KAPLAN INTERNATIONAL PATHWAYS

Assessment Task Information

Assessment Title : Lexical Analysis

Module Name : Computing

Module - IY469

Tutor - Chatrapathi Akula

Assessment Weighting - 50%

Assessment Start - 6-Apr-2020 (09:00 AM) UK Time

Assessment Deadline - 27-Apr-2020 (09:00 AM) UK Time

Assessment Task Instructions:

Core Task:

You are required to write a **lexer** that extracts tokens from a snippet of code written in a programming language called **Mk**. The assessment is via submission of a report together with a link to your code in the appendix. In addition, there will be a one-to-one private presentation after submitting the report. The focus of the presentation aside from the code, is to demonstrate sound research skills and the ability to think through the problem step-by-step. Specifically, the rationale behind choosing a specific way of solving a problem when there are competing solutions.

The following is a representative example of code written in **Mk**. Your lexer will take a file containing code like this and extract tokens from it.

```
let alpha_1 = 54;
let vec = 0:2:6;
let m = [1,2,3; 4,5,6];
let s = m[:,2];
m[:,2] = m[:,1];
```

The lexer should be able to extract tokens from a file contaning arbitrary Mk code, as long as it contains valid tokens. Please see below for the list of valid tokens.

Structure

A report of 1500 (+10%) words covering the following areas:

- **Abstract** - a short summary of your work - introduction, methodology, results and conclusion (1-2 sentence for each section).
- **Introduction** - a short introduction to the task undertaken and typical use cases.
- **Aims** and **Objectives** - What are your trying to achieve (aim) and steps on how you are going to achieve this (objectives).
- **Methodology** - Talk about your approach to the implementing the program. Mention and describe alternative approaches to the task, if any. Provide rationale for why you chose your method over others. Make use of flow charts and pseudocode to illustrate various parts of your program.
- **Discussion** - Discuss your program - is it working as intended, what are its limitations, what can be improved etc.
- **Conclusion** - Conclude your work (summary of your work, but in a different way than abstract) - what was achieved.
- **References** - UWE Harvard format.
- **Appendix** - Link to your code.

Resources

- https://www.stackoverflow.com
- https://en.cppreference.com/w/c
- https://www.tutorialspoint.com/cprogramming/index.htm

- Textbook - KN King - C Programming : A Modern Approach (Second Edition)
- Lecture Notes

Assessment Referencing Style

Harvard Referencing. Please see UWE guide

http://www1.uwe.ac.uk/students/studysupport/studyskills/referencing/uweharvard/howtociteawork.aspx#quoting

Expected Word Count

1500 (+10%) words.

Learning Outcomes Assessed

- Learn to analyze text through Lexical Analysis
- Understand the design process of developing complex programs with several functions
- Learn to use variables, loops, structures, pointers, and other programming functions effectively
- Understand how to read from and write to files
- Improve knowledge of C programming
- Learn to efficiently present your findings in a limited number of words
- Study independently and become a more effective learner
- Learn to evaluate different methods and choose the most suitable one for the task

Submission Requirements

Submission is via Turnitin link on VLE. Reports must be submitted by 09:00 AM (UK Time) on the day of submission deadline.

Assessment Mark

The project will be assess by submission of a project report, code and presentation. The report is worth 30 marks, code - 35 marks and presentation - 35 marks. 50% of the total mark will go towards your final module score.

The breakdown of report marks is as follows:

- Abstract - 5%
- Introduction - 5%
- Aims and Objectives - 5%
- Methodology - 35%
- Results and Discussion - 30%
- Conclusion - 7%
- References - 3%
- Homework - 10%

Assessment Feedback

Your tutor will mark the assessment and provide you with a written feedback sheet. You can use this feedback to guide your further learning on the module.

Background

Lexer

The transformation of source code to tokens is called *Lexical Analysis*. It's done by the *lexer* (also called *tokenizer* or *scanner*). *Tokens* itself are small, easily configurable datastructures. This is an example of input that can be given to your lexer:

```
let x = 5;
```

And what comes out of the lexer looks like -

```
[
    TOK_LET,
    TOK_IDENT("x"),
    TOK_EQ,
    TOK_INT_LIT(5),
    TOK_SEMICOLON
]
```

Each of the tokens have the original source code representation attached. `“let"` in the case of `TOK_LET`, `“+"` in the case of `TOK_PLUS`, and so on. Some like `TOK_IDENT` and `TOK_INT_LIT` in the example, also have concrete values thet represent attached: `5`(not `“5”`!) in the case of `TOK_INT_LIT` and `“x"` in the case of `TOK_IDENT`.

NOTE: Whitespace characters don’t show up as tokens.

The following are the list of valid tokens that your lexer should recognize:

- `TOK_IDENT` for variables, functions and constants names.
- `TOK_INT_LIT` for integer literals.
- `TOK_LET` for keyword `let`.
- Special characters -
    - `TOK_SQ_BKT_L` for `[`.
    - `TOK_SQ_BKT_R` for `]`.
    - `TOK_COLON` for `:`.
    - `TOK_SEMICOLON` for `;`.
    - `TOK_COMMA` for `,`.
    - `TOK_EQ` for `=`.
    - `TOK_EOF` to denote the end of a file.
    - `TOK_INVALID` to denote a token not known to the lexer.