Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Операционные системы и системы программирования (ОСиСП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему:

«Среда разработки на языке Brainfuck»

БГУИР КП 1-40 01 01 001 ПЗ

Студент: гр. 651001 Азаров А. А.

Руководитель: Базылев Е. Н.

Минск 2018

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

––––––––––––––––––––––––

(подпись)

2018 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту   Азарову Александру Александровичу ––––––––––––

1. Тема работы   Среда разработки на языке программирования Brainfuck– –––

2. Срок сдачи студентом законченной работы 20.12.2018

3. Исходные данные к работе   файл с исходными данными

4. Содержание расчётно-пояснительной записки:

*Введение.*

*1.Аналитический обзор литературы и существующих аналогов;*

*2.Разработка алгоритма;*

*3. Разработка программного средства;*

*4. Обоснование технических приемов программирования;*

*5. Тестирование, экспериментальные исследования и анализ полученных результатов;*

*6. Руководство пользователя программы;*

*Заключение, список литературы, ведомость, приложения.*

5. Перечень графического материала:

1*. Схема программы*

6. Консультант по курсовому проекту Базылев Е.Н.

7. Дата выдачи задания 07.09.2018 г.––   –

8. Календарный график работы над проектом на весь период проектирования:

раздел 1, введение к 15.09.2018 – 10 % готовности работы;

разделы 2 к 15.10.2018 – 30 % готовности работы;

разделы 3,4 к 15.11.2018 – 60 % готовности работы;

раздел 5, 6 к 15.12.2018 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 20.12.2018 – 100 % готовности работы.

Защита курсового проекта с 20.11 по 20.12 2018 г.

РУКОВОДИТЕЛЬ Е.Н. Базылев

(подпись)

Задание принял к исполнению –––\_\_\_\_–– А.А. Азаров 07.09.2018 г.

(дата и подпись студента)

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ. 5

1 АНАЛИТИЧЕСКИЙ ОБЗОР ЛИТЕРАТУРЫ 7

1.1 Развитие языков программирования 7

1.2 Анализ существующих интерпретаторов и сред разработки 10

1.3 Постановка задачи 14

2 РАЗРАБОТКА АЛГОРИТМА 15

2.1 Разработка архитектуры приложения 15

2.2 Разработка алгоритма процесса интерпретации программы 15

3 разработка программного средства 18

3.1 Инструменты и технологии, используемые в разработке 18

3.2 Разработка используемых структур данных 22

4 тестирование 24

5 Руководство пользователя 26

Заключение 28

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 29

приложение А. КОД Программы 30

**ВВЕДЕНИЕ**

Языки программирования — это основной инструмент каждого инженера-программиста. На данный момент по разным оценкам существует около десяти тысяч различных языков. По разным критериям их все можно разбить в разные категории: например, по парадигмам можно выделить процедурные (С), объектно-ориентированные (Java), функциональные (Haskell). Существует отдельная категория языков программирования, которые принято называть эзотерическими. Они могли быть разработаны для исследования возможностей разработки языков, для доказательства потенциально возможной реализации некоторой идеи или даже в качестве шутки. В целом код на таком языке программирования будет понятен лишь «посвящённому». На практике такие языки зачастую бесполезны, но программирование на них или создание их интерпретатора/компилятора может стать неплохой тренировкой.

Brainfuck один из наиболее известных эзотерических языков программирования, который даже породил целый класс себе подобных. Основной идеей в них является сокращение синтаксиса при сохранении Тьюринг-полноты. Полнота по Тьюрингу — это способность некого языка программирования реализовать любую вычислимую функцию.

Данная курсовая работа посвящена разработке собственного интерпретатора эзотерического языка программирования Brainfuck и созданию среды разработки с графическим интерфейсом, который реализуется с помощью функций WinAPI предоставляемых операционной системой Windows. В качестве языка программирования для курсового проекта был выбран язык ассемблера в синтаксисе FASM, так как он является низкоуровневым языком программирования и позволяет откинуть все слои абстракции и работать с WinAPI напрямую

В ходе выполнения данного проекта я постараюсь понять:

1. Как работают оконные приложения в ОС Windows
2. Какие форматы исполняемых файлов существуют в данной ОС
3. Как работать с WinAPI с использованием языка ассемблера
4. Как отлаживать программы, написанные для ОС Windows
5. Как написать собственный интерпретатор языка программирования Brainfuck

В этой пояснительной записке отображены следующие этапы написания курсовой работы:

1. Аналитический обзор существующих аналогов. Формирование требований к разрабатываемому программному средству.
2. Моделирование предметной области. Выбор средств разработки.
3. Разработка архитектуры программного средства и его функциональных возможностей.
4. Тестирование и проверка работоспособности программного средства. Анализ полученных результатов.
5. Руководство по использованию программного средства. Включает в себя последовательность действий, обеспечивающих успешное использование данного программного средства.

**1. АНАЛИТИЧЕСКИЙ ОБЗОР ЛИТЕРАТУРЫ**

**1.1 Развитие языков программирования**

Основное предназначение языка программирования – написание компьютерных программ, которые представляют собой набор правил, позволяющих компьютеру выполнить тот или иной вычислительный процесс, принимающий какие-то входные данные и выдающий свой результат работы. Язык программирования отличается от естественных языков тем, что предназначен для управления ЭВМ, в то время как естественные языки используются, прежде всего, для общения людей между собой. Большинство языков программирования использует специальные конструкции для определения и манипулирования [структурами данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и управления процессом вычислений. Как правило языки программирования определяются не только через спецификации языка, который формально определяет его синтаксис, но и через различные воплощения стандарта, которые могут отличатся по времени выпуска, дополнительным возможностям, могут даже иметь ошибки и особенности воплощения.

Одним из первых языков программирования можно считать «язык» на котором Ада Лавлейс в 1842 году написала программу для вычисления чисел Бернулли для вычислительной машины Чарльза Бэббиджа. В 1930-1940 годах Алан Тьюринг разработал математические абстракции для описания алгоритмов, например машину Тьюринга. Примерно в то же время появились электрические цифровые компьютеры и был разработан язык, который можно считать первым высокоуровневым языком программирования — Планкалкюль. Стоит отметить что он поддерживал операции присваивания, вызов подпрограмм, условные операторы, итерационные циклы, арифметику с плавающей запятой, массивы, иерархические структуры данных, и многие другие вполне современные средства языков программирования.

В целом языки программирования принято делить на пять поколений. К первому поколению относят машинные языки – языки программирования на уровне команд процессора конкретной машины. Для программирования не использовался транслятор, команды вводились непосредственно в машинном коде. В частности языки ассемблера принято относить к первому поколению языков программирования.

Языки второго поколения создавались для того чтобы облегчить работу программиста, перейдя в выражениях языка от низкоуровневых машинных понятий ближе к тому, как обычно мыслит программист. Скептицизм по поводу возможности создания эффективных программ с помощью автоматических компиляторов был довольно распространён, поэтому разработчикам таких систем должны были продемонстрировать, что они действительно могут генерировать почти такой же эффективный код, как и при ручном кодировании, причём практически для любой исходной задачи.

Под третьим поколением понимаются все языки более высокого уровня, чем ассемблер. Главной отличительной чертой языков третьего поколения стала независимость от аппаратного обеспечения, то есть выражение алгоритма в форме, не зависящей от конкретных характеристик машины, на которой он будет исполняться. Широко стала использоваться интерпретация программ – при этом инструкции программы не преобразуются в машинный код, а исполняются непосредственно одна за другой.

В 1990-х годах в связи с активным развитием Интернета распространение получили языки, позволяющие создавать сценарии для веб-страниц — главным образом Perl, развившийся из скриптового инструмента для Unix-систем, и Java. Возрастала также и популярность технологий виртуализации. Эти изменения, однако, также не представляли собой фундаментальных новаций, являясь скорее совершенствованием уже существовавших парадигм и языков (в последнем случае — главным образом семейства Си).

В настоящее время развитие языков программирования идёт в направлении повышения безопасности и надёжности, создания новых форм модульной организации кода и интеграции с базами данных.

Brainfuck был разработан в 1993 году немецким исследователем Урбаном Мюллером. Одним из его мотивов было создание языка с как можно меньшим компилятором. Машина которой управляют команды Brainfuck, состоит из набора ячеек и указателя текущей ячейки, по аналогии с машиной Тьюринга. Кроме того в языке есть операторы ввода вывода («,» и «.»).

|  |  |
| --- | --- |
| Команда Brainfuck | Описание команды |
| **>** | Перейти к следующей ячейке |
| **<** | Перейти к предыдущей ячейке |
| **+** | Увеличить значение в текущей ячейке на 1 |
| **–** | Уменьшить значение в текущей ячейке на 1 |
| **.** | вывести значение из текущей ячейки |
| **,** | ввести извне значение и сохранить в текущей ячейке |
| **[** | если значение текущей ячейки ноль, то перейти вперёд по тексту программы на ячейку, следующую за соответствующей **]** |
| **]** | если значение текущей ячейки не ноль, то перейти назад по тексту программы на символ **[** |

Несмотря на свою примитивность, Brainfuck с бесконечным набором ячеек имеет тьюринговскую полноту, а, следовательно, по потенциальным возможностям не уступает языкам вроде C, Pascal или Java.

Таким образом, данный язык достаточно прост в исполнении и хорошо подходит на роль языка для которого можно целиком написать свою среду разработки. Так как основной задачей курсового проектирования показать навыки работы с WinAPI, не желательно тратить много времени на реализацию интерпретатора языка.

**1.2 Анализ существующих интерпретаторов и сред разработки**

Так как тема курсовой работы была выбрана в первую очередь для получения знаний в области низкоуровневого программирования и взаимодействия с WinAPI, не ставится цель создать конкурентоспособный продукт. Тем не менее неплохо иметь представление об существующих средах разработки системах, о реализованных там функциях, об их достоинствах и недостатках.

**Visual Studio –**одна из самых популярных IDE на данный момент. Её интерфейс изображён на рисунке 1.1.

Основные отличия Windows:

1. При программировании под ОС Windows практически вне конкуренции, ведь разрабатывается той же компанией — Microsoft
2. Огромное сообщество пользователей

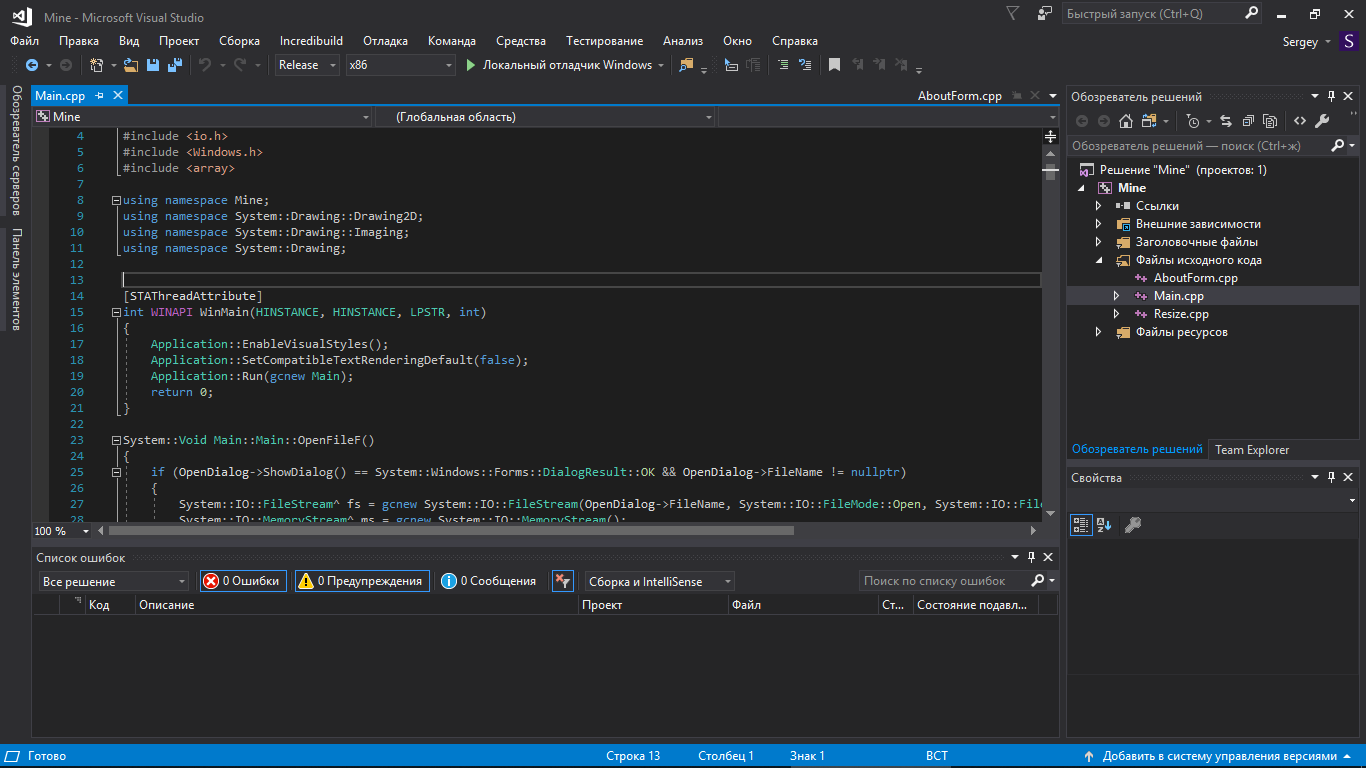


Рисунок 1.1 – Скриншот Visual Studio

Плюсы:

1. Понятный интерфейс и простота использования
2. Широкая распространённость и большое количество доступных мануалов
3. Многоязычность (в том числе русский язык)

Минусы:

1. Высокая стоимость не Community версии
2. Закрытый исходный код
3. Есть версия только под Windows

**Xcode –**среда разработки для платформ macOS, iOS, watchOS, tvOS, разработанная корпорацией Apple.

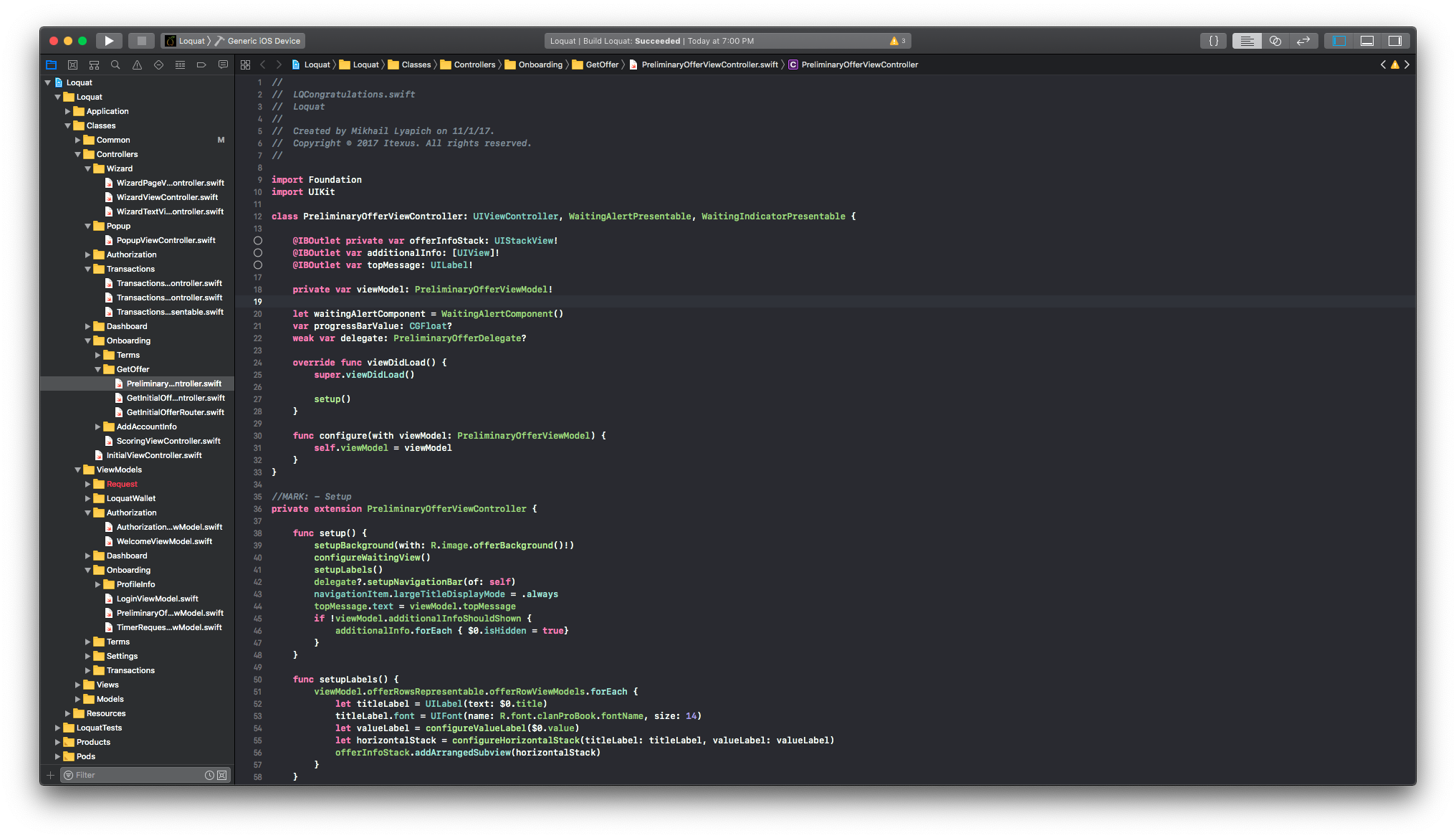


Рисунок 1.2 – Скриншот Xcode

Плюсы:

1. Удобный доступ к документации прямо из исходного кода
2. Распространяется бесплатно

Минусы:

1. Закрытый исходный код
2. Есть версия только для macOS

**https://copy.sh/brainfuck/–**редактор языка Brainfuck со встроенным интерпретатором, написанный на JavaScript.

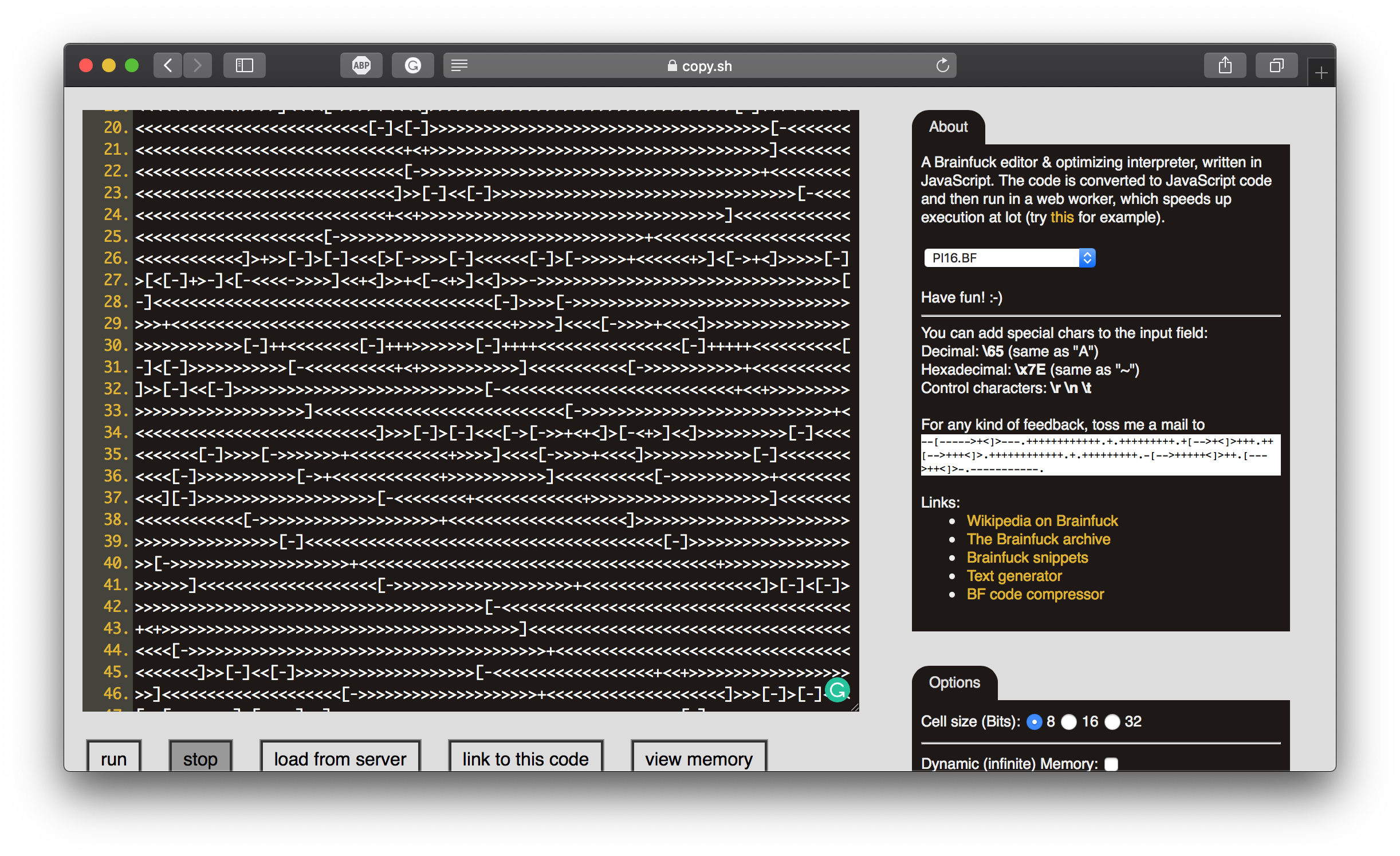


Рисунок 1.3 – Скриншот сайта

Плюсы:

1. Подборка примеров программ на языке Brainfuck
2. Доступен бесплатно

Минусы:

1. Требуется интернет-подключение
2. По скорости исполнения проигрывает нативным приложениям, ведь движок написан на JavaScript.

**1.3 Постановка задачи**

После изучения истории развития языков программирования и информации о современных средах разработки становится очевидно, что полноценные IDE разрабатываются крупными компаниями или сообществами энтузиастов на протяжении нескольких десятилетий. Так как в моём случае время ограничено, логичным представляется написать среду разработки для такого простого языка программирования как Brainfuck.

Для того, чтобы данный проект можно было считать полноценной средой разработки, должны быть реализованы следующие функции:

1. Отображение графического интерфейса используя функции WinAPI
2. Наличие поля для ввода кода на языке программирования Brainfuck
3. Наличие поля для ввода входных данных для исполняемой программы
4. Возможность исполнения программы
5. Вывод результата исполнения программы в отдельное поле
6. Возможность открывать файлы в формате **.bf** и **.b**

В качестве языка программирования выбран язык ассемблера FASM, так как я уже освоил его, выполняя лабораторные работы и курсовой проект по предмету Языки Программирования. К тому же разработку на других языках программирования снять слой абстракций и работать с WinAPI напрямую. FASM компилируется сразу в машинный код, что позволяет чётко контролировать процесс работы программы и интерпретатора на самом низком уровне.

Среда разработки должна предоставлять интуитивно понятный пользовательский интерфейс на английском языке, что дает возможность использования данного курсового в образовательных целях в любой точке мира.

**2 РАЗРАБОТКА АЛГОРИТМА**

**2.1** **Разработка архитектуры приложения**

На стадии проектирования приложения основной концепцией работы было выбрано выполнение выбранных пользователем действий. Изначально пользователю необходимо ввести исходный код программы. Для этого нужно создать окно и текстовое поле для ввода. Желательно использовать моноширинный шрифт, так как предполагается что это улучшит читаемость кода. Если нажать кнопку «Run», то интерпретатор запустит выполнение программы. Перед этим пользователь может опционально указать входные данные для программы. Так же следует предусмотреть возможность открывать файлы на жёстком диске, так как написание программы с нуля на Brainfuck достаточно трудоёмкая задача.

Укрупненная схема программы представлена на рисунке 2.1.

**2.2 Разработка алгоритма процесса интерпретации программы**

После нажатия пользователем кнопки “Run” происходит процесс интерпретации пользовательской программы. В классическом Brainfuck, описанном Мюллером, размер ячейки – один байт, количество ячеек 30 000. В начальном состоянии указатель находится в крайней левой позиции, а все ячейки заполнены нулями. Увеличение/уменьшение значений ячеек происходит по модулю 256. Ввод-вывод так же происходит побайтно, с учётом кодировки ASCII (то есть в результате операции ввода (,) символ 1 будет записан в текущую ячейку как число 0x31 (49), а операция вывода (.), совершённая над ячейкой содержащей 0x41 (65), напечатает латинскую А). Исходя из этих правил можно составить алгоритм интерпретации программного кода на языке Brainfuck. Для удобства работы с вводом/выводом удобно выделять буфер определённого размера для результатов, записывать их туда во время интерпретации программы, а в случае удачного выполнения выводить на экран за один раз. То же касается и пользовательского ввода, все входные данные должны быть известны в момент запуска, это ускорит процесс интерпретации.

Укрупнённая схема интерпретатора представлена на рисунке 2.2.

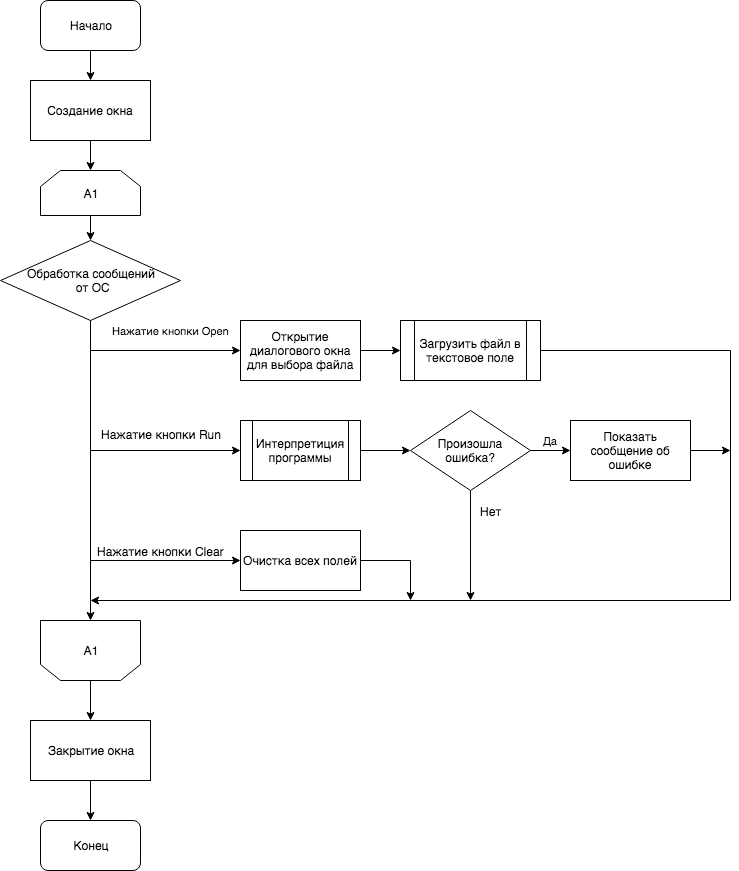


Рисунок 2.1 – Укрупнённая схема программы

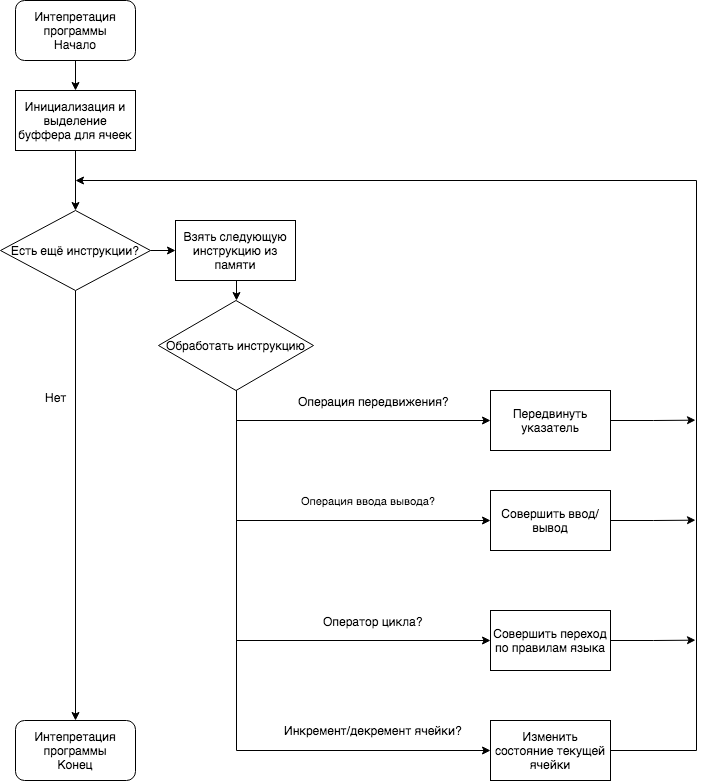


Рисунок 2.2 – Укрупнённая схема интерпретатора

**3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

**3.1 Инструменты и технологии, используемые в разработке**

Данное приложение подходит для персонального компьютера, работающего в операционной системе Windows. Программное средство не требует доступа к Интернету, используемые ресурсы полностью локализованы на устройстве.

* + 1. **Ассемблер FASM**

В качестве языка программирования для выполнения курсового проекта был выбран язык ассемблера, а точнее его синтаксис FASM. Это свободно распространяемый многопроходной ассемблер, написанный Томашем Грыштаром. Компиляция в FASM состоит из двух стадий: препроцессирование и ассемблирование. На стадии препроцессора раскрываются все макросы, символические константы, обрабатываются директивы препроцессора. На стадии ассемблирования определяютя адреса меток, обрабатываются условные директивы, раскрываются циклы и генерируется собственно программа. В FASM используется Intel-синтаксис записи инструкций. Единственное существенное отличие от формата, принятого в других ассемблерах (MASM, TASM) — значение ячейки памяти всегда записывается как *[label\_name],* а просто *label\_name* означает адрес (то есть порядковый номер) ячейки. Это позволяет обходиться без ключевого слова *offset*. Также в FASM при переопределении размера операнда вместо *byte ptr* пишется просто *byte*, вместо *word ptr — word* и т. д. Не позволяется использовать несколько квадратных скобок в одном операнде, таким образом вместо *[bx][si]* необходимо писать *[bx+si].*. Пример очень простой программы:

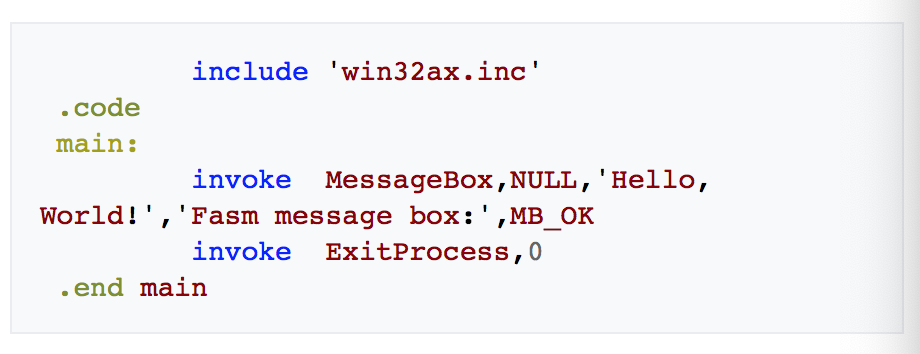


Рисунок 3.1 – Пример программы на языке ассемблера

* + 1. **Visual Studio Code**

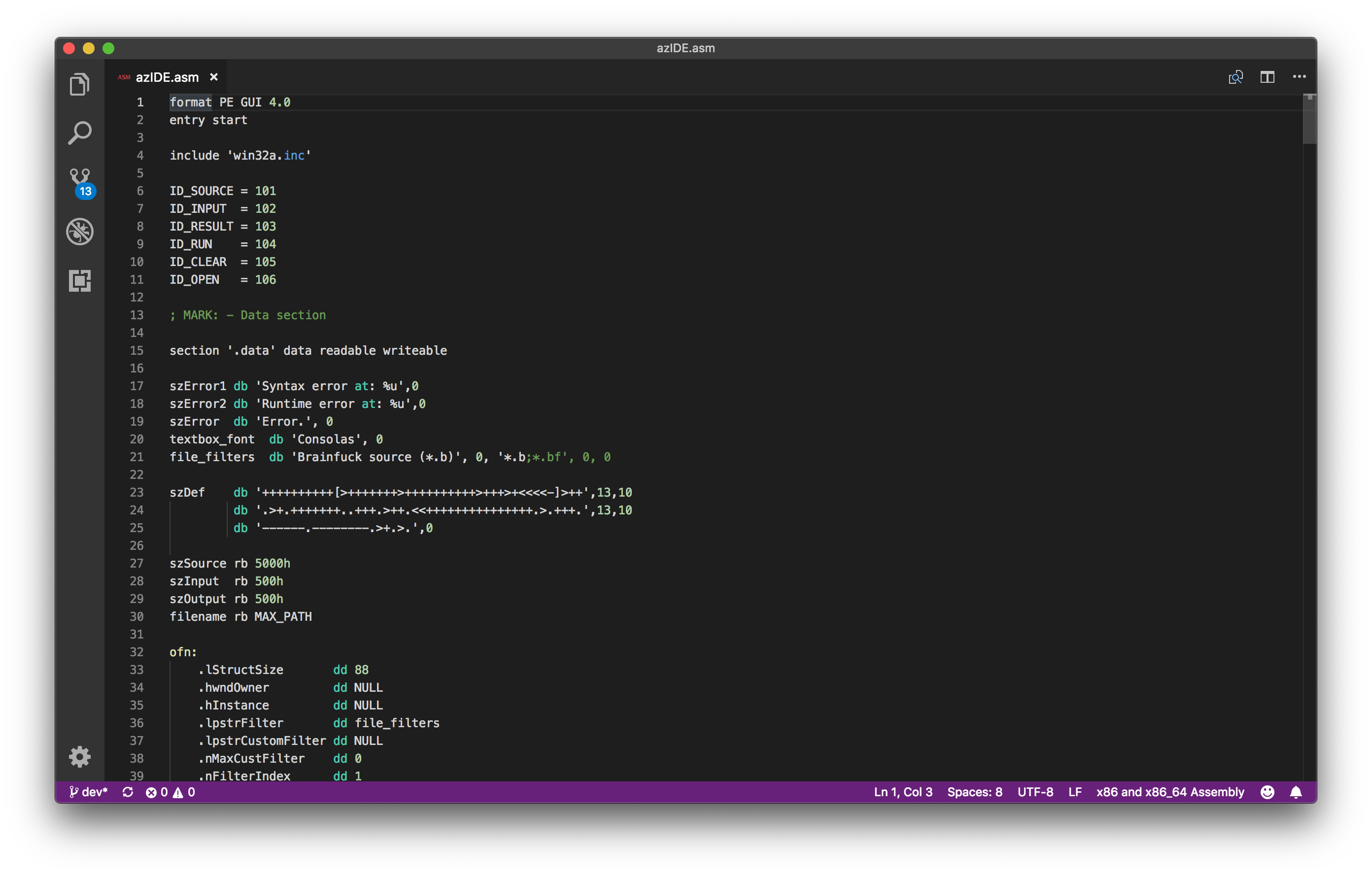
**Visual Sudio Code** – редактор исходного кода, разработанный Microsoft для Windows, Linux и macOS. Здесь стоит пояснить, что автор курсового проекта использует Apple Macbook под управлением macOS в качестве рабочей машины, разработка под Windows осложнена необходимостью устанавливать дополнительные средства виртуализации и работать в них. Это сильно сказывается на производительности системы и программиста в целом. Поэтому была создана виртуальная машина и каталог с общим доступом как из macOS, так и из виртуализированной Windows 10. Основной код пишется в macOS в редакторе Visual Studio Code, а специальный скрипт на виртуальной машине только запускает собирает приложение и пытается запустить. Для VSCode существует множество плагинов, даже для такого нераспространённого языка как ассемблер. 

Рисунок 3.2 – Интерфейс Visual Studio Code

* + 1. **IDA Disassembler**

Ни одно приложение сложнее “Hello, World!” невозможно написать, не используя средства отладки. На данный момент IDA является одним из самых мощных средств по дизассемблированию и реверс-инжинирингу приложений. Его возможности практически бесконечны, учитывая огромное сообщество пользователей и возможность писать собственные плагины на любые случаи жизни. Таким образом он идеально подходит под отладки приложения написанного на ассемблере для Windows.

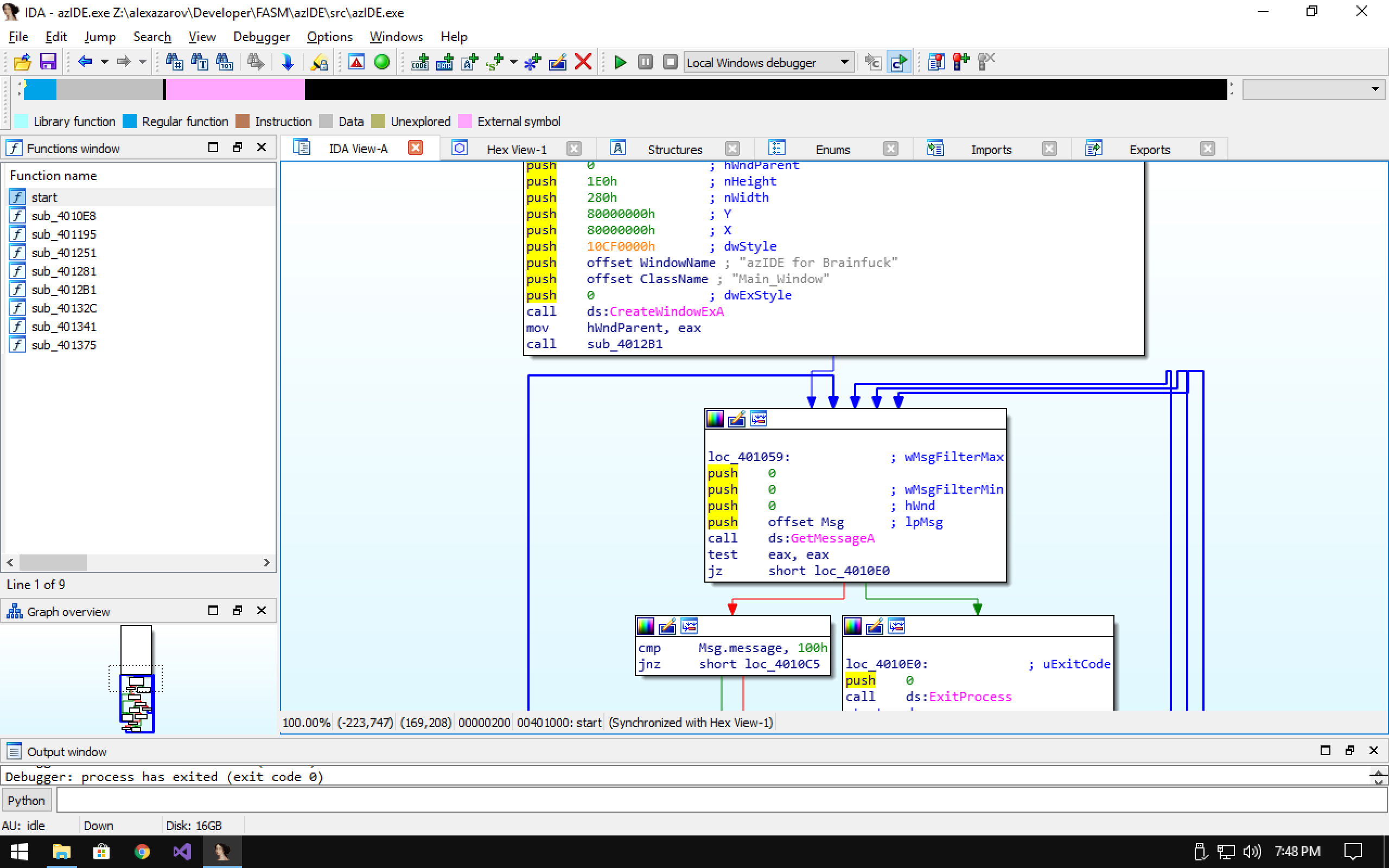


Рисунок 3.3 – Интерфейс IDA Disassemler

* + 1. **Автоматизация сборки**

Как уже было сказано ранее, для автоматической сборки приложения используется специальный скрипт с расширением *.bat*. Это текстовый файл, содержащий последовательность команд, предназначенных для исполнения командным интерпретатором. После запуска пакетного файла программа-интерпретатор (как правило, COMMAND.COM или cmd.exe) читает его строка за строкой и последовательно исполняет команды. Пакетный файл — аналог скриптовых файлов командной строки (shell script) в Unix-подобных операционных системах. Пакетные файлы полезны для автоматического запуска приложений. Основная область применения — автоматизация наиболее рутинных операций, которые регулярно приходится совершать пользователю компьютера: например, копирование, перемещение, переименование, удаление файлов; работа с папками; архивация и т. п. В данном случае они идеально подходят для автоматизации сборки и запуска приложения. На рисунке 3.4 представлен код скрипта использованного для сборки данного проекта.

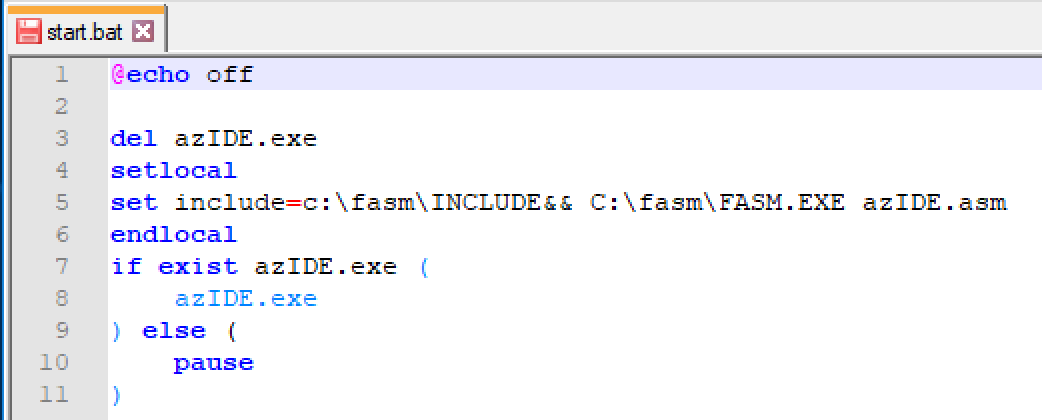


Рисунок 3.4 – Пакетный файл для автоматической сборки

**3.2 Разработка используемых структур данных**

Выбор структуры данных определяется конкретной задачей. В разрабатываемой программе существует две основные задачи, напрямую задествующие структуры: создание окна приложения и открытие файла с кодом.

Для начала рассмотрим открытие файлов в WinAPI. Сделать это можно при помощи функции *GetOpenFileName*, которая в качестве одного из параметров принимает указатель на структуру *OPENFILENAME*, которая содержит информацию необходимую для инициализации диалогового окна открытия файла. Для того, чтобы в ассемблерном коде была возможность использовать данную функцию, необходимо описать такую структуру в сегменте данных.



Рисунок 3.5 – Структура OPENFILENAME

Для создания окон графического интерфейса так же необходима структура, которая описывает характеристики конкретного окна. Но WinAPI предоставляет и другой вариант: можно использовать так называемые «шаблоны ресурсы», которые фактически являются теми же структурами, но способны существовать как отдельный файл и имеют более сложную внутреннюю структуру, способную содержать даже картинки и меню. Так как описание каждого элемента на экране в языке ассемблера вещь достаточно громоздкая, можно воспользоваться функцией отображения диалогового окна, которое описано в сегменте ресурсов.

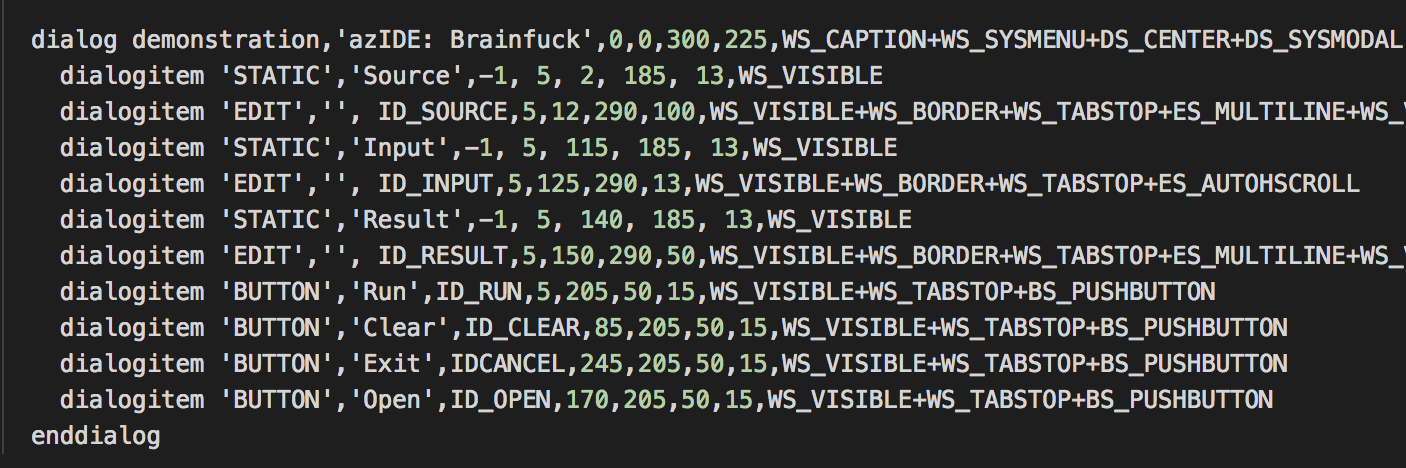


Рисунок 3.6 – Окно приложения, описанное в секции ресурсов

1. **ТЕСТИРОВАНИЕ**

Результаты тестирования ПС представлены в таблице 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| Тестируемая функция | Ход выполнения | Ожидаемый результат | Полученный результат |
| Сборка проекта из исходного кода | Запуск скрипта start.bat | Запуск приложения | Тестирование пройдено |
| Запуск простого приложения на Brainfuck | Запуск примера, идущего по умолчанию (вывод “Hello, World”) | Вывод Hello, World | Тестирование пройдено |
| Запуск приложения, использующего пользовательский ввод | Ввести код программы  Нажать на Run | Вывод результата исполнения | Тестирование пройдено |
| Переполнение ячейки | Запуск программы с переполнением ячейки | Переполнение корректно обрабатывается | Тестирование пройдено |
| Проверка балансировки скобок | Запуск программы с несбалансированным количеством скобок | Вывод ошибки | Тестирование пройдено |
| Выход за предел массива ячеек | Запуск программы, которая пытается перейти в 30 001 ячейку | Вывод ошибки | Тестирование пройдено |
| Очистка интерфейса | Нажатие кнопки Clear | Пустые поля с исходным кодом и вводом | Тестирование пройдено |
| Открытия файла с расширением .b/.bf | Нажать на кнопку Open Выбрать файл с расширением .b/.bf | В программе отображается содержимое файла | Тестирование пройдено |
| Выход из приложения | Нажатие на кнопку Exit | Приложение закрылось | Тестирование пройдено |

1. **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Для того чтобы начать работу с приложение необходимо запустить azIDE.exe или запустить скрипт build.bat, который соберёт новое приложение из исходных кодов. Первый вариант тривиален, тогда как для второго нужно отредактировать в скрипте путь к компилятору ассемблера FASM. После запуска программы отображается основной интерфейс (рисунок 5.1)

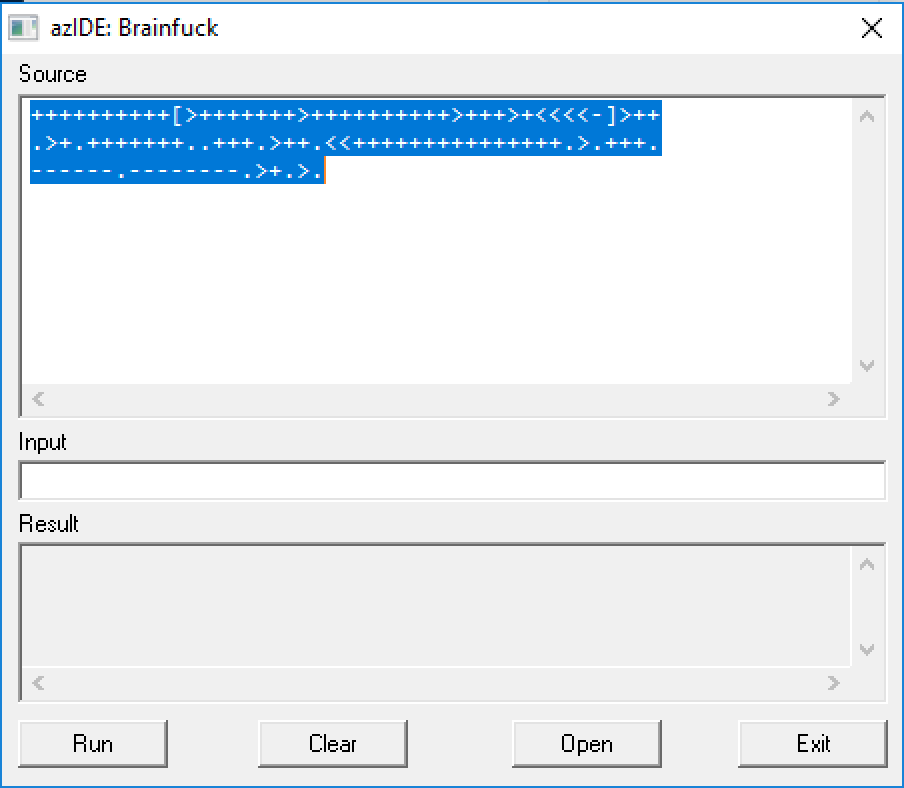


Рисунок 6.1 – Интерфейс программы

Если пользователь хочет убедится в работоспособности программы, то может сразу же нажать на кнопку “Run” и запустить предзагруженный пример программы на языке Brainfuck.

Всего пользовательский интерфейс состоит из трёх текстовых полей и четырёх кнопок. Верхнее текстовое поле предназначено для ввода исходного кода программы на языке Brainfuck. В среднем поле можно указать ввод для программы, а в нижнем после исполнения программы появится результат выполнения. Это поле не редактируемо пользователем.

Кнопка “Run” запускает процесс интерпретации программы, кнопка Clear очищает два верхних текстовых поля, кнопка Open открывает стандартный диалог операционной системы с предложением выбрать файл с исходным кодом программы на языке Brainfuck. Следует отметить, что так как большинство таких программ распространяется в файлах с расширением «.b» и «.bf», пользователь не сможет выбрать файл с другим расширением. После выбора файла, его содержимое отобразится в верхнем текстовом поле, а пользователь сможет запустить его на выполнение кнопкой “Run”.

**ЗАКЛЮЧЕНИЕ**

В результате работы над курсовым проектом была разработана полноценная среда разработки на языке Brainfuck со своим встроенным интерпретатором, а следовательно, возможностью запускать свои приложения. Данная среда разработки в полной мере соответствует всем требованиям, сформулированным в условии курсового проекта.

В ходе разработки были изучены возможности языка программирования ассемблер, а конкретно диалекта FASM, получены знания о WinAPI и навыки работы в отладчике IDA, опыт написания скриптов автоматизации и многое другое.

Работа была разделена на этапы, такие как анализ прототипов, литературных источников и постановка требований к проектируемому программному средству, разработка алгоритма и его схемы, конструирование программного средства, отладка и тестирование. После последовательного выполнения вышеперечисленных этапов разработки было получено исправно работающее программное средство.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Библиотека MSDN [Электронный ресурс] Режим доступа: msdn.microsoft.com/en-us/

[2] Flat assembler 1.72 Programmer's Manual – Tomasz Grysztar.

[3] Современные операционные системы – Э. Таненбаум

[4] IDA Decompiler Manual [Электронный ресурс] Режим доступа: [www.hex-rays.com/products/decompiler/manual/index.shtml](http://www.hex-rays.com/products/decompiler/manual/index.shtml)

**ПРИЛОЖЕНИЕ А**

**Исходный код программы**

format PE GUI 4.0

entry start

include 'win32a.inc'

ID\_SOURCE = 101

ID\_INPUT = 102

ID\_RESULT = 103

ID\_RUN = 104

ID\_CLEAR = 105

ID\_OPEN = 106

; MARK: - Data section

section '.data' data readable writeable

szError1 db 'Syntax error at: %u',0

szError2 db 'Runtime error at: %u',0

szError db 'Error.', 0

textbox\_font db 'Consolas', 0

file\_filters db 'Brainfuck source (\*.b)', 0, '\*.b;\*.bf', 0, 0

szDef db '++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++',13,10

db '.>+.+++++++..+++.>++.<<+++++++++++++++.>.+++.',13,10

db '------.--------.>+.>.',0

szSource rb 5000h

szInput rb 500h

szOutput rb 500h

filename rb MAX\_PATH

ofn:

.lStructSize dd 88

.hwndOwner dd NULL

.hInstance dd NULL

.lpstrFilter dd file\_filters

.lpstrCustomFilter dd NULL

.nMaxCustFilter dd 0

.nFilterIndex dd 1

.lpstrFile dd filename

.nMaxFile dd MAX\_PATH

.lpstrFileTitle dd NULL

.nMaxFileTitle dd 0

.lpstrInitialDir dd NULL

.lpstrTitle dd NULL

.Flags dd OFN\_HIDEREADONLY

.nFileOffset dw 0

.nFileExtension dw 0

.lpstrDefExt dd NULL

.lCustData dd NULL

.lpfnHook dd NULL

.lpTemplateName dd NULL

.lpReserved dd NULL

.dwReserved dd 0

.FlagsEx dd 0

file\_data\_ptr dd ?

heap\_handle dd ?

file\_handle dd ?

file\_size dd ?

num\_bytes\_read dd ?

hwnd\_textbox dd ?

hwnd\_main dd ?

; MARK: - Code section

section '.code' code readable executable

start:

push 0

call [GetModuleHandle]

push 0

push DialogProc

push NULL

push 42

push eax

call [DialogBoxParam]

push 0

call [ExitProcess]

proc DialogProc hwnddlg,msg,wparam,lparam

push ebx esi edi

cmp [msg],WM\_INITDIALOG

je .wminitdialog

cmp [msg],WM\_COMMAND

je .wmcommand

cmp [msg],WM\_CLOSE

je .wmclose

xor eax,eax

jmp .finish

.wminitdialog:

push szDef

push ID\_SOURCE

push [hwnddlg]

call [SetDlgItemText]

push textbox\_font

push DEFAULT\_PITCH

push PROOF\_QUALITY

push CLIP\_DEFAULT\_PRECIS

push OUT\_DEFAULT\_PRECIS

push ANSI\_CHARSET

xor ecx, ecx

mov cl, 7

@@:

push 0

loop @r

push 14

call [CreateFont]

push TRUE

push eax

push WM\_SETFONT

push ID\_SOURCE

push [hwnddlg]

call [SendDlgItemMessage]

jmp .processed

.wmcommand:

cmp [wparam],IDCANCEL

je .wmclose

cmp [wparam],ID\_RUN

je .wmrun

cmp [wparam],ID\_CLEAR

je .wmclear

cmp [wparam], ID\_OPEN

je .wmopen

jmp .processed

.wmopen:

mov eax, [hwnddlg]

mov [ofn.hwndOwner], eax

push ofn

call [GetOpenFileName]

test eax, eax

jz .processed

push NULL

push FILE\_ATTRIBUTE\_NORMAL

push OPEN\_EXISTING

push NULL

push FILE\_SHARE\_READ

push GENERIC\_READ

push filename

call [CreateFile]

cmp eax, INVALID\_HANDLE\_VALUE

jne .open\_ok

call error\_msgbox

jmp .processed

.open\_ok:

mov [file\_handle], eax

push NULL

push eax

call [GetFileSize]

inc eax

mov [file\_size], eax

call [GetProcessHeap]

mov [heap\_handle], eax

push [file\_size]

push HEAP\_ZERO\_MEMORY

push eax

call [HeapAlloc]

mov [file\_data\_ptr], eax

push NULL

push num\_bytes\_read

push [file\_size]

push [file\_data\_ptr]

push [file\_handle]

call [ReadFile]

push [file\_data\_ptr]

push ID\_SOURCE

push [hwnddlg]

call [SetDlgItemText]

push [file\_handle]

call [CloseHandle]

push [file\_data\_ptr]

push 0

push [heap\_handle]

call [HeapFree]

jmp .processed

.wmclear:

push NULL

push ID\_SOURCE

push [hwnddlg]

call [SetDlgItemText]

push NULL

push ID\_INPUT

push [hwnddlg]

call [SetDlgItemText]

push NULL

push ID\_SOURCE

push [hwnddlg]

call [SetDlgItemText]

jmp .processed

.wmrun:

push 5000h

push szSource

push ID\_SOURCE

push [hwnddlg]

call [GetDlgItemText]

push 5000h

push szInput

push ID\_INPUT

push [hwnddlg]

call [GetDlgItemText]

push 500h

push szOutput

call [RtlZeroMemory]

stdcall Brainfuck,szSource,szInput,szOutput

cmp eax,-1

jne @f

push edx ; Syntax error

push szError2

push szOutput

call [wsprintf]

jmp .set\_result

@@:

cmp eax,-2

jne .set\_result

push edx ; Runtime error

push szError1

push szOutput

call [wsprintf]

.set\_result:

add esp,12

push szOutput

push ID\_RESULT

push [hwnddlg]

call [SetDlgItemText]

jmp .processed

.wmclose:

push 0

push [hwnddlg]

call [EndDialog]

.processed:

mov eax,1

.finish:

pop edi esi ebx

ret

endp

error\_msgbox:

push MB\_OK

push szError

push szError

push HWND\_DESKTOP

call [MessageBox]

ret

;------------------------------------------------------------------

; Output:

; EAX = 0 - success

; EAX = -1 - syntax error \ EDX has

; EAX = -2 - runtime error / the error position

;------------------------------------------------------------------

proc Brainfuck lpSource:DWORD, lpInput:DWORD, lpOutput:DWORD

BRAINFUCK\_MEM\_LENGTH=30000

locals

hHeap dd ?

mem dd ?

endl

call [GetProcessHeap]

mov [hHeap],eax

push BRAINFUCK\_MEM\_LENGTH

push HEAP\_ZERO\_MEMORY

push eax

call [HeapAlloc]

mov [mem],eax

mov esi,[lpSource]

cld

xor ecx,ecx

.scan\_loop:

lodsb

or al,al

jz .scan\_loop\_done

cmp al,'['

jne @f

inc ecx

jmp .scan\_loop

@@:

cmp al,']'

jne .scan\_loop

or ecx,ecx

je .loc\_syntax\_error

dec ecx

jmp .scan\_loop

.scan\_loop\_done:

or ecx,ecx

jnz .loc\_syntax\_error

; Execute the code

mov edi,[lpOutput]

mov esi,[lpInput]

mov edx,[lpSource]

xor ebx,ebx

.loc\_loop:

mov al,[edx]

or al,al

jz .loc\_success

cmp al,'>'

jne .not\_increase

inc ebx

cmp ebx,BRAINFUCK\_MEM\_LENGTH

je .loc\_runtime\_error

inc edx

jmp .loc\_loop

.not\_increase:

cmp al,'<'

jne .not\_lower

cmp ebx,0

je .loc\_runtime\_error

dec ebx

inc edx

jmp .loc\_loop

.not\_lower:

cmp al,'+'

jne .not\_bigger

mov eax,[mem]

inc byte [eax+ebx]

inc edx

jmp .loc\_loop

.not\_bigger:

cmp al,'-'

jne .not\_decrease

mov eax,[mem]

dec byte [eax+ebx]

inc edx

jmp .loc\_loop

.not\_decrease:

cmp al,'.'

jne .not\_output

mov eax,[mem]

mov al,[eax+ebx]

stosb

inc edx

jmp .loc\_loop

.not\_output:

cmp al,','

jne .not\_input

mov al,byte [esi]

or al,al

jz @f

inc esi

@@:

mov ecx,[mem]

mov [ecx+ebx],al

inc edx

jmp .loc\_loop

.not\_input:

cmp al,'['

jne .not\_cycle\_start

xor ecx,ecx

inc ecx

mov eax,[mem]

cmp byte [eax+ebx],0

je .smth

inc edx

jmp .loc\_loop

.smth:

inc edx

mov al,[edx]

cmp al,'['

jne @f

inc ecx

@@:

cmp al,']'

jne .smth

dec ecx

or ecx,ecx

jnz .smth

inc edx

jmp .loc\_loop

.not\_cycle\_start:

cmp al,']'

jne .not\_cycle\_end

xor ecx,ecx

inc ecx

mov eax,[mem]

cmp byte [eax+ebx],0

jne .not\_cycle\_start\_1

inc edx

jmp .loc\_loop

.not\_cycle\_start\_1:

dec edx

mov al,[edx]

cmp al,']'

jne @f

inc ecx

@@:

cmp al,'['

jne .not\_cycle\_start\_1

dec ecx

or ecx,ecx

jnz .not\_cycle\_start\_1

jmp .loc\_loop

.not\_cycle\_end:

inc edx ; skip instruction

jmp .loc\_loop

.loc\_syntax\_error:

sub esi,[lpSource]

mov edx,esi

mov eax,-2

jmp .loc\_ret

.loc\_runtime\_error:

sub edx,[lpSource]

inc edx

mov eax,-1

jmp .loc\_ret

.loc\_success:

xor eax,eax

.loc\_ret:

push eax edx

push [mem]

push NULL

push [hHeap]

call [HeapFree]

pop edx eax

ret

endp

; MARK: - Import section

section '.idata' import data readable

library kernel32,'kernel32.dll',\

user32,'user32.dll',\

shell32,'shell32.dll',\

Gdi32, 'Gdi32.dll',\

Comdlg32 , 'Comdlg32.dll'

import Gdi32, \

CreateFont, 'CreateFontA'

import Comdlg32, \

GetOpenFileName, 'GetOpenFileNameA', \

GetSaveFileName, 'GetSaveFileNameA'

include 'api\kernel32.inc'

include 'api\user32.inc'

include 'api\shell32.inc'

; MARK: - Resourse section

section '.rsrc' resource data readable

directory RT\_DIALOG,dialogs

resource dialogs,42,LANG\_ENGLISH+SUBLANG\_DEFAULT,demonstration

dialog demonstration,'azIDE: Brainfuck',0,0,300,225,WS\_CAPTION+WS\_SYSMENU+DS\_CENTER+DS\_SYSMODAL

dialogitem 'STATIC','Source',-1, 5, 2, 185, 13,WS\_VISIBLE

dialogitem 'EDIT','', ID\_SOURCE,5,12,290,100,WS\_VISIBLE+WS\_BORDER+WS\_TABSTOP+ES\_MULTILINE+WS\_VSCROLL+WS\_HSCROLL+ES\_WANTRETURN

dialogitem 'STATIC','Input',-1, 5, 115, 185, 13,WS\_VISIBLE

dialogitem 'EDIT','', ID\_INPUT,5,125,290,13,WS\_VISIBLE+WS\_BORDER+WS\_TABSTOP+ES\_AUTOHSCROLL

dialogitem 'STATIC','Result',-1, 5, 140, 185, 13,WS\_VISIBLE

dialogitem 'EDIT','', ID\_RESULT,5,150,290,50,WS\_VISIBLE+WS\_BORDER+WS\_TABSTOP+ES\_MULTILINE+WS\_VSCROLL+WS\_HSCROLL+ES\_READONLY+ES\_WANTRETURN

dialogitem 'BUTTON','Run',ID\_RUN,5,205,50,15,WS\_VISIBLE+WS\_TABSTOP+BS\_PUSHBUTTON

dialogitem 'BUTTON','Clear',ID\_CLEAR,85,205,50,15,WS\_VISIBLE+WS\_TABSTOP+BS\_PUSHBUTTON

dialogitem 'BUTTON','Exit',IDCANCEL,245,205,50,15,WS\_VISIBLE+WS\_TABSTOP+BS\_PUSHBUTTON

dialogitem 'BUTTON','Open',ID\_OPEN,170,205,50,15,WS\_VISIBLE+WS\_TABSTOP+BS\_PUSHBUTTON

enddialog

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР КП 1–40 01 01 001 ПЗ | | | | Пояснительная записка | | | | 39 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 651001  001 СП | | | | Среда разработки на языке Brainfuck. Схема программы | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | **ГУИР КП 1-40 01 01 001 Д1** | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | Среда разработки на языке Brainfuck  Ведомость курсового  проекта |  | | | | Лист | Листов |
| Разраб. | | Азаров А.А. |  | 28.11.18 | Т |  | |  | 39 | 39 |
| Пров. | | Базылев Е.Н. |  |  | Кафедра ПОИТ  гр. 651001 | | | | | |
|  | |  |  |  |