершов Е.В., Виноградова Л.Н., Селивановских В.В., Череповец: – ЧГУ,[Книга], 2015. |
| [2] | « Информация о виртуальных функциях». *https://docs.microsoft.com/ru-ru/cpp/cpp/virtual-functions. [Интернет].* |
| [3] | Радченко Г.И, Захаров Е.А., Объектно-ориентированное программирование, Челябинск: Издательский центр ЮУрГУ, 2013. |
| [4] | Р. Лафоре, Объектно-Ориентированное Программирование в С++., Питер, 2004. |
| [5] | П. А., Объектно-ориентированное программирование на C++, СПб: «Невский диалект», 2001. |
| [6] | vpleanda, «Полное руководство по диаграмме классов UML,» *Кибермедиана,* p. 1, 10 март 2022. |

Приложение 1. Техническое задание

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное

Образовательное учреждение высшего образования

«ЧЕРЕПОВЕЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

                              Институт информационных технологий

*наименование института (факультета)*

                   Математическое и программное обеспечение ЭВМ

*наименование кафедры*

                Объектно-ориентированное программирование

*наименование дисциплины в соответствии с учебным планом*

УТВЕРЖДАЮ

Зав. кафедрой                  ,

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

«      »                       2022 г.

Объектно-ориентированное программирование на языке С++

Техническое задание на курсовой проект

Листов 5

Руководитель   Шаханов. Н.И.

*ФИО преподавателя*

Исполнитель

студент     1ПИб-01-1оп-21

*группа*

Пикалова Анастасия Сергеевна

*Фамилия, имя, отчество*

2023 год

Введение

Курсовая работа на тему: объектно-ориентированное программирование на языке С++. Закрепить способы создания производного класса и особенности работы с ним, правила инициализации и доступа к элементам производного класса, приобрести практические навыки наследования.

1. Основания для разработки

Основанием для разработки является задание на курсовую работу ООП\_2 по дисциплине "Объектно-ориентированное программирование", выданное на кафедре МПО ЭВМ ИИТ ЧГУ.

Дата утверждения: 13 февраля 2023 года.

Наименование темы разработки: объектно-ориентированное программирование на языке С++.

2. Назначение разработки

Основной задачей курсовой работы является создание программы с разработкой иерархии родственных типов, корневой класс которой абстрактный базовый класс.

3. Требования к программе

3.1 Требования к функциональным характеристикам

Программа должна содержать следующие параметрам:

* Разработать иерархию родственных типов, корневой класс которой абстрактный базовый класс (класс-интерфейс), для моделирования и обработки данных предметной области набором отложенных методов - полиморфная обработка родственных объектов (согласно варианту А1. Программное обеспечение вычислительной техники);
* Создать обобщенный (void**\***) контейнерный класс (базовый) и от него, используя закрытое наследование, производный класс – шаблон для хранения указателей на абстрактный базовый класс-интерфейс (согласно варианту В 2. Динамический вектор);
* Для хранения объектов каждого производного класса использовать структуру данных (согласно варианту С 7. Стек);
* 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.

3.2. Требования к надёжности

Программа должна:

* Предусматривать обработку различных исключительных ситуаций;
* Работа всех функций должна быть проверена и результаты проверки оформлены протоколом тестирования.

3.3. Условия эксплуатации

Программа должна исполняться на исправном ПК и новой версией обработчика кода.

3.4. Требования к составу и параметрам технических средств

Минимальные системные требования:

* Процессор: не менее 2,6 ГГц или SoC;
* ОЗУ: 1 ГБ для 32-разрядной системы или 2 ГБ для 64-разрядной системы;
* Место на жестком диске:16 ГБ для 32-разрядной ОС или 20 ГБ для 64-разрядной ОС;
* Оперативная память: 4 ГБ;
* Монитор;
* Клавиатура;
* Мышь.

3.5. Требования к информационной и программной совместимости

ПК с операционной системой Windows.

Требования к системе:

Windows 10 или более поздней версии.

Среда разработки Visual Studio 2022 и новее.

3.6. Требования к маркировке и упаковке

Программа должна распространяться на CD-диске.

3.7. Требования к транспортированию и хранению

Не предъявляются.

3.8. Специальные требования

Не предъявляются.

4. Требования к программной документации

4.1 Содержание расчётно-пояснительной записки:

Титульный лист

Оглавление

Введение

1. Объектно-ориентированный анализ предметной области
2. Проектирование классов
3. Логическая структура программы
4. Модульная структура программы
5. Тестирование программы

Заключение

Список литературы

Приложение 1. Техническое задание

Приложение 2. Текст программы

Приложение 3. Руководство пользователя

Приложение 4. Детальная диаграмма классов

4.2. Требования к оформлению

Требования к оформлению, установленные ГОСТ, должны быть выполнены на протяжении всей работы без каких-либо изменений (в табл. П1.1).

Таблица П1.1

Требования к оформлению

|  |  |
| --- | --- |
| Документ | Печать на отдельных листах формата А4 (210х297 мм); оборотная сторона не заполняется; листы нумеруются. Печать возможна ч/б. |
| Страницы | Ориентация — книжная; отдельные страницы, при необходимости, альбомная. Поля: верхнее, нижнее — по 2 см, левое — 3 см, правое — 1 см. |
| Абзацы | Межстрочный интервал — 1,5, перед и после абзаца — 0. |
| Шрифты | Кегль — 14. В таблицах шрифт 12. Шрифт листинга — 10 (возможно в 2 колонки). |
| Рисунки | Подписывается под ним по центру: «Рис.Х. Название В» приложениях: «Рис.П1.3. Название» |
| Таблицы | Подписывается: над таблицей, выравнивание по правому: «Таблица Х». В следующей строке по центру Название Надписи в «шапке» (имена столбцов, полей) — по центру. В теле таблицы (записи) текстовые значения — выравнены по левому краю, числа, даты — по правому. |

5. Стадии и этапы разработки

Стадии и этапы разработки представлены в таблице П1.2.

Таблица П1.2

Стадии и этапы разработки

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование этапа разработки | Сроки разработки | Результаты выполнения | Отметка о выполнении |
| Изучение предметной области | 10.03.2023 – 18.03.2023 | Предметная область изучена |  |
| Проектирование классов | 19.03.2023-30.03.2023 | Сделано проектирование классов |  |
| Логическое проектирование | 05.04.2023-30.04.2023 | Сделано логическое проектирование |  |
| Оформление технического задания | 05.05.2023 | Оформленное техническое задание |  |
| Модульное проектирование | 05.04.2023-30.04.2023 | Сделано модульное проектирование |  |
| Написание программы | 15.04.2023-  20.05.2023 | Конечный вариант программы |  |
| Оформление сопроводительной документации - РПЗ | 24.05.2023-  30.05.2023 | Оформленная сопроводительная документация |  |

6. Порядок контроля и приёмки

Порядок контроля и приёма представлены в таблице П1.3.

Таблица П1.3

Порядок контроля и приёма

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование контрольного этапа выполнения курсовой работы | Сроки контроля | Результат выполнения | Отметка о приёмке результата контрольного этапа |
| Оформление Техническое задание. | 26.05.2023 | Выполненное оформление ТЗ |  |
| Оформление сопроводительной документации – РПЗ. | 08.06.2023 | Выполненное оформление РПЗ |  |
| Сдача курсовой работы | 10.06.2023 | Получение оценки за выполненную работу |  |

Приложение 2. Текст программы

|  |
| --- |
| #pragma once  #include <iostream>  using namespace std;  class Interface\_Software {  public:  virtual void print()const = 0;  virtual double get\_price()const = 0;  virtual void FileOutput() const = 0;}; |

Рис. П2.1 Файл Interface\_Software.h

|  |
| --- |
| #pragma once virtual void print\_name()  #include<iostream> const = 0; string GetProperties()  #include"Enum1.h" const { string result = "";  #include<string> result += "name = ";  #include"Interface\_Software.h" result += get\_Name();  using namespace std; result += "\nprice = "  class Software:virtual public + to\_string(get\_price()) + "\n";  Interface\_Software result += "manufacturer="  {public:Software(const char\* + to\_string(get\_manufacturer())+ "\n";  name, int year, double price, switch (get\_device()) { case x64:  Platforms device); result += "Category =x64\n"; break;  Software(const Software& same); case x68:result += "Category = x68\n";break;  ~Software(); case IMUL: result += "Category = IMUL\n";}  void set\_device(Platforms device); result += "manufacturer = " +  void set\_Name(const char\* name); to\_string(get\_manufacturer()) + "\n";  void set\_year(int year); return result;  void set\_price(double price); return string();};  void set\_manufacturer(char private: char\* name = nullptr;  manufacturer); Platforms device; double price;  Platforms get\_device()const; int year;static char manufacturer;};  char\* get\_Name()const;  int get\_year()const;  double get\_price()const;  static char get\_manufacturer();  void print()const  void FileOutput(iostream& fs)  const {if (!fs) throw string  ("File read error");  fs <<GetProperties()<<endl;}; |

Рис. П2.2 Файл Software.h

|  |
| --- |
| #include "Software.h"  #include<iostream>  #include<string>  #include <fstream>  #include"Enum1.h"  using namespace std;  Software::Software(const char\* name, int year, double price, Platforms device)  {if (year > 2023) {  throw string("Выпуск устройство не может быть 2023+");}set\_Name(name); this->year = year; this->price = price;this->device= device;}  Software::Software(const Software& same)  :Software(same.name, same.year, same.price, same.device) {}  Platforms Software::get\_device() const{return device;}  char\* Software::get\_Name()const{return name;}  int Software::get\_year()const{return year;}  double Software::get\_price() const{ return price;}  char Software::get\_manufacturer(){return manufacturer;}  void Software::set\_device(Platforms device){ this->device = device;}  void Software::set\_Name(const char\* name) {  if (strlen(name) <= 0) throw exception( "ошибка имя не может быть пустой строкой!"); if (this->name != nullptr) { delete[] this->name; }this->name = new char[strlen(name) + 1];strcpy\_s(this->name, strlen(name) + 1, name);}  void Software::set\_year(int year){this->year = year;}  void Software::set\_price(double price){this->price = price;}  void Software::set\_manufacturer(char manufacturer){}  Software::~Software(){ delete[] name;}  void Software::print() const{ cout << GetProperties() << endl;} |

Рис. П2.3 Файл Software.cpp

|  |
| --- |
| #pragma once  #include"Software.h"  #include <iostream>  using namespace std;  class SystemSoftware: public Software{  private:string management;//управеление  string limited;//ограниченный  public:  SystemSoftware(const char\* name, int year, double price, Platforms device, string management, string limited): Software(name, year, price, device), management(management), limited(limited) {};  SystemSoftware(const SystemSoftware& same) : Software(same),limited(same.get\_limited()), management(same.get\_management()) {};  void set\_management(string value\_management) {management = value\_management;}  string get\_management()const {return management;};  void set\_limited(string value\_limited) {limited = value\_limited; }  string get\_limited()const { return limited; };  void print()const { Software::print(); };  string GetProperties() const {  string result = Software::GetProperties();  result += "management = " + string(management) + "\n"+ "limited = " + string(limited) + "\n"; return result;}}; |

Рис. П2.4 Файл SystemSoftware.h

|  |
| --- |
| #pragma once  #include"Software.h"  #include <iostream>  #include"Enum1.h"  #pragma warning (disable: 4996)  using namespace std;  class ApplicationSoftware : public Software{  public:ApplicationSoftware(const char\* name, int year, double price, Platforms device, string creature, string free): Software(name, year, price, device), creature(creature), free(free) {};  ApplicationSoftware(const ApplicationSoftware& same) :Software(same),free(same.get\_free()), creature(same.get\_creature()) { };  void print()const { Software::print(); };  string GetProperties() const {  string result = Software::GetProperties();  result += "= "; result += creature+free; result += "\n"; return result;}  void set\_Creature(string creature) { creature = creature; }  string get\_creature()const { return creature; }  void set\_free(string free) { free = free; }  string get\_free()const { return free; }  private: string creature;//управеление string free;//ограниченный}; |

Рис. П2.5 Файл ApplicationSoftware.h

|  |
| --- |
| #pragma once  #include "SystemSoftware.h"  #include <iostream>  using namespace std;  class OperatingSystem:public SystemSoftware{public:  OperatingSystem(const char\* name, int year, double price, Platforms device, string management, string limited, OS OS,string functioning\_and\_access)  : SystemSoftware(name,year,price,device, management, limited),OS(OS), functioning\_and\_access(functioning\_and\_access) {};  void print()const {SystemSoftware::print(); cout << OS << " "; };  void print\_name()const { cout << "Operating System" << endl;};  OperatingSystem(const OperatingSystem& same) :SystemSoftware(same), functioning\_and\_access(same.get\_functioning\_and\_access()) {};  void set\_functioning\_and\_access(string value\_functioning\_and\_access) {  functioning\_and\_access = functioning\_and\_access;}  string get\_functioning\_and\_access ()const { return functioning\_and\_access;};  string GetProperties() const {  string result =SystemSoftware::GetProperties();result += OS;result += "functioning\_and\_access = " + string(functioning\_and\_access) + "\n";  return result;}  protected: OS OS; string functioning\_and\_access;}; |

Рис. П2.6 Файл OperatingSystem.h

|  |
| --- |
| #pragma once  #include "SystemSoftware.h"  #include <iostream>  using namespace std;  class Utilities : public SystemSoftware{  public:  Utilities(const char\* name, int year, double price, Platforms device, string management, string limited, Uti uti, string checking\_and\_setting\_up)  : SystemSoftware(name, year, price, device, management, limited), uti(uti), checking\_and\_setting\_up(checking\_and\_setting\_up) {};  Utilities(const Utilities& same) :SystemSoftware(same), checking\_and\_setting\_up(same.get\_checking\_and\_setting\_up()) {};  void set\_checking\_and\_setting\_up(string checking\_and\_setting\_up) {checking\_and\_setting\_up = checking\_and\_setting\_up;}  string get\_checking\_and\_setting\_up()const { return checking\_and\_setting\_up; };  void print()const { SystemSoftware::print();cout << uti << " "; };  void print\_name() const {cout << "Utilities" << endl;};  string GetProperties() const { string result = SystemSoftware::GetProperties(); result += uti;  result += "checking\_and\_setting\_up = " + string(checking\_and\_setting\_up) + "\n";  return result;}  private:Uti uti; string checking\_and\_setting\_up;}; |

Рис. П2.7 Файл Utilities.h

|  |
| --- |
| #include "ApplicationSoftware.h"  #include "Enum1.h"  #include <iostream>  using namespace std;  class GraphicEditors :public ApplicationSoftware{  public:  GraphicEditors(const char\* name, int year, double price, Platforms device, string creature, string free, form2 format2, string tip\_info, type\_of\_presentation2 type2)  : ApplicationSoftware(name, year, price, device, creature, free), format2(format2), tip\_info(tip\_info), type2(type2) { };  GraphicEditors(const GraphicEditors& same) :ApplicationSoftware(same), tip\_info(same.get\_tip\_info()){};  void set\_tip\_info(string tip\_info) { tip\_info = tip\_info; }  string get\_tip\_info()const { return tip\_info; }  void print()const {  ApplicationSoftware::print();cout << format2 << " ";cout << type2 << endl;};  void print\_name()const { cout << "GraphicEditors" << endl;};  string GetProperties() const { string result = ApplicationSoftware::GetProperties();  result += " = "; result += tip\_info; result +=format2 + type2;  result += "\n"; return result; }  protected:form2 format2; string tip\_info; type\_of\_presentation2 type2;}; |

Рис. П2.8 Файл GraphicEditors.h

|  |
| --- |
| #include "ApplicationSoftware.h"  #include "Enum1.h"  #include <iostream>  using namespace std;  class TextEditors :public ApplicationSoftware{public:  TextEditors(const char\* name, int year, double price, Platforms device, string creature, string free, form1 format1, string tip\_info,type\_of\_presentation1 type1)  : ApplicationSoftware(name, year, price, device, creature, free), format1(format1), tip\_info(tip\_info), type1(type1) {};  TextEditors(const TextEditors& same) :ApplicationSoftware(same), tip\_info (same.get\_tip\_info()) {};  void set\_tip\_info(string tip\_info) { tip\_info = tip\_info; }  string get\_tip\_info()const { return tip\_info; }  void print()const { ApplicationSoftware::print();  cout << format1 << " "; cout << type1 << endl; };  void print\_name()const { cout << "TextEditors" << endl; };  string GetProperties() const{string result = ApplicationSoftware::GetProperties();  result += "= "; result += tip\_info; result += format1 + type1;  result += "\n";return result;}  protected: form1 format1; string tip\_info; type\_of\_presentation1 type1;}; |

Рис. П2.9 Файл TextEditors.h

|  |
| --- |
| #pragma once  enum Platforms{ x64,x68,IMUL,};  enum OS{Microsoft\_Windows, macOS,Linux,FreeBSD, Android,};  enum Uti{File\_Manager,Archivers,Viewers,Utilities\_for\_hardware\_or\_software\_diagnostics,  Crash\_Recovery\_Utilities,Disk\_Optimizer,File\_Shredders,The\_uninstaller,Process\_management\_utilities,};  enum form1{txt,rtf,pdf,odt,doc,docx,html,};  enum form2{gif,jpeg,png,bmp,tiff,psd,raw,};  enum type\_of\_presentation1{document,book,};  enum type\_of\_presentation2{photography,infographics,drawing,diagram,}; |

Рис. П2.10 Файл Enum1.h

|  |
| --- |
| #pragma once  #include<iostream>  #include"Interface\_Software.h"  using namespace std;  class Stack{ struct Bufer {  Interface\_Software\* data;  Bufer\* next;  Bufer(Interface\_Software\* d) {data = d; next = nullptr;} };  public:Stack(); ~Stack();  int get\_count() { return size;}  void sorting();  void push(Interface\_Software\* d);  bool empty();  void find\_by\_price(double price) const;  void print();  Interface\_Software\* pop();  Interface\_Software\* getCountingByIndex(int index) const {  if (index >= size) { return nullptr; }  Bufer\* ptr = top; int count = 0;  while (ptr->data != nullptr)  {if (count == index) return ptr->data;  if (ptr->next != nullptr) { ptr = ptr->next; count++; }  else break; } return nullptr; };  private: int size = 5000; Bufer\* top;}; |

Рис. П2.11 Файл Stack.h

|  |
| --- |
| #include<string>  #include<iostream>  #include "Stack.h"  using namespace std;  Stack::Stack()  { top = nullptr;size = 0;}  Stack::~Stack(){if (!empty())  {Interface\_Software\* top1 = top->data;  Bufer\* ptr = top;top = top->next;delete ptr; }}  bool Stack::empty(){return top == nullptr;}  void Stack::sorting(){ if (empty()) { return; }  for (int i = 0; i < size; i++) { Bufer\* ptr = top;  while (ptr != NULL && ptr->next != NULL)  {if (ptr->data->get\_price() > ptr->next->data->get\_price()) {  swap(ptr->data, ptr->next->data); } ptr = ptr->next; } };}  void Stack::push(Interface\_Software\* d)  { Bufer\* ptr = new Bufer(d); size++;  if (!empty()) { ptr->next = top;} top = ptr;}  Interface\_Software\* Stack::pop()  { if (empty()) { return nullptr; }  Interface\_Software\* top1 = top->data;  Bufer\* ptr = top; top = top->next; size--; }  void Stack::find\_by\_price(double price) const  { cout << " значение больше:" << price << endl;  Bufer\* ptr = top; while (ptr != NULL)  { if (ptr->data->get\_price() > price) {  ptr->data->print(); }  ptr = ptr->next;} cout << endl; }  void Stack::print()  { if (empty()) { return; }  Bufer\* ptr = top;  while (ptr != NULL) {  ptr->data->print();  ptr = ptr->next; } cout << endl;} |

Рис. П2.12 Файл Stack.cpp

|  |
| --- |
| #pragma once  #include "Void\_Dynamic\_Vector.h"  template <class T>  class Dynamic\_Vector :private Void\_Dynamic\_Vector {  public:  void popBack() { Void\_Dynamic\_Vector::\_pop\_back();}  void push(T\* p) { Void\_Dynamic\_Vector::\_push\_back(p); };  T\* pop() { return static\_cast<T\*>(Void\_Dynamic\_Vector::\_pop());};  int size() { return Void\_Dynamic\_Vector::\_size(); };  int empty() {return Void\_Dynamic\_Vector::\_empty();};}; |

Рис. П2.13 Файл Dynamic\_Vector.h

|  |
| --- |
| #pragma once  #include<vector>  #include "Interface\_Software.h"  class Void\_Dynamic\_Vector {  protected:  void \_pop\_back() { VectorInterface.pop\_back(); }  void \_push\_back(void\* p) {VectorInterface.push\_back(p); }  void\* \_pop() { return VectorInterface.back(); }  int \_size() {return VectorInterface.size(); }  int \_empty() { return VectorInterface.empty(); }  private:  std::vector <void\*> VectorInterface;  }; |

Рис. П2.14 Файл Void\_Dynamic\_Vector.h

|  |
| --- |
| #define \_CRT\_SECURE\_NO\_WARNINGS #include"Interface\_Software.h"  #define \_CRT\_SECURE\_NO\_DEPRECATE #include"Enum1.h"  #define \_CRT\_NONSTDC\_NO\_DEPRECATE #include"Stack.h"  #include"Void\_Dynamic\_Vector.h" #include"OperatingSystem.h"  #include"Software.h" #include"SystemSoftware.h"  #include"ApplicationSoftware.h" #include"Utilities.h"  #include"Utilities.h" #include"GraphicEditor.h"  #include"TextEditors.h" #include"Dynamic\_Vector.h"  #include<iostream>  #include<string>  #include<fstream>  using namespace std;  char Software::manufacturer=0;  int main() {  setlocale(LC\_ALL, "Russian");  Dynamic\_Vector <Interface\_Software> vectorINT;  Dynamic\_Vector<void\*>vec;  Stack stack; int action = 1;  while (action) {  try {  cout << "Выберите действие" << endl;  cout << "1: Чтение из файла" << endl;  cout << "2: Добавить \"OperatingSystem\" " << endl;  cout << "3: Добавить \"Utilities\" " << endl;  cout << "4: Добавить \"GraphicEditors\" " << endl;  cout << "5: Добавить \"TextEditors\" " << endl;  cout << "6. Вывод лементов в консоль" << endl;  cout << "7. Вывод элементов в файл" << endl;  cout << "8: Удалить последний элемент " << endl;  cout << "9: Сортировать " << endl;  cout << "0 - Закончить работу" << endl;  cin >> action;  switch (action) {  case 1:{int count = 0;ifstream f("input.txt");  if (!f) {cout << "File was not open" << endl;return 0;}  while (!f.eof()) { string type; getline(f, type, ' ');  type = type[0] == '\n' ? type.substr(1) : type;  if (type == "GraphicEditors") { int year; double price; Platforms device; string creature; string free; form2 format2; string tip\_info; type\_of\_presentation2 type2;  string \_brand, res, brd;  getline(f, brd, ' ');\_brand = brd; getline(f, res, ' ');  (res == "x64") { device = x64; }  else { if (res == "x68") { device = x68; }else { device = IMUL; }}  getline(f, brd, ' ');\_brand = brd;getline(f, res, ' ');  if (res == "gif") { format2 = gif; }  else { if (res == "jpeg") { format2 = jpeg; }  else { if (res == "png") { format2 = png; }  else { if (res == "bmp") { format2 = bmp; }  else { if (res == "tiff") { format2 = tiff; }  else {if (res == "psd") { format2 = psd; } else { format2 = raw; }}}}}}  getline(f, brd, ' ');\_brand = brd; getline(f, res, ' ');  if (res == "photography") { type2 = photography; }  else { if (res == "infographics") { type2 = infographics; }  else { if (res == "drawing") { type2 = drawing; }  else { type2 = diagram; }}}  f >> year; f >> price;  f >> creature; f >> free;  f >> tip\_info; count++;  const char\* name = strcpy(new char[\_brand.length() + 1], \_brand.c\_str());  Interface\_Software\* Graphic = new GraphicEditors(name, year, price, device, creature, free, format2, tip\_info, type2);stack.push(Graphic);vectorINT.push(Graphic); }  else if (type == "TextEditors") {  int year; double price; Platforms device; string creature; string free; form1 format1; string tip\_info; type\_of\_presentation1 type1;  string \_brand, res, brd; getline(f, brd, ' ');  \_brand = brd;getline(f, res, ' ');  if (res == "x64") { device = x64; }  else { if (res == "x68") { device = x68; }else { device = IMUL; }}  getline(f, brd, ' ');\_brand = brd;  getline(f, res, ' ');  if (res == "txt") { format1 = txt; }  else { if (res == "rtf") { format1 = rtf; }  else {if (res == "pdf") { format1 = pdf; }  else { if (res == "odt") { format1 = odt; }  else { if (res == "doc") { format1 = doc; }  else {if (res == "docx") { format1 = docx; }  else { format1 = html; } }}}}}  getline(f, brd, ' '); \_brand = brd;  getline(f, res, ' ');if (res == "document") { type1 = document; }  else { type1 = book; }f >> year;  f >> price; f >> creature;  f >> free; f >> tip\_info; count++;  char\* name = strcpy(new char[\_brand.length() + 1], \_brand.c\_str());  Interface\_Software\* Text = new TextEditors(name, year, price, device, creature, free, format1, tip\_info, type1);  stack.push(Text); vectorINT.push(Text);}  else if (type == "OperatingSystem") {  int year; double price; Platforms device; string management; string limited; OS OS; string functioning\_and\_access;  string \_brand, res, brd; getline(f, brd, ' ');  \_brand = brd;getline(f, res, ' ');  if (res == "x64") { device = x64; }  else {if (res == "x68") { device = x68; }  else { device = IMUL; } }  getline(f, brd, ' ');\_brand = brd;  getline(f, res, ' ');if (res == "Microsoft\_Windows") { OS = Microsoft\_Windows; }  else { if (res == "macOS") { OS = macOS; }  else { if (res == "Linux") { OS = Linux; }  else { if (res == "FreeBSD") { OS = FreeBSD; }  else { OS = Android; }}} }  f >> year; f >> price;  f >> management; f >> functioning\_and\_access;  f >> limited; count++;  char\* name = strcpy(new char[\_brand.length() + 1], \_brand.c\_str());  Interface\_Software\* Operating = new OperatingSystem(name, year, price, device, management, limited, OS, functioning\_and\_access);  stack.push(Operating); vectorINT.push(Operating);}  else if (type == "Utilities") {  int year; double price; Platforms device; string management; string limited; Uti uti; string checking\_and\_setting\_up;  string \_brand, res, brd; getline(f, brd, ' ');  \_brand = brd;getline(f, res, ' ');  if (res == "x64") { device = x64; }  else { if (res == "x68") { device = x68; }  else { device = IMUL; } }  getline(f, brd, ' '); \_brand = brd;  getline(f, res, ' ');if (res == "File\_Manager") { uti = File\_Manager; }  else { if (res == "Archivers") { uti = Archivers; }  else { if (res == "Viewers") { uti = Viewers; }  else { if (res == "Utilities\_for\_hardware\_or\_software\_diagnostics") { uti = Utilities\_for\_hardware\_or\_software\_diagnostics; }  else { if (res == "Crash\_Recovery\_Utilities") { uti = Crash\_Recovery\_Utilities; }  else { if (res == "Disk\_Optimizer") { uti = Disk\_Optimizer; }  else { if (res == "File\_Shredders") { uti = File\_Shredders; }  else { if (res == "The\_uninstaller") { uti = The\_uninstaller; }  else { uti = Process\_management\_utilities; } }}}}}}  f >> year; f >> price;  f >> management;f >> checking\_and\_setting\_up;  f >> limited;count++;  char\* name = strcpy(new char[\_brand.length() + 1], \_brand.c\_str());  Interface\_Software\* Utilit = new Utilities(name, year, price, device, management, limited, uti, checking\_and\_setting\_up);  stack.push(Utilit); vectorINT.push(Utilit); } }  if (count > 0) {cout << "Данные из файла прочитаны" << endl; }  else { cout << "Некорректный формат файла" << endl; return 0;}break;}  case 2: //OperatingSystem  {cout << "Выберите куда добавить элемент" << endl;  cout << "1:стэк\n";  cout << "2: Динамический вектор\n";  cout << "3: В динамический вектор и стэк" << endl;  int roll; cin >> roll;//выбор  char\* name = new char[128];  cout << "Введите название утилиты: ";  scanf("%s", name); int year;  cout << "Введите год выпуска: ";  cin >> year; double price;  cout << "Введите цену: "; cin >> price;  string limited = "ограниченный в использовании";//ограниченный  string functioning\_and\_access = "функционирование\_и\_доступ";  string management = "назначение:управеление";//управеление  Platforms device; OS OS; int k;  cout << "Введите вид платформы, если x68 - 1, x64 - 2, IMUL -3: ";  cin >> k;  switch (k) {  case(1): device = x68; break;  case(2): device = x64; break;  case(3): device = IMUL; break; }  int ki;  cout << "Введите операционную систему, если Microsoft\_Windows - 1, macOS - 2, Linux -3, FreeBSD - 4,Android - 5 : ";  cin >> ki;  switch (ki) {  case(1): OS = Microsoft\_Windows; break;  case(2): OS = macOS; break;  case(3): OS = Linux; break;  case(4): OS = FreeBSD; break;  case(5): OS = Android; break; }  Interface\_Software\* Operating = new OperatingSystem(name, year, price, device, management, limited, OS, functioning\_and\_access);  switch (roll) {case 1: {  cout << "Элемент класса GraphicEditors добавлен" << endl;  stack.push(Operating); break; }  case 2: {  cout << "Элемент класса GraphicEditors добавлен" << endl;  vectorINT.push(Operating);break; }  case 3:  {cout << "Элемент класса LaserRecordingDevice добавлен" << endl;  stack.push(Operating); vectorINT.push(Operating); break; }  default: { cout << "Ошибка в выборе контейнера для элемента" << endl;  break; }}break; }  case 3: //Utilities  { cout << "Выберите куда добавить элемент" << endl;  cout << "1:стэк\n";  cout << "2: Динамический вектор\n";  cout << "3: В динамический вектор и стэк" << endl;  int roll; cin >> roll;//выбор  char\* name = new char[128];  cout << "Введите название утилиты: ";  scanf("%s", name); int year;  cout << "Введите год выпуска: ";  cin >> year; double price;  cout << "Введите цену: "; cin >> price;  string limited = "ограниченный в использовании";//ограниченный  string checking\_and\_setting\_up = "проверка и настройка";  string management = "назначение:управеление";//управеление  Platforms device; Uti uti; int k;  cout << "Введите вид платформы, если x68 - 1, x64 - 2, IMUL -3: ";  cin >> k; switch (k) {  case(1):device = x68; break;  case(2): device = x64; break;  case(3): device = IMUL; break; } int ki;cout << "Введите вид утилиты, если File\_Manager - 1, Archivers - 2, Viewers -3, Utilities\_for\_hardware\_or\_software\_diagnostics - 4,Crash\_Recovery\_Utilities -5, Disk\_Optimizer - 6,File\_Shredders - 7,The\_uninstaller -,Process\_management\_utilities - 9 : "; cin >> ki;  switch (ki) {  case(1): uti = File\_Manager; break;  case(2): uti = Archivers; break;  case(3): uti = Viewers; break;  case(4): uti = Utilities\_for\_hardware\_or\_software\_diagnostics; break;  case(5):uti = Crash\_Recovery\_Utilities;break;  case(6): uti = Disk\_Optimizer; break;  case(7): uti = File\_Shredders;break;  case(8): uti = The\_uninstaller; break;  case(9):uti = Process\_management\_utilities; break; }  Utilities\* Utilitiesi = new Utilities(name, year, price, device, management, limited, uti, checking\_and\_setting\_up);  switch (roll) {  case 1: {cout << "Элемент класса GraphicEditors добавлен" << endl;  stack.push(Utilitiesi);break;}  case 2: {cout << "Элемент класса GraphicEditors добавлен" << endl;  vectorINT.push(Utilitiesi); break; }  case 3:{  cout << "Элемент класса LaserRecordingDevice добавлен" << endl;  stack.push(Utilitiesi);vectorINT.push(Utilitiesi);break }  default: cout << "Ошибка в выборе контейнера для элемента" << endl;break;}}break; }  case 4: //GraphicEditors  {cout << "Выберите куда добавить элемент" << endl;  cout << "1:стэк\n";  cout << "2: Динамический вектор\n";  cout << "3: В динамический вектор и стэк" << endl;  int roll; cin >> roll;//выбор  char\* name = new char[128];  cout << "Введите название редактора: ";  scanf("%s", name); int year;  cout << "Введите год выпуска: ";cin >> year;  double price; cout << "Введите цену: ";  cin >> price;string free = "свободный в использовании";//ограниченный  string tip\_info = "графический тип";  string creature = "назначение:создание";//управеление  Platforms device; form2 format2;  type\_of\_presentation2 type2; int k;  cout << "Введите вид платформы, если x68 - 1, x64 - 2, IMUL -3: "; cin >> k;  switch (k) {  case(1): device = x68; break;  case(2):device = x64; break;  case(3):device = IMUL; break; }  int ki; cout << "Введите вид вормата, если gif - 1, jpeg - 2, png -3, bmp-4,tiff - 5, psd- 6,raw-7 : "; cin >> ki;  switch (ki) {  case(1): format2 = gif; break;  case(2):format2 = jpeg; break;  case(3): format2 = png; break;  case(4):format2 = bmp; break;  case(5): format2 = tiff; break;  case(6): format2 = psd; break;  case(7): format2 = raw; break; }  int kis;cout << "Введите тип изображения, если фотография - 1, инфографика - 2, рисунок -3, диаграмма-4 : ";cin >> kis;  switch (kis) {  case(1):type2 = photography; break;  case(2):type2 = infographics; break;  case(3): type2 = drawing; break;  case(4): type2 = diagram; break; }  Interface\_Software\* Graphic = new GraphicEditors(name, year, price, device, creature, free, format2, tip\_info, type2);  switch (roll) {  case 1: {cout << "Элемент класса GraphicEditors добавлен" << endl;  stack.push(Graphic);break;}  case 2: {cout << "Элемент класса GraphicEditors добавлен" << endl;  vectorINT.push(Graphic); break; }  case 3:{cout << "Элемент класса LaserRecordingDevice добавлен" << endl;  stack.push(Graphic);vectorINT.push(Graphic);break }  default: {cout << "Ошибка в выборе контейнера для элемента" << endl; break;  }}break;}  case 5: //TextEditors  {cout << "Выберите куда добавить элемент" << endl;  cout << "1:стэк\n";  cout << "2: Динамический вектор\n";  cout << "3: В динамический вектор и стэк" << endl;  int roll; cin >> roll;//выбор  char\* name = new char[128];  cout << "Введите название редактора: ";  scanf("%s", name); int year;  cout << "Введите год выпуска: ";cin >> year;  double price;  cout << "Введите цену: "; cin >> price;  string free = "свободный в использовании";//ограниченный  string tip\_info = "графический тип";  string creature = "назначение:создание";//управеление  Platforms device; form1 format1;  type\_of\_presentation1 type1; int k;  cout << "Введите вид платформы, если x68 - 1, x64 - 2, IMUL -3: ";  switch (k) {  case(1): device = x68; break;  case(2): device = x64; break;  case(3):device = IMUL; break; }cin >> k; int ki;  cout << "Введите вид вормата, если txt - 1, rtf - 2,pdf -3, odt-4,doc - 5, docx- 6,html-7 : "; cin >> ki;  switch (ki) {  case(1): format1 = txt; break;  case(2): format1 = rtf; break;  case(3): format1 = pdf; break;  case(4):format1 = odt;break;  case(5):format1 = doc; break;  case(6):format1 = docx; break;  case(7):format1 = html; break; } int kis;  cout << "Введите тип изображения, если документ - 1, книга - 2 : ";cin >> kis;  switch (kis) {  case(1):type1 = document; break;  case(2):type1 = book; break; }  Interface\_Software\* Text = new TextEditors(name, year, price, device, creature, free, format1, tip\_info, type1);  switch (roll) {  ase 1: cout << "Элемент класса GraphicEditors добавлен" << endl;  stack.push(Text); break; }  case 2: {cout << "Элемент класса GraphicEditors добавлен" << endl;  vectorINT.push(Text);break}  case 3:{  cout << "Элемент класса LaserRecordingDevice добавлен" << endl;  stack.push(Text); vectorINT.push(Text); break; }  default: {cout << "Ошибка в выборе контейнера для элемента" << endl;break;}}break;}  case 6:// консоль  {int b;  cout << "Выберите какой контейнер будет выводится в консоль" << endl;  cout << "1. Стек\n";  cout << "2. Динамический вектор\n";  cin >> b; switch (b) {  case 1: {cout << "Вывод всех элементов из динамического вектора в консоль" << endl;  for (int i = 0; i < stack.get\_count(); i++) {  if (!stack.empty()) {  stack.getCountingByIndex(i)->print();} }break;}  case 2: {  ofstream out("output1.txt");  cout << "Вывод списка: " << endl;  Dynamic\_Vector<Interface\_Software> recordingptr1;  recordingptr1 = vectorINT;  int countList;  out << "Все элементы односвязного списка выведены в файл\n";  countList = vectorINT.size();  for (int i = 0; i < countList; i++) {  //recordingptr1.pop()->FileOutput(out);  recordingptr1.popBack(); }break;}  default: {cout << "Некорректный выбор действия" << endl;break;}}break;}  case 7:// файл  {int b;cout << "Выберите какой контейнер будет выводится в файл" << endl;  cout << "1. стек\n";  cout << "2. Динамический вектор\n";  cin >> b;  switch (b) {  case 1: {  ofstream out("output.txt");  cout << "Все элементы динамического массива выведены в файл" << endl;  for (int i = 0; i < stack.get\_count(); i++) {  out << "Устройство Номер " << (i + 1) << endl;  if (!stack.empty()) { //stack.getCountingByIndex(i)->FileOutput(out);}} break; } case 2: { ofstream out("output1.txt");  cout << "Вывод списка: " << endl;  Dynamic\_Vector<Interface\_Software> recordingptr1;  recordingptr1 = vectorINT;  int countList;  out << "Все элементы односвязного списка выведены в файл\n";  countList = vectorINT.size();  for (int i = 0; i < countList; i++) {  //recordingptr1.pop()->FileOutput(out);  recordingptr1.popBack();} break; }  default: { cout << "Некорректный выбор действия" << endl;break;}}break;}  case 9: {int b;  cout << "Выберите действие" << endl;  cout << "Удалить последний элемент в векторе" << endl;  cout << "Удалить последний элемент в списке" << endl;  cin >> b;  switch (b)  {case 1: stack.pop(); break;  case 2: vectorINT.popBack(); break;  default:cout << "Некорректный выбор действия" << endl; break; }break;}  case 10:{stack.sorting();cout << "Стек отсортирован" << endl;break;}    catch (string error\_message) { cout << error\_message << endl; action = 0; }  catch (...) {cout << "Возникла неизвестная ошибка" << endl;}}  system("pause"); return 0;} |

Рис. П2.15 Файл Main.cpp

Приложение 3. Руководство пользователя

1. Общие сведения о программе

Исполнительный файл: Project140423.exe

1. Описание установки

Установка программы не требуется.

1. Описание запуска

Двойным нажатием левой кнопки мыши запустить файл Project140423.exe Инструкция по работе

4.1 Описание ручного ввода

В открывшемся консольном окне выбрать режим работы путем ввода цифры от 0 до 10 (рис. П3.1). При вводе цифры 0 программа завершит работу. Действия 1-5 нужны для ввода, 6,7 для вывода, а 8-11 для работы со стеком. При выборе ручного ввода данные вводятся строго согласно пояснениям в самой консоли. Если указан выбор из цифр, вводится только та цифру, которая есть в списке предложенных (рис. П3.2).

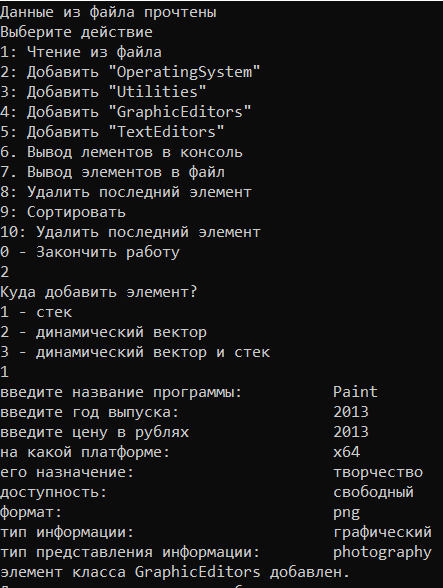


Рис. П3.1. Консольное окно с выбором действия

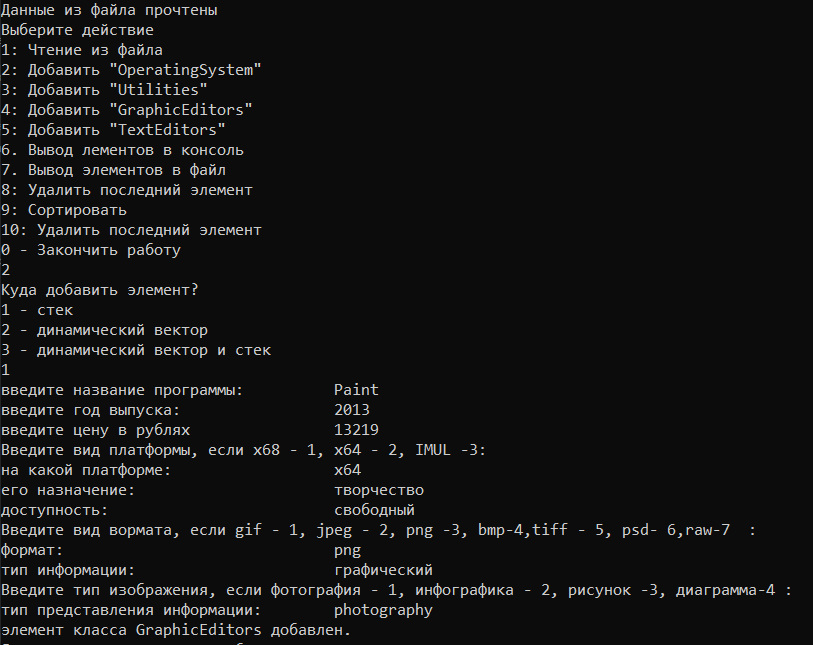


Рис. П3.2. Интерфейс

4.2 Описание файлового ввода

При выборе действия чтения из файла, данные заполняются автоматически, так как файл задан в программе (Рис. П3.3).

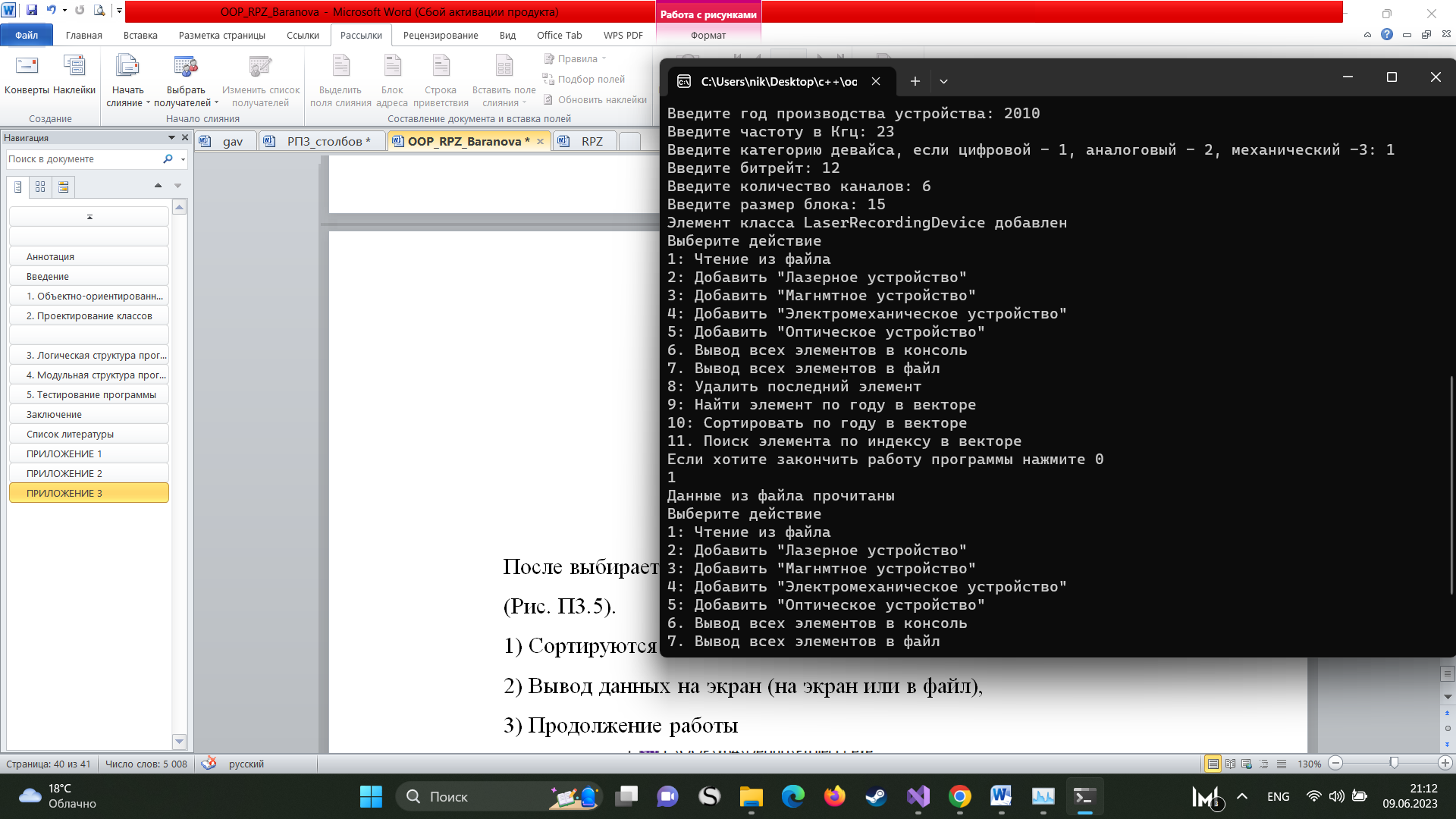


Рис. П3.3. Ввод данных из файла

После пользователь может продолжить выбирать дальнейшее действия с введенными данными:

Ниже приведен пример заполнения данных в файле. Каждое устройство прописывается в отдельной строке (Рис. П3.4).

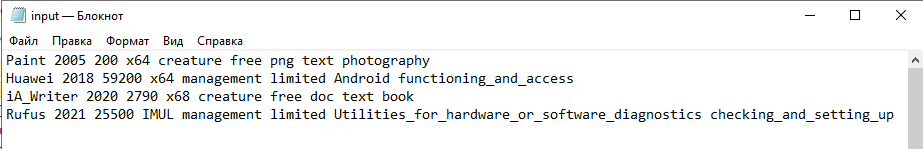


Рис. П3.4. Пример ввода данных

4.3 Описание файлового вывода

В открывшемся консольном окне ввести 7 номер действия (файловый вывод). После выбирается дальнейшее действие: 1 – вывод элементов стека в файл, 2 – вывод элементов динамического вектора в файл (Рис. П3.5).

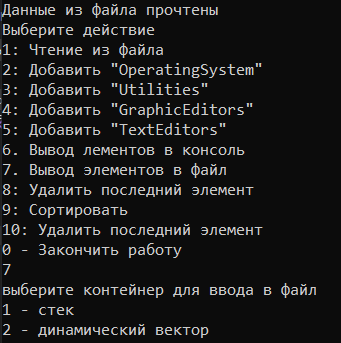


Рис. П3.4. Пример ввода данных

Пример вывода в файл из стека, а также из динамического вектора (рис. П.3.5).

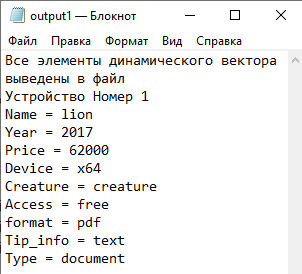
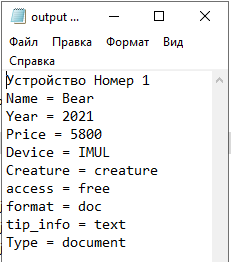


Рис. П.3.5. Вывод в файл из стека динамического вектора

Приложение 4. Детальная диаграмма классов



Рис. П.4.1. Детальная диаграмма классов