Documentation for Delivery 1

CSCI 465 – Fall 2015

Marshall Mattingly III

1 October 2015

Table of Contents

[1. Scanner 4](#_Toc431452024)

[1.1. Assumptions 4](#_Toc431452025)

[1.1.1. Debug information shall be outputted to stdout and stderr 4](#_Toc431452026)

[1.1.2. Initially, only a subset of Pascal shall be parsed 4](#_Toc431452027)

[1.1.3. Flex / Bison shall be used as counter-points to the hand-made scanner 4](#_Toc431452028)

[1.2. Current Status 4](#_Toc431452029)

[1.2.1. Feature-complete with the requirements 4](#_Toc431452030)

[1.2.2. Multi-line and inline-comments 4](#_Toc431452031)

[1.2.3. Minor warnings and error suggestions 4](#_Toc431452032)

[1.3. Future Work 4](#_Toc431452033)

[1.3.1. Completion of Flex implementation 4](#_Toc431452034)

[1.3.2. Add more Pascal functionality 4](#_Toc431452035)

[1.3.3. Enhanced error messages 5](#_Toc431452036)

[2. IO module 6](#_Toc431452037)

[2.1. Assumptions 6](#_Toc431452038)

[2.1.1. Standard C library FILE handling 6](#_Toc431452039)

[2.2. Current Status 6](#_Toc431452040)

[2.2.1. Read / Write / Unget 6](#_Toc431452041)

[2.3. Future Work 6](#_Toc431452042)

[2.3.1. Read line-by-line 6](#_Toc431452043)

[3. Symbol Table 7](#_Toc431452044)

[3.1. Assumptions 7](#_Toc431452045)

[3.1.1. Only initialized for Delivery 1 7](#_Toc431452046)

[3.2. Current Status 7](#_Toc431452047)

[3.2.1. Basic functionality 7](#_Toc431452048)

[3.3. Future Work 7](#_Toc431452049)

[3.3.1. Hashtable 7](#_Toc431452050)

[3.3.2. Scopes 7](#_Toc431452051)

# Scanner

The scanner is responsible for the lexical analysis in the program. It takes input from the IO module, character by character, and transforms the input into lexemes (or tokens) for use by the parser.

## Assumptions

This section details any assumptions made about the scanner.

### Debug information shall be outputted to stdout and stderr

Any debugging information will be outputted to stdout (for general information) and stderr (for warnings). If the user would like to store this output, it must be redirected into a file.

### Initially, only a subset of Pascal shall be parsed

Given constraints, only the subset of Pascal defined in our documentation will be provided. However, it is the intention of this author to create a fully-working Pascal compiler.

### Flex / Bison shall be used as counter-points to the hand-made scanner

Flex and Bison, being automated generators for lexical analysis and parsing, respectively, will be used to attempt to verify the hand-written scanner and parser. This shall work as a nice study into the limitations of Flex / Bison vs hand-written compilers.

## Current Status

This section details the current status of the scanner.

### Feature-complete with the requirements

The scanner can create lexemes for each feature in the requirements.

### Multi-line and inline-comments

Multi-line and inline comments are accepted with the following form: (\* … \*) and { … }. All comments are simply passed over by the scanner, as if they were whitespace.

### Minor warnings and error suggestions

The scanner attempts to be somewhat intelligent with its warnings and errors, provided a line number and suggested issue when encountering an error – i.e. if an identifier starts with a number, the scanner will respond with incorrect number or identifier (since it isn’t entirely sure which has occurred). Warnings are things that may be unintended behavior, such as providing an empty string, since the scanner cannot differentiate between a string and a character.

## Future Work

This section details any future work that needs to be completed on the scanner.

### Completion of Flex implementation

Currently, the Flex implementation is missing key parts, such as string literal recognition. It must be extended to serve as a true counter-point.

### Add more Pascal functionality

The scanner currently parses a subset of the Pascal language. Adding tokens and logic to the scanner is relatively trivial, and shall be added in the future. However, adding parsing logic to the parser afterward may be more troublesome.

### Enhanced error messages

While the current error messages provide some detail on suggested fixes, enhancing the scanner to provide even more robust error messages would be ideal.

## Design

This section is intended to have the FSM used to create the scanner. However, given time restraints it has been omitted.

# IO module

The IO module is responsible for reading files and writing output. This is called by the scanner to get the characters required for lexical analysis.

## Assumptions

This section details the assumptions for the IO module.

### Standard C library FILE handling

The IO module, to remain as cross-platform as possible, uses the basic C-library functionality for FILE handling.

## Current Status

This section details the current status of the IO module.

### Read / Write / Unget

Currently, the IO module can simply read a character from a FILE, write to the stderr FILE, and unget a character back into a FILE stream.

## Future Work

This section details the future work for the IO module.

### Read line-by-line

Currently, characters are read in individually. However, it would be a nice feature to have the IO module load in entire lines, storing them into a buffer, and providing the characters for the scanner from the buffer. This will allow lines to be printed before lexemes, rather than afterward.

# Symbol Table

The symbol table is responsible for keeping track of all lexemes and their values in the program, which varies by scope.

## Assumptions

This section details the assumptions for the symbol table.

### Only initialized for Delivery 1

The symbol table requires the parser for context; therefore, only the keywords are initialized into the symbol table for the first delivery.

## Current Status

The section details the current status of the symbol table

### Basic functionality

Currently, the symbol table has basic linked-list style functionality, with support for multiple scopes. However, this functionality will be altering greatly in the future.

## Future Work

This section details the future work for the symbol table.

### Hashtable

The symbol table uses link-listed to store values, which is inefficient compared to a hashtable. For the next deliverable, the symbol table will use a hashtable instead.

### Scopes

Scopes have given basic support in the current implementation; however, they will need to be fully supported for the next deliverable.