**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Образовательная программа «Программная инженерия»

|  |  |  |  |
| --- | --- | --- | --- |
| СОГЛАСОВАНО  Научный руководитель, профессор департамента «Программной инженерии», доктор технических наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.И. Легалов  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | |  | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия»  профессор департамента программной инженерии, канд. техн. наук  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н. А. Павлочев  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **КРОСС-КОМПИЛЯТОРС ЯЗЫКА ПРОГРАММИРОВАНИЯ SMALL C**  **В АССЕМБЛЕР ПРОЦЕССОРА RISC-V, ДЛЯ ЭМУЛЯТОРА RARS**  **Программа и методика испытаний**  **ЛИСТ УТВЕРЖДЕНИЯ**  **RU.17701729.04.01-01 51 01-1-ЛУ** | | | | | |
|  | | Исполнитель  студент группы БПИ229  \_\_\_\_\_\_\_\_\_ / Р. С. Громов/  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | | | |
|  | | | |

**Москва 2024**

УТВЕРЖДЕН

RU.17701729.10.03-01 51 01-1-ЛУ

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | ***Подп. и дата*** |  | | ***Инв. № дубл.*** |  | | ***Взам. инв. №*** |  | | ***Подп. и дата*** |  | | ***Инв. № подл*** |  | | **КРОСС-КОМПИЛЯТОРС ЯЗЫКА ПРОГРАММИРОВАНИЯ SMALL C**  **В АССЕМБЛЕР ПРОЦЕССОРА RISC-V, ДЛЯ ЭМУЛЯТОРА RARS**  **Программа и методика испытаний**  **RU.17701729.04.01-01 51 01-1**  **Листов 17** | |
|  |  |
|  |
|  | |

**Москва 2024**

**СОДЕРЖАНИЕ**

[**1.** **ОБЪЕКТ ИСПЫТАНИЙ** 3](#_Toc162888679)

[1.1. Наименование программы 3](#_Toc162888680)

[1.2. Краткая характеристика области применения 3](#_Toc162888681)

[**2.** **ЦЕЛЬ ИСПЫТАНИЙ** 4](#_Toc162888682)

[**3.** **ТРЕБОВАНИЯ К ПРОГРАММЕ** 5](#_Toc162888683)

[3.1 Требования к функциональным характеристикам 5](#_Toc162888684)

[3.1.1 Требования к составу выполняемых функций 5](#_Toc162888685)

[3.1.2 Требование к интерфейсу программы: 6](#_Toc162888686)

[3.2. Требования к надежности 6](#_Toc162888687)

[3.3. Требования к производительности 6](#_Toc162888688)

[**4.** **ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ** 7](#_Toc162888689)

[**5.** **СРЕДСТВА И ПОРЯДОК ИСПЫТАНИЙ** 8](#_Toc162888690)

[5.2. Программные средства, используемые во время испытаний 8](#_Toc162888691)

[5.3. Порядок проведения испытаний 8](#_Toc162888692)

[**6.** **МЕТОДЫ ИСПЫТАНИЙ** 9](#_Toc162888693)

[6.1. Испытания выполнения требований к программной документации 9](#_Toc162888694)

[6.2. Испытания выполнения требований к функциональным характеристикам 9](#_Toc162888695)

[6.2.1. Установка 9](#_Toc162888696)

[6.2.2. Проверка лексического анализатора 9](#_Toc162888697)

[6.2.3. Проверка синтаксического анализатора 10](#_Toc162888698)

[6.3. Испытания надежности программы 12](#_Toc162888699)

[**7.** **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 14](#_Toc162888700)

[**ПРИЛОЖЕНИЕ** 15](#_Toc162888701)

[**ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ** 17](#_Toc162888702)

1. **ОБЪЕКТ ИСПЫТАНИЙ**

## **Наименование программы**

Наименование программы: «Кросс-компилятор с языка программирования Small C в ассемблер процессора RISC-V для эмулятора RARS».

Краткое наименование: «RISC-V Cross-compiler».

## **Краткая характеристика области применения**

Это приложение предназначено для студентов программной инженерии, которые хотят улучшить свои знания в области архитектуры вычислительных систем и усовершенствовать свои навыки работы с языком ассемблера процессора RISC-V для успешного выполнения семинарских и практических заданий.

1. **ЦЕЛЬ ИСПЫТАНИЙ**

Целью испытаний является проверка разработанной программы на соответствие требованиям к функциональным характеристикам и надежности, изложенным в техническом задании.

# **ТРЕБОВАНИЯ К ПРОГРАММЕ**

* 1. Требования к функциональным характеристикам
     1. Требования к составу выполняемых функций

Компилятор должен поддерживать следующие конструкции языка:

* Одиночные объявления переменной;
* Множественные объявления переменных;
* Одиночное определение переменной;
* Множественные определения переменных;
* Арифметическое выражение;
* Логическое выражение;
* Сравнительное выражение;
* Поразрядное выражение;
* Префиксное выражение;
* Постфиксное выражение;
* Условный оператор if-else;
* Цикл while;
* Цикл do-while;
* Цикл for;
* Конструкция switch-case;
* Объявление функции.
  + 1. Требование к интерфейсу программы:

Пользователь взаимодействует с программной через консольное окно. Должны быть доступны следующие команды:

* Ввод пути (абсолютного и относительного) до исходного файла, который написан на языке Small-C;
* Выбор форматирования файла препроцессором;
* Генерация кода.

## **3.2. Требования к надежности**

Компилятор не должен завершаться аварийно при любом пользовательском вводе наборе входных данных. В случае, если исходного файла не существует, или он занят другим процессом, отсутствует утилита CMake или компилятор GCC, пользователь должен получить сообщение об ошибке в консоль.

Компилятор должен выдавать ошибку, если компилируемая программа некорректна с точки зрения синтаксиса или семантики языка Small-C.

Генерируемая программа на языке ассемблера может завершаться аварийно в эмуляторе RARS, если исходная программа написана с нарушением логики, но не должна этого делать в силу чисто технических причин, например, переполнение виртуального стека.

## **3.3. Требования к производительности**

Скорость работы компилятора должна быть такой, чтобы это не затрудняло работу программиста. Для этого время компиляции файла, объём которого превосходит стандартную библиотеку языка Small-C, не должно превышать одной минут. Если программа имеет размер, не превышающий 4 Кб, то время перекомпиляции не должно превышать 10 секунд.

1. **ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ**

На испытание должна быть представлена документация к программе в следующем составе:

Программная документация включает в себя следующие документы:

1. Техническое задание (ГОСТ 19.201-78);
2. Пояснительная записка (ГОСТ 19.404-79)
3. Программа и методика испытаний (ГОСТ 19.301-79);
4. Руководство оператора (ГОСТ 19.505-79);
5. Текст программы (ГОСТ 19.401-78).
6. **СРЕДСТВА И ПОРЯДОК ИСПЫТАНИЙ**
   1. **Технические средства, используемые во время испытаний**

Испытания проводились на персональном компьютере с установленной виртуальной машиной

Характеристики персонального компьютера:

1. 8 ядерный процессор AMD Ryzen 7 5800U 1.90 GHz;
2. 16 Гб оперативной памяти (доступно 13.9 Гб);
3. SSD диск размером 952 Гб (свободно 671 Гб);
4. Клавиатура;
5. Монитор.

## **Программные средства, используемые во время испытаний**

Во время испытаний использовались следующие версии программных средств:

1. Операционная система Windows 11;
2. Компилятор GCC 6.3.0
3. CMake 3.27.6
4. Эмулятор RARS
5. Виртуальная машина VirtualBox 7.0.8
6. Операционная система Ubuntu 21.04 на виртуальной машине

## **Порядок проведения испытаний**

Испытания должны проводиться в следующем порядке:

1. Проверка требований к программной документации;
2. Проверка требований к функциональным характеристикам;
3. Проверка требований к надежности;
4. Проверка требований к производительности.
5. **МЕТОДЫ ИСПЫТАНИЙ**

## **6.1. Испытания выполнения требований к программной документации**

Состава программной документации проверяется вручную. Сначала необходимо проверить наличие каждого требуемого документа. Затем удостовериться, что содержание каждого документа соответствует ГОСТ.

**6.2. Испытания выполнения требований к функциональным характеристикам**

### **6.2.1. Установка**

Программа хранится в публичном репозитории, доступ к которому можно получить по следующей ссылке: *https://github.com/mrForza/RISC-V-Cross-compiler.*

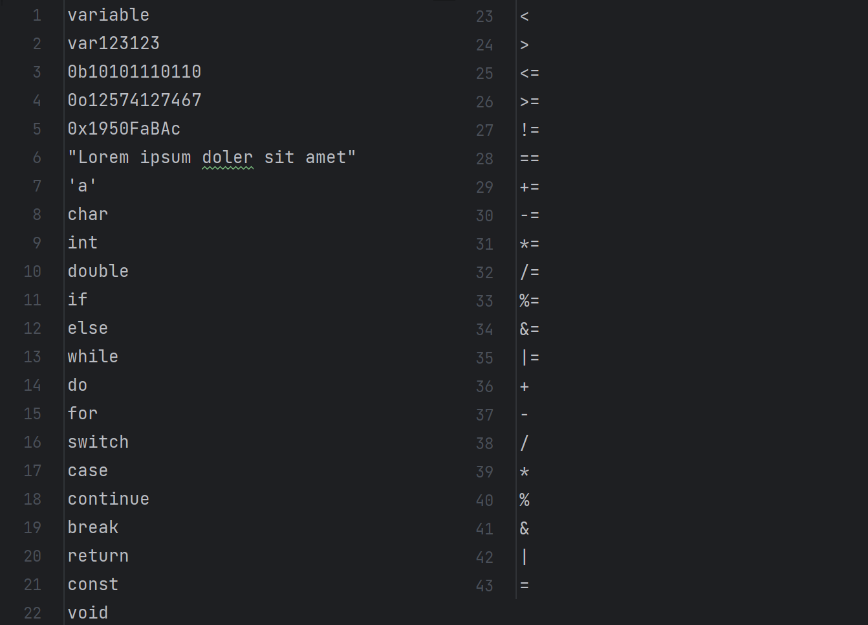
Чтобы установить программу себе на локальное устройство, нужно:

1. Склонировать репозиторий с помощью утилиты *gitbash* с помощью команды *git clone https://github.com/mrForza/RISC-V-Cross-compiler.git;*
2. Перейти в директорию, в которую был склонирован репозиторий;
3. С помощью компилятора gcc скомпилировать файл main.c в исполняемый файл;
4. Запустить исполняемый файл.

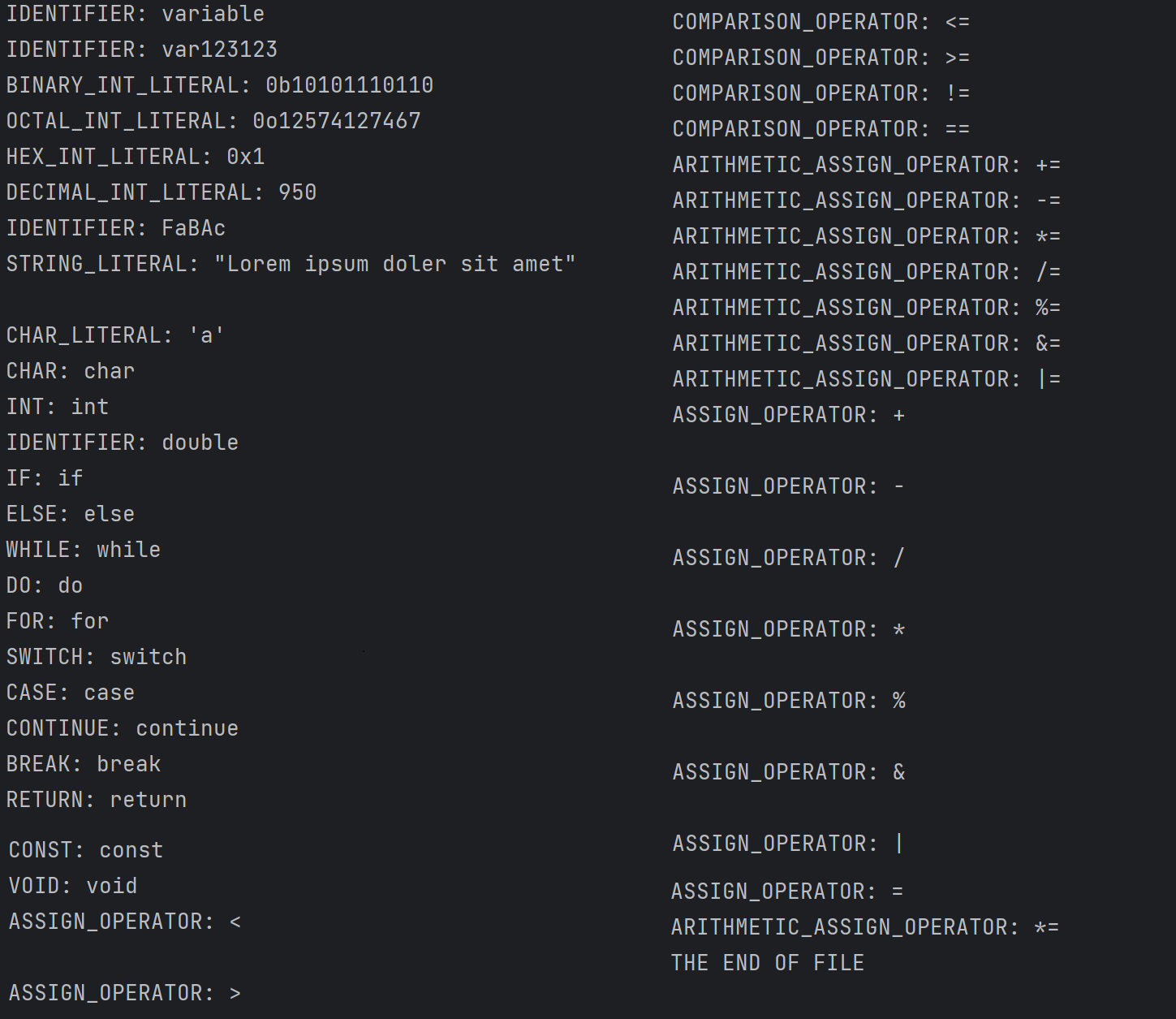
### **6.2.2. Проверка лексического анализатора**

Тесты для синтаксического анализатор представлены в виде текстового файла, в котором хранятся всевозможные лексемы языка программирования Small-C. После отработки данного файла программа к каждой лексеме должна подписать тип токена.

*Рисунок 1. Тестовые данные для лексического анализатора*



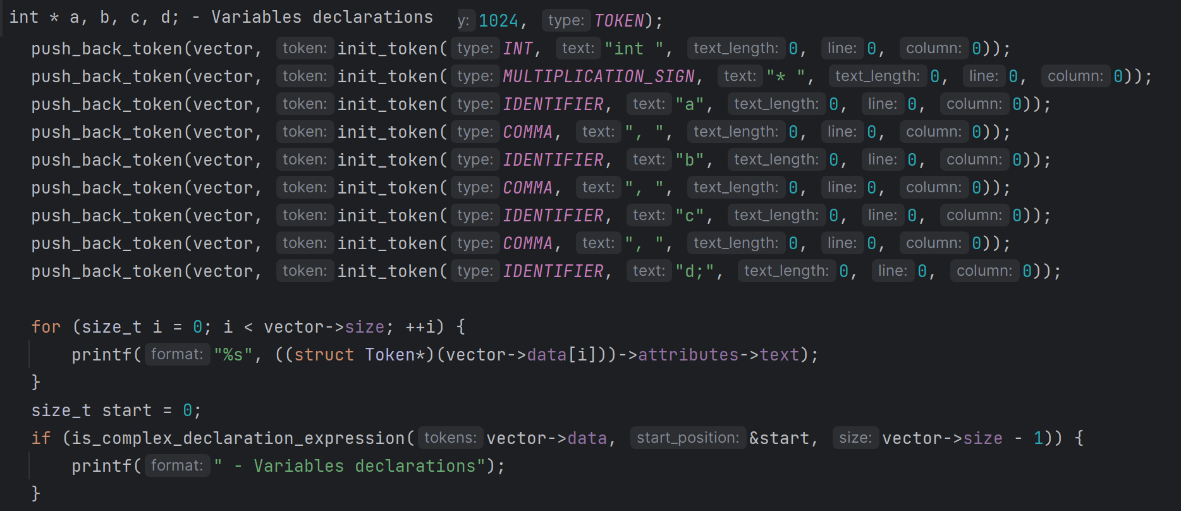
*Рисунок 2. Результат работы лексического анализатора*



### **6.2.3. Проверка синтаксического анализатора**

Тесты для синтаксического анализатора представлены в виде вектора токенов, написанных на языке C. В качестве проверки используются функция-распознаватель. Если она вернула true, значит данная последовательность токенов была распознана верно. И на экран консоли выводится информация о типе данной синтаксической структуры.

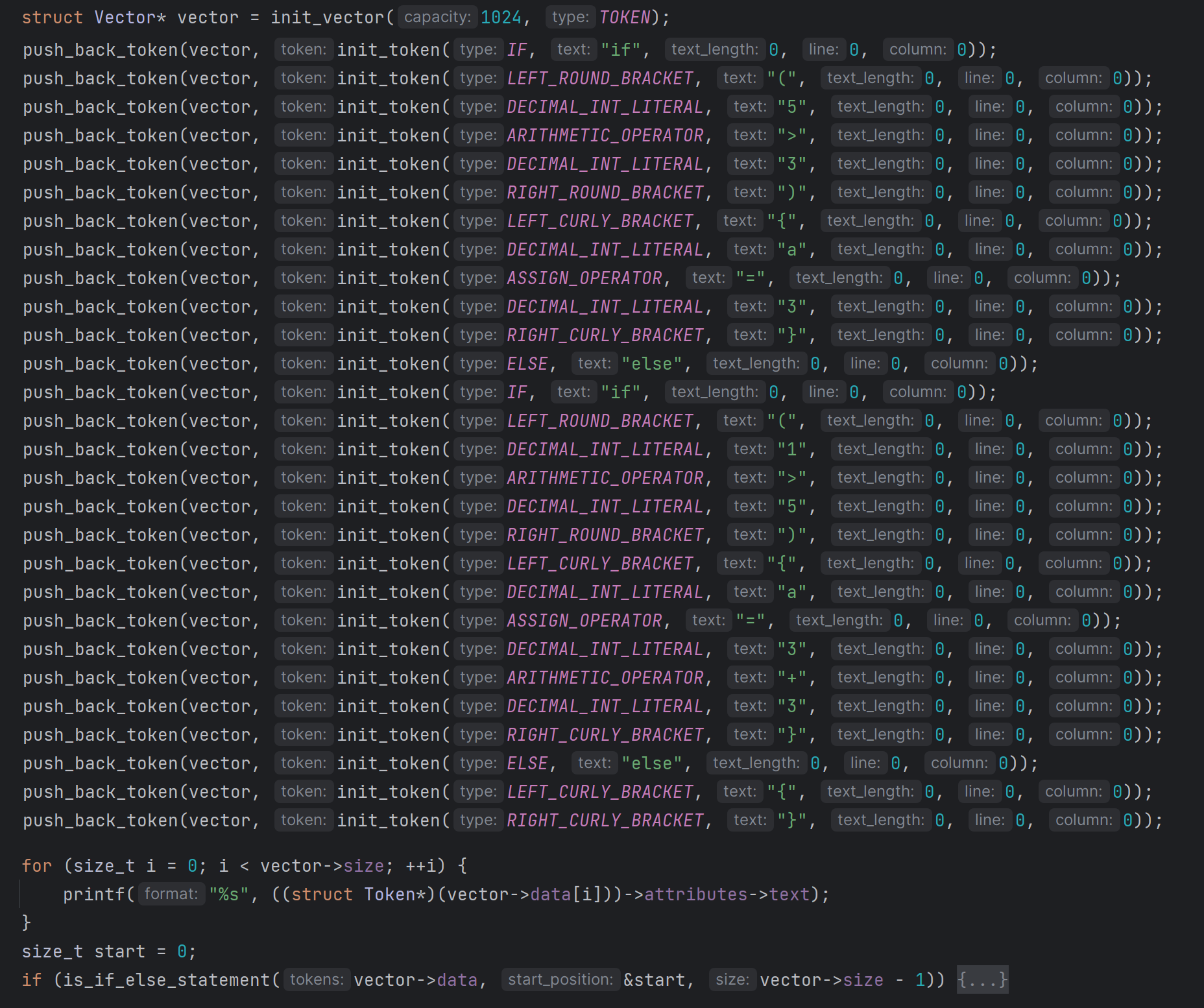
*Рисунок 3. Тестовые данные для синтаксического анализатора. Объявление переменных*



*Рисунок 4. Результат работы синтаксического анализатора. Объявление переменных*



*Рисунок 5. Тестовые данные для синтаксического анализатора. Условный оператор*



*Рисунок 6. Результат работы синтаксического анализатора. Условный оператор*



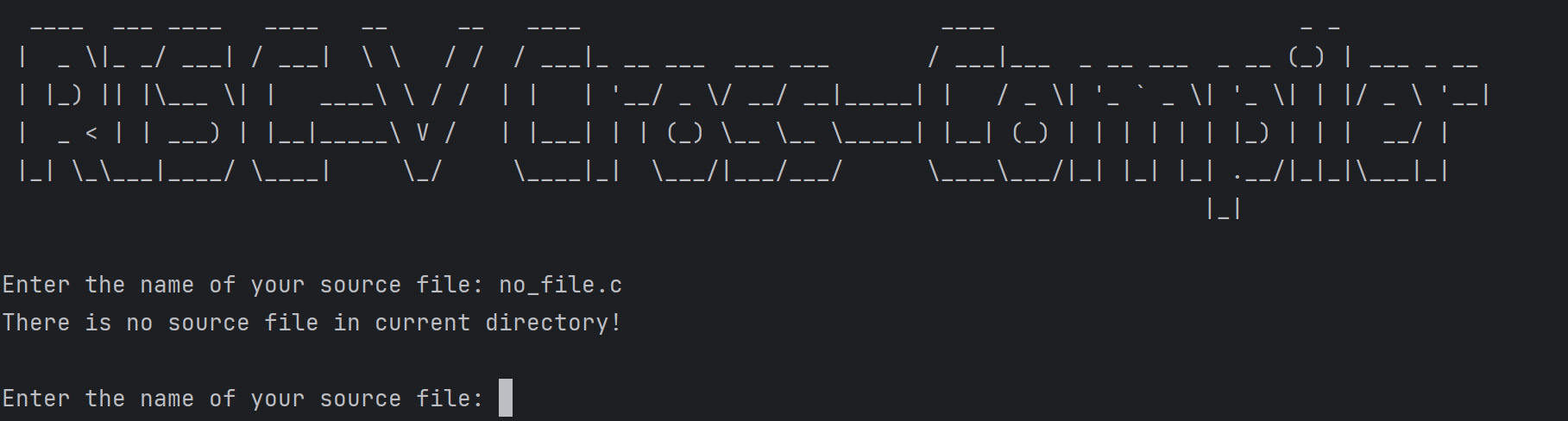
## **6.3. Испытания надежности программы**

В этом пункте будет оценено то, как компилятор реагирует на ошибки. Для начало будут рассмотрены ошибки препроцессора.

*Рисунок 7. Ввод некорректного имени файла.*

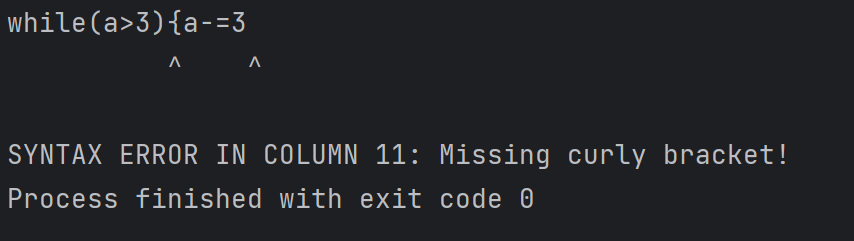


*Рисунок 8.Отсутствие файла в директории с исполняемым файлом.*

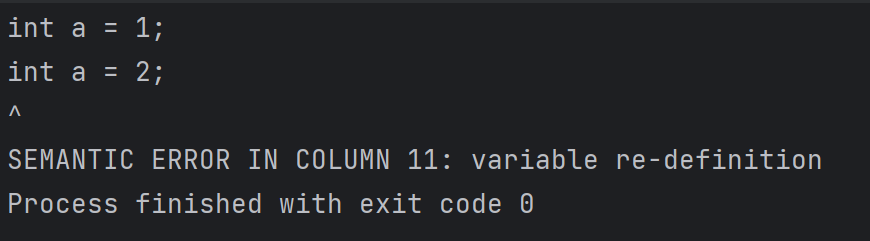


Далее рассмотрим ошибки в коде исходной программы. Они разделяются на синтаксические и семантические. Пример сообщения о синтаксической ошибке представлен на *рисунке 8*. В данном случае была пропущена фигурная скобка в теле цикла while. Сообщение содержит тип ошибки, её описание, а также номер колонки в строке, в которой была допущена ошибка. Пример сообщения о семантической ошибке представлен на *рисунке 9*.

*Рисунок 8. Синтаксическая ошибка, связанная с пропущенной фигурной скобкой.*

**

*Рисунок 9. Семантическая ошибка, связанная с повторным определением переменной*

**

1. **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**
2. Small-C Wikipedia.
3. J.G. Harstone. // C compilers for the BBC Computer.
4. Д. Хендрикс. // Компилятор языка Си для микроЭВМ.
5. Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. // Компиляторы: принципы, технологии и инструментарий.
6. Свердлов С. // Конструирование компиляторов, учебное пособие.

# **ПРИЛОЖЕНИЕ**

**ТЕРМИНОЛОГИЯ**

**Компилятор** — программа, переводящая написанный на языке программирования текст в набор машинных кодов.

**Лексический анализ** (токенизация) – это процесс аналитического разбора входной последовательности символов на распознанные группы — лексемы — с целью получения на выходе идентифицированных последовательностей, называемых «токенами».

**Синтаксический анализ** (парсинг) в лингвистике и информатике - это процесс сопоставления линейной последовательности лексем (слов, токенов) естественного или формального языка с его формальной грамматикой. Результатом обычно является дерево разбора (синтаксическое дерево). Обычно применяется совместно с лексическим анализом.

**Синтаксис** языка программирования — набор правил, описывающий комбинации символов алфавита, считающиеся правильно структурированной программой (документом) или её фрагментом.

**Препроцессор** — компьютерная программа, принимающая данные на входе и выдающая данные, предназначенные для входа другой программы.

**Метаязык** — язык, предназначенный для описания другого языка, называемого объектным языком.

**Терминальные** **символы** – это символы, которые могут появляться в выходных данных производственных правил формальной грамматики и которые не могут быть изменены с помощью правил грамматики.

**Нетерминал** (нетерминальный символ) — объект, обозначающий какую-либо сущность языка (например: формула, арифметическое выражение, команда) и не имеющий конкретного символьного значения.

**Формальная грамматика** или просто грамматика в теории формальных языков — способ описания формального языка, то есть выделения некоторого подмножества из множества всех слов некоторого конечного алфавита.

**Регулярные выражения** – это формальный язык, используемый в компьютерных программах, работающих с текстом, для поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов.

**Лексема** – это последовательность символов исходной программы, которая соответствует шаблону токена и идентифицируется лексическим анализатором как экземпляр токена.

**Шаблон (pattern)** — это описание вида, который может принимать лексема токена. В случае ключевого слова шаблон представляет собой просто последовательность символов, образующих это ключевое слово.

**Токен** – это пара, состоящая из имени токена и необязательного атрибута.

**ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Номера листов (страниц) | | | | | Всего листов (страниц в докум.) | № документа | Входящий № сопроводительного докум. и дата | Подп. | Дата |
| Изм. | Измененных | Замененных | Новых | Аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |