Gandi Gaman

ES21BTECH11014

Compilers - II

Assignment - 1

Introduction

This document contains a brief overview of the assignment. I have implemented a lexical analyser using lex tool which reads an input file (input file contains a source program whose rules are given in the problem statement) and generates 2 output files. (one contains the token stream and the other contains the corresponding c program for the input program).

Compilation steps

The source program (source_prog.l) is in the TP1 directory. To test the program on an input file , name the input file as “input.txt” and place it in the same directory as the source program.

To compile the program , execute the following commands on the linux terminal :

1) **lex source_prog.l**

   This generates a lex.yy.c file. To compile the generated c file :

2) **gcc lex.yy.c**

   This generates the executable file. To run the executable file on input file “input.txt” :

3) **./a.out input.txt**

   This will generate 2 files named “tokens_output.txt” and “C_output.txt” in the same directory as the source program.

About the output files

“tokens_output.txt” contains the stream of tokens for the given input file. Each line in this file contains a recognized token along with its classification.

“C_output.txt” contains the corresponding c code equivalent for the input program.

Issues with my implementation

- If there is an ‘_’ operator in the input file , I have assumed that it always denote the square root and I have printed the following in the c code : sqrt(‘id’)
- I have assumed that the ‘_’ operator in the input file will be an integer (2) following it. If there is an identifier following it, then in the corresponding c code ‘id’ ‘id’ will be printed without the ‘_’ symbol.
- If there are multiple ‘_’ operators in a single expression, then in the corresponding c code I am only printing the sqrt() for the first ‘_’ operator.

Regular expressions

I have defined the regular expressions following the rules of the language given in the problem statement

The naming convention I used in the source code is easily understood and the regular expressions I used are self explanatory in the code.

I faced some difficulties while writing the regular expressions for string literals and to recognize the invalid identifier.

I have defined a regular expression for the square root operator as {id}{blank}[_]{blank}{number} so that it will be easy to handle this operator.

**********END**********