National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

CS 354: Compiler Construction

Class: BSCS-5C

Lab [02]: Lexical Analysis – Part I

Date: 11th Sep, 2018

Time: [2:00pm – 4:50pm]

Instructor: Dr. Sohail Iqbal

Lab Engineer: Azaz Farooq

National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Lab [02]: Lexical Analysis – Part I

Introduction

The lexical analyzer is the part of the compiler that reads the source program and performs certain secondary tasks at the user interface. One such task is stripping out comments and white space in the form of blanks, tabs and new line characters, from the source program. Another is correlating error messages from the compiler with the source program i.e. keeping a correspondence between errors and source line numbers.

Objectives

1. Successful understanding/implementation Lexical Analysis in C/C++/Java

Tools/Software Requirement

1. gcc, g++, javac, GNU Make

Description

Lexical analysis is the process of converting a sequence of characters into a sequence of tokens. A program or function which performs lexical analysis is called a lexical analyzer, lexer or a scanner. A lexer often exists as a single function which is called by a parser or another function.

Lab Tasks

We will be considering a *subset* of C/C++ language as our source language. In particular, your scanner implementation should return tokens for the following *C/C++ features/specifications*:

Keywords: break, case, char, const, continue, default, double, else, enum, extern, float, for, goto, if, int, long, return, short, static, struct, switch, void, while
Ref: http://tigcc.ticalc.org/doc/keywords.html

Arithmetic Operators: + - * / % ++ --

Relational Operators: == != > < >= <=

Punctuators: {} () [] = , . ; :

Identifiers and Numbers Floating Point and Integer Numbers

1. Design a comprehensive DFA for the above language specifications using the guidelines given in class.
2. Write skeleton code in C/C++/Java for the above DFA, including stub functions for whitespace/comments removal, keyword lookups, identifiers and symbol-table management.
3. Write code for removing comments from the source code
   a) Single line comments
   b) Multi-line comments

Test your implementation using the **“input_scanner.c”** and **“input_scanner.h”** source files provided on LMS.

NOTE: Part II of this lab will add additional tasks to the same scanner implementation, so please ensure that you keep your implementation code in a safe place!

Deliverables

You are required to upload your task (Sources & PDF document) using the link created on LMS followed by a viva.