Введение

Раскрыть актуальность, практическую значимость выбранной темы, привести цели, задачи, объект и предмет исследования.

Основная часть

Теоретическая часть:

Рассмотреть теоретические аспекты выбранной темы:

- понятие разработки десктопных приложений;

- понятие интегрированная среда разработки;

- история интегрированных сред разработки;

- используемый стек технологий.

Практическая часть:

Разработка интегрированной среды разработки для языков программирования C++, Python.

Заключение

Подвести итоги дипломной работы, сделать краткие выводы, выявить соответствие между целью работы и полученными результатами, определить дальнейшие перспективы.

Введение

В современном мире не обойтись без программистов, постепенная автоматизация и компьютеризация всех сфер жизни человека не даёт разработчикам различного программного обеспечения и алгоритмов автоматизации отдыха. Но для работы самих программистов требуется специальное программное обеспечение, от самих языков программирования до различных сред работы и проектирования программного обеспечения.

Актуальность данной работы обусловлена постепенным уходом с российского рынка западных компаний и прекращением предоставления лицензий гражданам и компаниям в России в связи с этим необходимо разрабатывать российские аналоги программного обеспечения.

Интегрированная среда разработки — это основа работы любого программиста и без неё скорость и удобство работы может значительно упасть. IDE нам предоставляет удобные инструменты для разработки и отладки проектов разной сложности и объёма.

Целью данной работы является разработка интегрированной среды разработки для языков программирования C++, Python.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Изучить понятие разработка интегрированной среды разработки.
2. Описание программного обеспечения, языков и технологий, использованных в разработке.
3. Проектирование интегрированной среды разработки.
4. Разработка.
5. Рассмотрение продвижения и улучшения программного продукта.

Объектом исследования является интегрированная среда разработки, предметом исследования является разработка интегрированная среда разработки для языков программирования C++, Python для Linux и Windows.

Структура дипломной работы состоит из введения, теоретической и практической части, заключения и списка используемых источников.

Введение содержит актуальность, цель, практическую значимость,

задачи, объект и предмет исследования. В теоретической части рассмотрены основные элементы, которые использовались для разработки интегрированной среды разработки, такие как: понятие IDE, используемые компоненты, среда разработки, доступный функционал инструментов. В практической части описаны этапы разработки интернет-магазина канцтоваров, проектирование макетов и их описание.

При разработке были использованы различные источники информации, такие как учебная литература и документация для различных библиотек и Фреймворков, а также различные Интернет-ресурсы содержащие теоретические материалы.

# 1 Теоретическая часть

# Интегрированная среда разработки

# Интегрированная среда разработки(IDE) – это комплекс различных программных средств для работы программиста.

IDE как правило включает в себя:

Текстовый редактор;

Способы взаимодействия с установленным на устройстве или внедрённым в программу компилятором и/или интерпретатором;

Средства автоматизации сборки,

Средства автоматизации отладки.

Также IDE может включать в себя средства интеграции для работы с различными Фреймворками и системами управления версий. IDE может включать в себя конструкторы графического интерфейса. Зачастую IDE создаётся для нескольких языков программирования схожих по принципу работы или применяющихся в одной сфере программирования. Большинство современных IDE имеют графический интерфейс что значительно упрощает работу.

* 1. История интегрированных сред разработки

Первые IDE использовались без графического интерфейса и для работы на таком программном обеспечении требовалось заучивание различных комбинации клавиш или держания под рукой мануала для работы с программой.

1.2.1 Ранние этапы (1950–1970-е годы)

Разработчики применяли примитивные инструменты для работы. Текстовые редакторы и компиляторы работали отдельно друг от друга. Программисты писали код на перфокартах или в текстовых файлах, а затем использовали отдельные программы для компиляции и отладки.

Примером раннего инструментария можно считать IBM FORTRAN Assembly Program (1950-е годы), который позволял писать и компилировать код на языке FORTRAN.

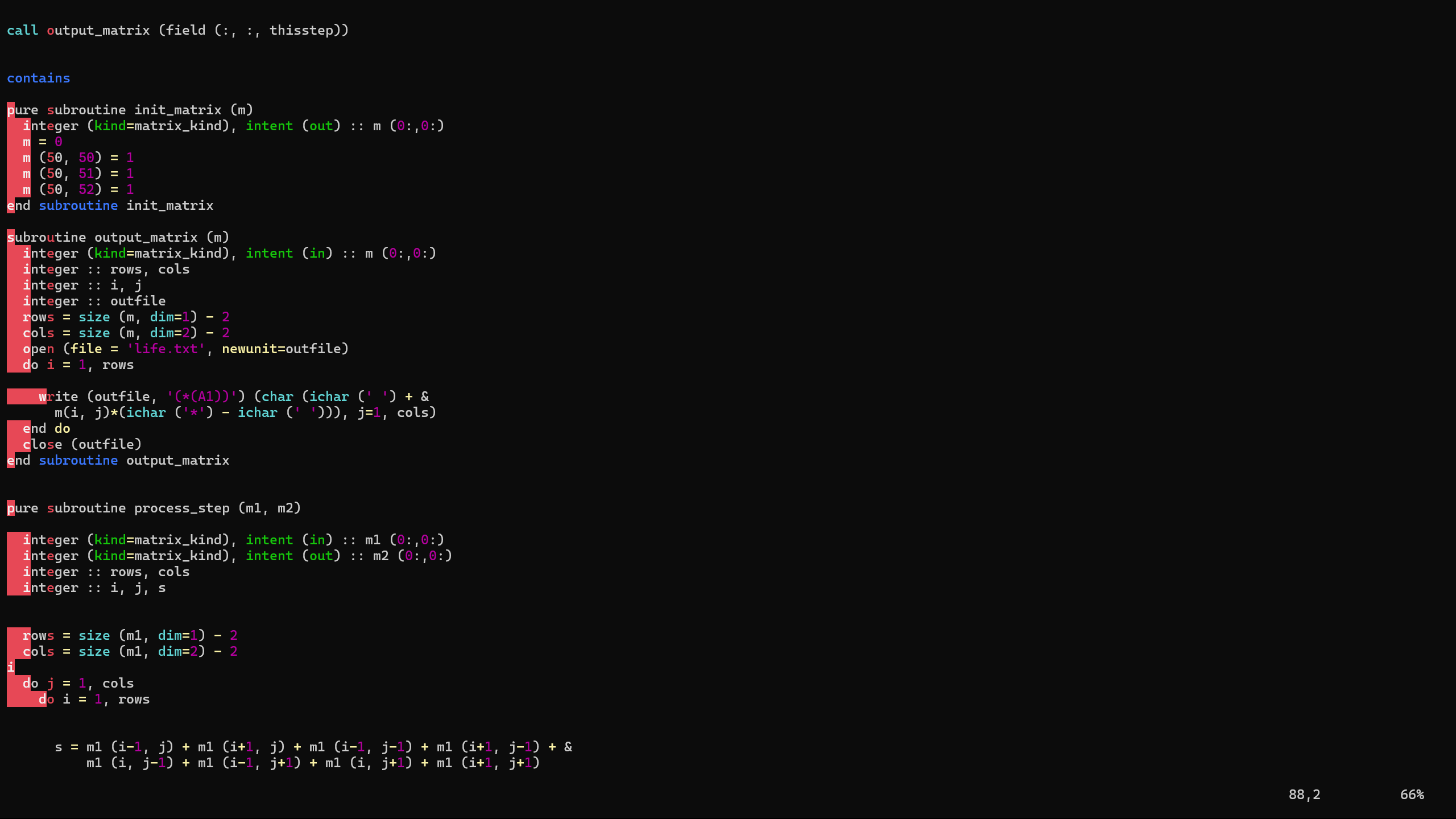


Рисунок 1 – IBM FORTRAN Assembly Program.

1.2.2 Появление первых IDE (1970–1980-е годы)

В середине 70-х годов начали появляются программы, объединившие в себе функции текстового редактора и компилятора с отладчиком напоминая современные среды разработки.

Одной из первых IDE считается Maestro I (1975 год), разработанная для операционной системы OS/360. Она стала стандартом индустрии.

В какой-то момент по всему миру было установлено 22 000 систем. Первые системы в США были установлены в 1979 году в компании Boeing, это были восемь систем Maestro I, а также в Bank of America, где было 24 системы и 576 терминалов разработчиков. До 1989 года в Федеративной Республике Германия было установлено 6000 систем. Одна из последних систем Maestro I находится в Музее информационных технологий в Арлингтоне.



Рисунок 2 - Клавиатура Maestro I

В 1980-х годах IDE стали более популярными благодаря развитию персональных компьютеров. Например, Turbo Pascal (1983 год) от компании Borland стал одной из первых IDE для ПК. Он включал в себя текстовый редактор, компилятор и отладчик, что значительно упрощало процесс разработки.

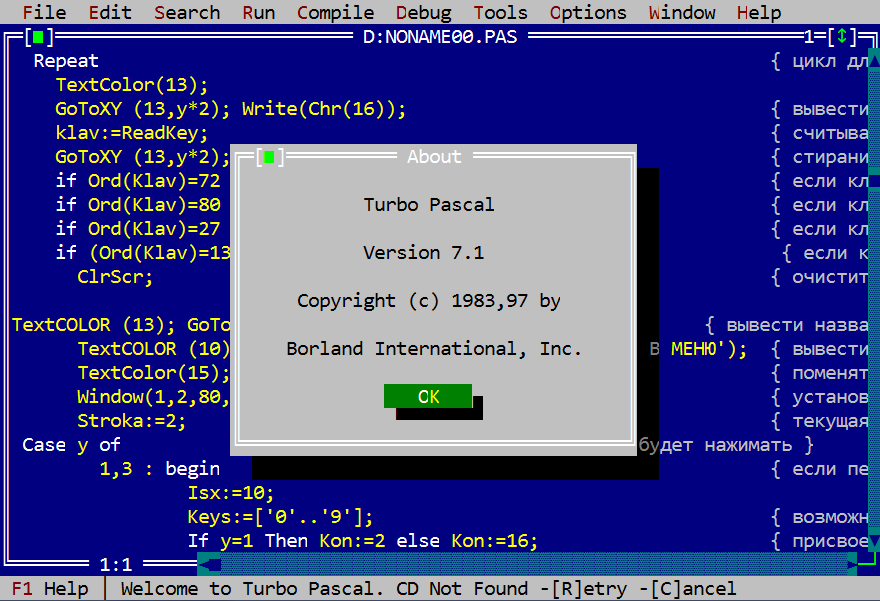


Рисунок 3 - Среда разработки Turbo Pascal 7.0

1.2.3 Расцвет IDE (1990-е годы)

В 1990-х годах IDE стали стандартом для разработки программного обеспечения. Появились мощные инструменты, такие как Microsoft Visual Studio (1997 год), который поддерживал несколько языков программирования (C++, Visual Basic) и предлагал богатый набор функций, включая графический интерфейс для создания приложений.

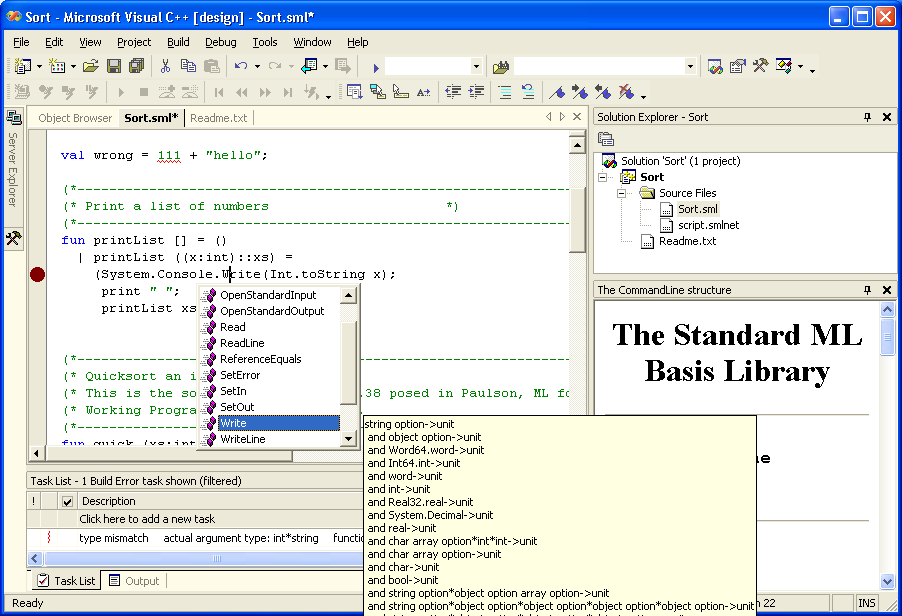


Рисунок 4 - Microsoft Visual Studio

Другим важным событием стало появление Eclipse (2001 год), открытой IDE, изначально разработанной для Java, но позже расширенной для поддержки других языков.

В этот период также появились специализированные IDE, такие как Delphi (1995 год) для разработки на языке Pascal и CodeWarrior для разработки игр и приложений на C++.

1.2.4 Современные IDE (2000-е годы – настоящее время)

В 2000-х годах IDE стали еще более мощными и универсальными. Они начали поддерживать множество языков программирования, интеграцию с системами контроля версий (например, Git), автоматическое завершение кода, рефакторинг и другие продвинутые функции.

IntelliJ IDEA (2001 год) стала одной из самых популярных IDE для Java, а позже и для других языков благодаря своей модульной архитектуре.

Visual Studio Code (2015 год) от Microsoft стал одним из самых популярных редакторов кода благодаря своей легкости, расширяемости и поддержке множества языков.

Современные IDE, такие как PyCharm (для Python), Xcode (для разработки под macOS и iOS) и Android Studio (для разработки под Android), предлагают специализированные инструменты для конкретных платформ и языков.

1.2.5 Будущее IDE

Современные IDE активно интегрируют технологии искусственного интеллекта для автоматического завершения кода, поиска ошибок и оптимизации производительности.

Развитие облачных технологий привело к появлению облачных IDE, таких как GitHub Codespaces и Replit, которые позволяют разрабатывать код прямо в браузере.

Увеличивается поддержка распределенной разработки и collaboration-инструментов, что делает IDE более удобными для командной работы.

Таким образом, история IDE отражает эволюцию программирования: от простых текстовых редакторов до мощных интегрированных сред, которые сегодня являются неотъемлемой частью разработки программного обеспечения.

* 1. Используемый стек технологий

Для разработки интегрированной среды разработки были использованы компоненты, соответствующие современным требованиям и обеспечивающее высокую производительность. В данном разделе будет идти речь об основных компонентах разработки.

1.3.1 Язык программирования C++



Рисунок 00 – Логотип С++

С++ компилируемый, статически типизированный, объектно-ориентированный язык программирования.

С++ - это компилируемый статически типизированный язык программирования общего назначения. C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также компьютерных игр. Существует множество реализаций языка C++, как бесплатных, так и коммерческих и для различных платформ. Например, на платформе x86 это GCC, Clang, Visual C++, Intel C++ Compiler, Embarcadero (Borland) C++ Builder и другие. C++ оказал огромное влияние на другие языки программирования, в первую очередь на Java и C#.

C++ появился как развитие языка C, который был создан в начале 1970-х годов Деннисом Ритчи в Bell Labs. C стал популярным благодаря своей эффективности, низкоуровневым возможностям и переносимости. Однако с ростом сложности программ возникла необходимость в более высокоуровневых конструкциях.

1979: Бьёрн Страуструп, датский программист, начал работу над расширением языка C, добавив в него возможности объектно-ориентированного программирования (ООП). Он назвал этот язык "C with Classes" ("C с классами").

1983: Язык был переименован в C++. Название отражает использование оператора инкремента ++, что символизирует "улучшение" или "эволюцию" языка C.

1985: Вышло первое издание книги Страуструпа "The C++ Programming Language", которая стала основным руководством по языку.

1989: Выпущена версия C++ 2.0, которая добавила множественное наследование, абстрактные классы и другие важные функции.

1998: Первый международный стандарт C++ — C++98. Он включал стандартную библиотеку шаблонов (STL), которая предоставила мощные инструменты для работы с контейнерами и алгоритмами.

2003: Выпущен стандарт C++03, который внёс незначительные исправления в C++98.

2011: Вышел стандарт C++11, который стал крупным обновлением. Он добавил поддержку лямбда-выражений, умных указателей, многопоточности и других современных возможностей.

2014, 2017, 2020: Последующие стандарты (C++14, C++17, C++20) продолжали улучшать язык, добавляя новые функции, такие как концепции, модули, корутины и улучшенная поддержка многопоточности.

C++ остаётся одним из самых популярных языков программирования благодаря своей производительности, гибкости и широкой области применения. Он используется в:

Разработке игр (Unreal Engine, Unity).

Системном программировании (операционные системы, драйверы).

Высокопроизводительных вычислениях (HPC).

Встраиваемых системах (микроконтроллеры, IoT).

Финансовых технологиях и научных исследованиях.

C++ продолжает развиваться, сохраняя баланс между низкоуровневым контролем и высокоуровневыми абстракциями, что делает его универсальным инструментом для современных разработчиков.

Язык программирования C++ был выбран в качестве основного языка разработки благодаря следующим преимуществам:

Высокая производительность: C++ позволяет создавать высокоэффективные приложения, что критически важно для IDE, где важна скорость обработки кода и отклика интерфейса.

Низкоуровневый контроль: C++ предоставляет возможность управления памятью и ресурсами.

Широкая поддержка библиотек: C++ имеет множество библиотек и Фреймворков, которые упрощают разработку сложных приложений.

1.3.2 Фреймворк Qt

Qt — Фреймворк для разработки кроссплатформенного программного обеспечения на языке программирования C++. Для многих языков программирования существуют библиотеки, позволяющие использовать преимущества Qt: Python — PyQt, PySide; Ruby — QtRuby; Java — QtJambi PHP — PHP-Qt и другие.

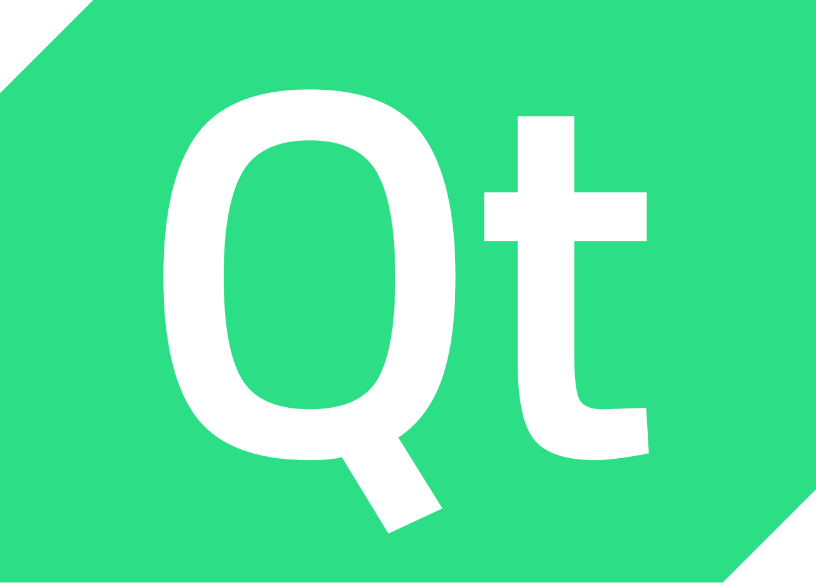


Рисунок 00 – Логотип Qt

1991: Норвежские программисты Хаавард Норд и Эйрик Чамбе-Энг начали разработку фреймворка для создания графических интерфейсов. Они основали компанию Trolltech.

1995: Выпущена первая версия Qt. Изначально Qt был разработан для Unix-систем, но быстро стал кроссплатформенным, поддерживая Windows и macOS.

1996: Qt стал популярным среди разработчиков благодаря своей простоте и мощности. Он использовался для создания приложений с графическим интерфейсом на C++.

2000: Выпущена Qt 2.0, которая добавила поддержку Unicode, улучшенную работу с сетью и новые виджеты.

2005: Вышла Qt 4.0, которая стала важным этапом в развитии фреймворка. Она представила модульную архитектуру, улучшенную поддержку графики и новую систему обработки событий.

2008: Компания Nokia приобрела Trolltech, чтобы использовать Qt для разработки программного обеспечения для своих мобильных устройств. Это привело к активному развитию Qt для платформ смартфонов.

2012: После ухода Nokia из мобильного рынка права на Qt были проданы компании Digia.

2014: Digia разделила Qt на два подразделения: Qt Project (открытая часть) и The Qt Company (коммерческая поддержка).

2016: The Qt Company стала независимой и сосредоточилась на развитии фреймворка и поддержке коммерческих клиентов.

Qt 5: Выпущен в 2012 году. Основные нововведения:

Поддержка QML (язык для создания динамических интерфейсов).

Улучшенная производительность и поддержка современных платформ.

Интеграция с OpenGL и 3D-графикой.

Qt 6: Выпущен в декабре 2020 года. Основные изменения:

Улучшенная поддержка C++17 и C++20.

Новый графический движок RHI (Render Hardware Interface).

Упрощённая архитектура и улучшенная производительность.

Поддержка современных стандартов, таких как Vulkan и Metal.

1.3.3 Библиотека QScintilla

Для реализации текстового редактора с подсветкой синтаксиса и автодополнением используется библиотека QScintilla, которая является портом библиотеки Scintilla для Qt. Основные функции QScintilla:

Поддержка подсветки синтаксиса для различных языков программирования.

Автодополнение кода и подсказки.

Возможность настройки цветовых схем и шрифтов.

Поддержка отступов, нумерации строк и других функций, необходимых для удобного редактирования кода.

1.3.3 HTML

HTML – это язык гипертекстовый разметки, используемый для создания интернет-страниц и структурирования контента на интернет-страницах. HTML позволяет создавать различные элементы с помощью тегов, которые определяют, как содержимое страницы должно отображаться в браузере. С помощью добавления вы можете создавать гиперссылки, изображения, таблицы, формы и многое другое.



Рисунок ? – Логотип HTML

1.3.4 CSS

CSS (Cascading Style Sheets) – это язык стилей, который используется для оформления веб-страниц. Он позволяет определить внешний вид элементов HTML-документа, таких как текст, изображения, таблицы и другие, а также управлять их расположением на странице.



Рисунок ? – Логотип CSS

1.3.5 CMake

CMake (МФА (cross-platform make) — кроссплатформенное программное средство автоматизации сборки программного обеспечения из исходного кода. Не занимается непосредственно сборкой, а лишь генерирует файлы сборки из предварительно написанного файла сценария CMakeLists.txt и предоставляет простой единый интерфейс управления. Помимо этого, способно автоматизировать процесс установки и сборки пакетов.

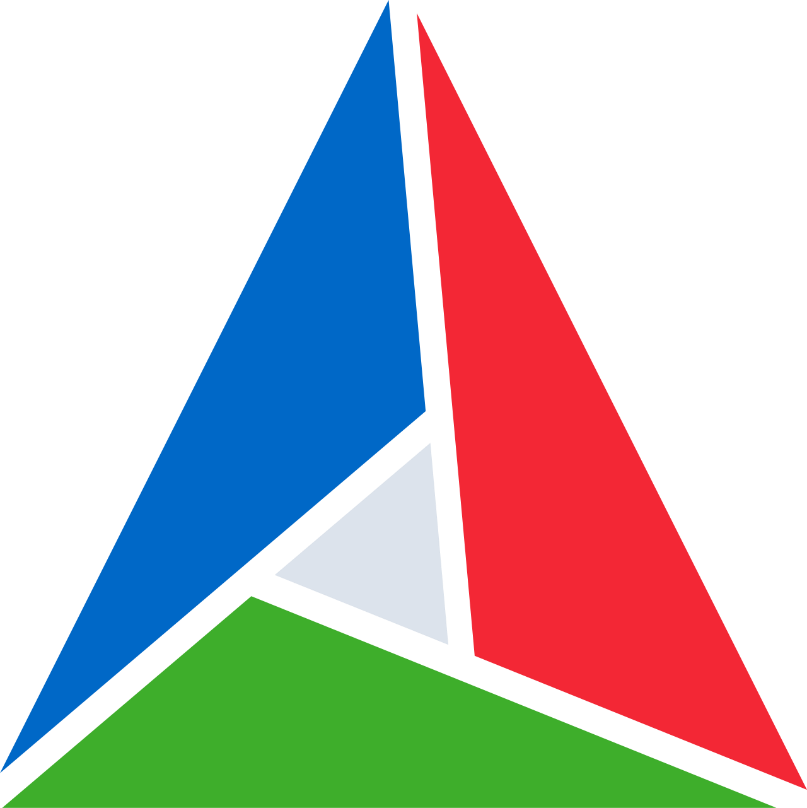


Рисунок ? – Логотип CMake

Считается альтернативой распространённой в сообществе GNU системе Autotools, разработанной на базе Perl и M4, основными недостатками которой считаются необходимость нетривиальных навыков для практического использования и несовместимость версий в ряде случаев.

В сравнении с другой альтернативой Autotools — основанной на Python системой SCons — является более быстродействующей, поскольку написана на Си и использует крайне простой макроязык, но при этом SCons обладает большими возможностями по расширению.

1.3.6 Git

Git - это распределенная система контроля версий, которая используется разработчиками по всему миру для управления своими проектами. Она предоставляет ряд важных преимуществ и функциональности, которые делают Git незаменимым инструментом в современной разработке программного обеспечения. Вот основные причины, по которым Git необходим:

* Позволяет отслеживать изменения в файлах проекта, создавая "снимки" (коммиты) на различных этапах разработки.
* Использует децентрализованную модель, где каждый разработчик имеет локальную копию всего репозитория.
* Предоставляет мощные инструменты для создания и управления ветками (branches).
* Облегчает совместную работу над проектом, позволяя нескольким разработчикам вносить изменения в один репозиторий.
* Ведет подробную историю всех изменений, включая автора, дату, комментарии и другую полезную информацию.
* Git легко адаптируется к различным рабочим процессам и методологиям разработки.

Git является мощным и гибким инструментом, который помогает разработчикам управлять версиями своих проектов, координировать совместную работу, отслеживать изменения и обеспечивать надежность и масштабируемость разрабатываемого ПО.



Рисунок 7 – Логотип сервиса git

1.3.7 GNU Compiler Collection

GNU Compiler Collection (GCC) — набор компиляторов для различных языков программирования, разработанный в рамках проекта GNU. GCC является свободным программным обеспечением, распространяется в том числе фондом свободного программного обеспечения (FSF) на условиях GNU GPL и GNU LGPL и является ключевым компонентом GNU toolchain. Он используется как стандартный компилятор для свободных UNIX-подобных операционных систем.

Изначально названный GNU C Compiler поддерживал только язык Си. Позднее GCC был расширен для компиляции исходных кодов на таких языках программирования, как C++, Objective-C, Java (исключена из состава GCC начиная с версии 7 в 2017 году[5]), Фортран, Ada, Go, GAS и D.

В базовую поставку компилятора входят такие программы:

libc6-dev - заголовочные файлы стандартной библиотеки Си;

libstdc++6-dev - заголовочные файлы стандартной библиотеки С++;

gcc - компилятор языка программирования Си;

g++ - компилятор языка программирования C++;

make - утилита для организации сборки нескольких файлов;

dpkg-dev - инструменты сборки пакетов deb.



Рисунок? – Логотип GCC

# 2 Основная часть

# 2.1 Функции и макеты

2.1.1 Определение функций интегрированной среды разработки

В данной части работы будут рассмотрены базовые и основные функции IDE также необходимо определиться с базовым макетом интерфейса. Большинство фреймворков в том числе и Qt заставляют придерживается определённой структуры проекта. Основной графический интерфейс необходимо разработать.

Современная интегрированная среда разработки должна иметь определённый минимальный функционал:

-Наличие текстового редактора;

-Возможность просмотра, создания и удаления файлов редактируемого проекта;

-Возможность запуска редактируемого проекта;

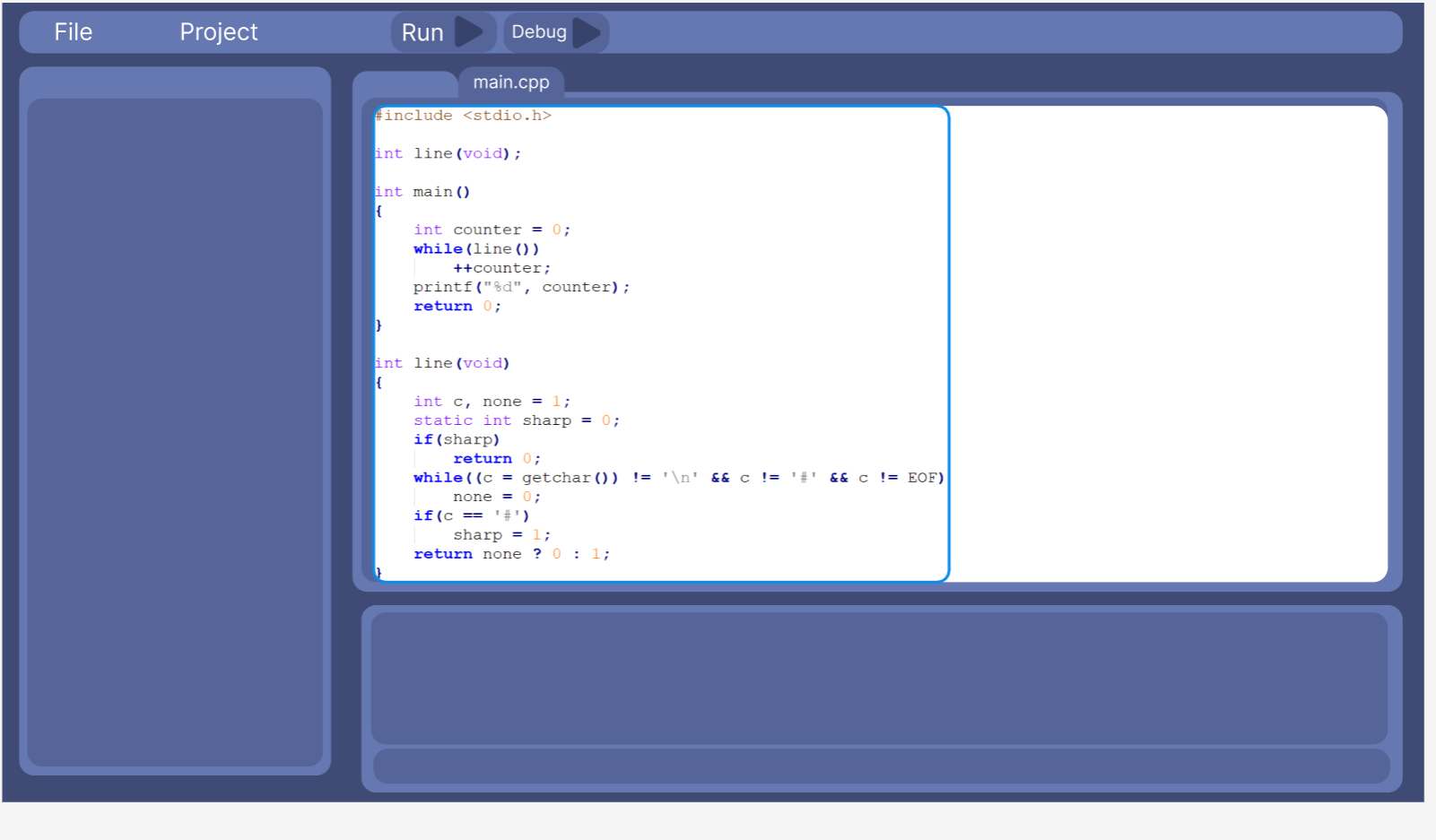
-Наличие терминала для отображения работы редактируемого проекта;

-хз.

2.1.2 Проектирование макета основного графического интерфейса.

После определения основных функций программы необходимо разработать макет самой рабочей среды, в которой пользователь будет находится большую часть времени. Для этого будет использовано веб приложение figma с помощью которого можно быстро создать необходимый макет. Функционал данного приложения предоставляет все необходимые функции, для проектирования различных приложений и вебсайтов. Также данное приложение используется дизайнерами по всему миру.

На данном макете расположены основные функции, которые необходимо удобно расположить на главной панели для удобного доступа. Раздел файлы в котором будут находится функции необходимые для работы с файлами. Раздел проект, в котором будут находится функции необходимые для редактирования и настройки конфигурации проекта. Кнопки Run и Debug будут запускать интерпретатор или компилятор в зависимости от проекта. На правой панели будут отображается список фалов проекта. На центральной панели будет происходить написание и редактирование кода. Нижняя панель является терминалом для отображения работы программы.



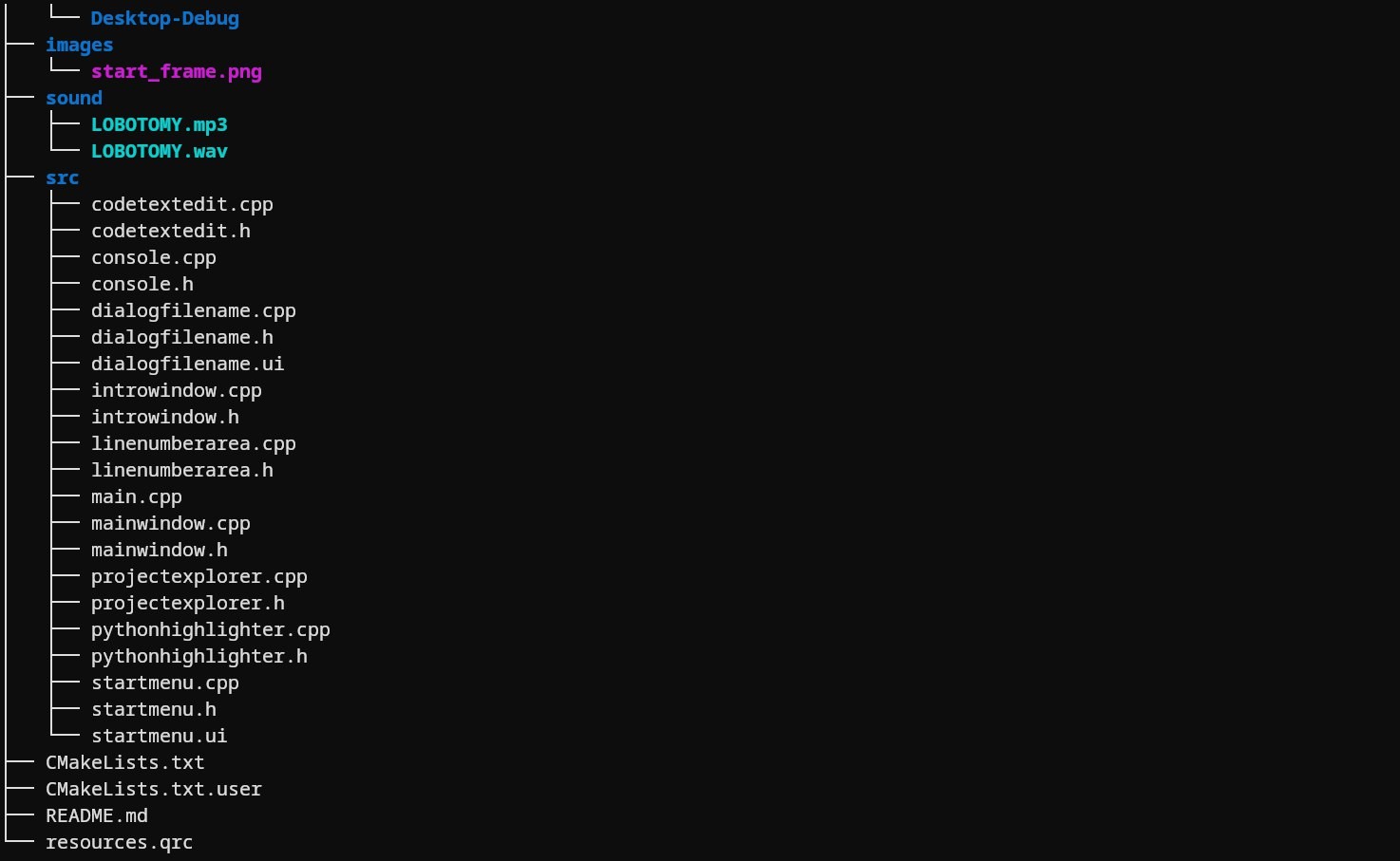
2.2 Разработка интегрированной среды разработки.

Разработка любого проекта начинается с определения архитектуры проекта. Поскольку Qt устанавливает некоторые требования для реализации проекта досконально продумывать всю архитектуру не требуется. Графический интерфейс в Qt может реализовываться либо статично через QML, либо динамически через создание экземпляров классов и расположение их на главном окне, будет использован гибридный подход.

2.1.3 Разработка Структуры проекта

При запуске приложения пользователь увидит небольшое окно, в котором он может выбрать какой проект создать или открыть уже созданный ранее. Далее уже следует работа в самом приложении.

Структура программы представляет собой сл. Схему:



Структура исходного кода состоит из исполняемого файла main.cpp и файла класса mainwindow.cpp который представляет собой главное окно.

Директория images хранит в себе различные спрайты, иконки и изображения для отображения их в интерфейсе. Директория sound хранит в себе различные звуки для воспроизводимые во время работы. Директория src хранит в себе исходный код программы основные, заголовочные и файлы qml.

Главным исполняемым файлом является main.cpp внутри него находится цикл работы программы который прекращается после её закрытия. Файл mainwindow.cpp является файлом класом главного окна, на котором будут отображается все элементы, виджеты и кнопки, экземпляр класса создаётся внутри main.cpp и существует пока программа работает, а после завершения вызывается деструктор. Внутри класса mainwindow будут вызываться другие классы такие как console, progectexplorer..Таким образом будет выглядеть структура проекта.

2.2 Подготовка рабочей среды

Перед началом разработки необходимо подготовить рабочую среду, установить все необходимые системы и установить зависимости. Все программы и библиотеки будут загружается через пакетный менеджер, все команды будут писаться на примере команд для Advanced Packaging Tool программа для установки, обновления и удаления программных пакетов в операционных системах Debian и основанных на них дистрибутивах (например: Ubuntu, Linux Mint и т. п.), иногда также используется в дистрибутивах, основанных на Mandrake, например Mandriva, ALT Linux и PCLinuxOS.

1 Проверка наличия и установка компилятора gcc

Для большинства UNIX подобных систем свойственно иметь компилятор в составе программ по умолчанию. Для проверки наличия компилятора необходимо прописать:

“gcc -- version”

После этого gcc сообщит свою версию.

2 Установка CMake

Для удобства сбоки проекта необходим CMake.

“sudo apt install cmake”

3 Установка Qt creator

Для удобной работы с qt необходима специальная среда разработки Qt creator которая предоставляет удобный функционал для разработки и работы с qt на C++ и Python. Также Qt creator самостоятельно устанавливает необходимые зависимости для создания и работы над проектом. Для начала загрузить библиотеки qt для C++:

“sudo apt install qt6-default”

Затем загрузить саму Qt Creator:

“sudo apt install qtcreator”

4Установка Git

Для установки Git