Compilers Phase III

12.05.2018

Contributors :

Ahmed Khaled Saad	07
Ahmed Reda Amin	09
Hisham Osama	78

Overview

This phase of the assignment aims to practice techniques of constructing semantics rules to generate intermediate code , Where the generated bytecode must follow Standard bytecode instructions defined in Java Virtual Machine Specification .

Specifications

1-Write the semantics rules of the context free grammar described in the problem statement of phase 2 . Use tools like bison to convert a context - free grammar and semantics rules into a parse tree .

2 - The semantics rules are used to output a byte code that follows the standard java bytecode instructions.

3-Generated bytecode can be tested using any of the Java Bytecode tools.

Data Structures Used :

-Symbol Table : A symbol table for storing variable names and respective locations .

-vector<int>: This data structure is widely used in creating **true_list**, **false_list** and **next_list** that are responsible of tracking the flow of the parser to be able to insert missing labels in the code. That is to implement backpatching

Algorithms and Techniques :

- We have chosen to implement our parser in backpatching mode to be able to generate the java byte code in just a single pass (where a 2 pass generator will be considered a great overhead for our simple compiler)

  In order to use this technique we have to implement 3 main functions

  - make_list
  - merge
  - back_patch

  These functions are responsible for guaranteeing a single pass parser

  We have also implemented two helper productions: **Marker** and **Next_Marker** that helped us in identifying the actual values of the PC that we should fill the missing go to with

Comments about Used Tools :

- Flex : Flex uses C++ as opposed to the C only nature of lex and it can take its data type definitions from a file generated by Bison which is the *.tab.h .
- YACC
  - It has the same structure of LEX
  - We defined the required semantic rules in this file

Explanation of Functions :

- Make_list
  - It is used to create a list of the locations where it should be filled later with the appropriate data
  - It's commonly used during the initialization of the true_list, false_list and next_list
- Merge
  - It is used to get the UNION of 2 lists
- Back_patch
  - It's used to fill the back the GO TO instructions (or any other instruction that need an address) with the appropriate address

Assumptions and Final Thoughts :

Assumptions :

1- The input is assumed to be one file at a time and not a stream of back to back files .

1- Using FOR loop is subject to the following format

FOR (DECLARATION BOOLEAN_EXPRESSION ; DECLARATION)

where the DECLARATION should end with a semicolon

EX: for (i = 0; i < 5; i = i + 1**;**)