40-414 – Compiler Design, Spring 1402, Sharif University of Technology

Programming Assignment IV

(Intermediate Code Generator - Part 2)

(Plus one Optional Task)

1 Introduction

In this programming assignment, you are supposed to complete the intermediate code generator of your compiler for C-minus. Please note that you may use codes from text books, with a reference to the used book in your code. However, using codes from the internet and/or other students in this course is **strictly forbidden** and may result in Fail grade in the course. Besides, even if you did not implement the parser in assignment II, you may not use the parsers from other students. In such a case, you need to implement the parser, too.

2 Intermediate Code Generator Specification

In this assignment, you will implement the second part of the intermediate code generator with the following characteristics:

- The code generator is called by the parser to perform a code generation task, which can be modifying the semantic stack and/or generating a number of three address codes.
- Code generation is performed in the same pass as other compilation tasks are performed (because the compiler is supposed to be a **one pass compiler**).
- Parser calls a function called '**code_gen**' and sends an **action symbol** as an argument to '**code_gen**' at appropriate times during parsing.
- Code generator (i.e., the '**code_gen**' function) executes the appropriate **semantic routine** associated with the received action symbol (based on the technique introduced in Lecture 9).
- Generated three-address codes are saved in an output text file called '**output.txt**'.

3 Augmented C-minus Grammar

To implement the second part of the intermediate code generator, you need to add a few more action symbols to the grammar of C-minus of the grammar of programming assignments II and III. For each action symbol, you need to write an appropriate semantic routine in **Python** that performs the required code generation tasks such as modifying the semantic stack and/or generating a number of three address codes. Note that again **you should not change the given grammar in any ways other than adding the required action symbols to the right hand side of the production rules**.

4 Intermediate Code Generation

The intermediate code generation is performed with the same method that was introduced in Lecture **8**. In the second part of implementing intermediate code generation in this assignment, all constructs supported by the given C-minus grammar are to be implemented including **return** statements and **function calls**. However, you do not need to consider **nested** or **recursive** functions. Besides, in this assignment, all the sample and test cases will be lexically, syntactically, and semantically correct programs. In implementing the required sematic routines for the intermediate code generation, you should pay attention to the following points:

- Every input program may include a number of global variables and a main function with the signature '**void main (void)**'.
- All local variables of the functions are declared at the beginning of the functions. That is, there will not be any declaration of variables inside other constructs such as loops.
- In conditional statements such as 'if' and/or 'while', if the expression value is **zero**, it will be regarded as a '**false**' condition; otherwise, it will be regarded to be '**true**'. Moreover, the result of a '**relop**' operation that is **true**, will be '**1**'. Alternatively, if the result of a '**relop**' operation is '**false**', its value will be '**0**'.
- You should implicitly define a function called '**output**' with the signature '**void output (int a);**' which prints its argument (an integer) as the main program's output.
- Implementation of **recursive functions** is optional (see section 6 below). Successful implementation of the optional part adds 50% to the total mark of this assignment!

5 Available Three address Codes

In this assignment, you can only use exactly the same set three address codes that was explained in section 5 of the description of Programming Assignment III. Besides, three address codes produced by your compiler will be executed by the same '**Tester**' program that was distributed with Programming Assignment III. Please note that again the most important factor in evaluating this assignment is that the output of your code generator will be successfully interpreted by the '**Tester'** program and produce the expected output value.

6 An Optional task

In this assignment, you can optionally improve your compiler so that it can produce three address codes for **recursive programs** such as the following example for computing **Factorial** function. Please note that in order to do this task, you should somehow implement some sort of dynamic memory allocation such as a runtime stack. The three address code for this program should print 120 when it is given to the tester program:

lineno	code
1	int fact (int n)
2	{
3	int f;
4	if (n == 1) f = 1;
5	else f = n * fact (n - 1);
6	return f;
7	}
8	void main (void)
9	{
10	int i;
11	i = fact (5);
12	output (i);
13	}

Fig. 1 C-minus recursive program sample

7 What to Turn In

Before submitting, please ensure you have done the following:

- It is your responsibility to ensure that the final version you submit does not have any debug print statements.
- You should submit a file named '**compiler.py**', which includes the Python code of scanner, transition based predictive parser, and intermediated code generator modules. Please write your **full name(s)** and **student number(s)**, and any reference that you may have used, as a comment at the top of '**compiler.py**'.
- Your parser should be the main module of the compiler so that by calling the parser, the compilation process can start, and the parser then invokes other modules when it is needed.
- The responsibility of showing that you have understood the course topics is on you. Obtuse code will have a negative effect on your grade, so take the extra time to make your code readable.
- Your parser will be tested by running the command line '**python3 compiler.py**' in Ubuntu using Python interpreter version **3.8**. It is a default installation of the interpreter without any added libraries except for '**anytree**', which may be needed for creating the parse trees. No other additional Python's library function may be used for this or other programming assignments. Please do make sure that your program is correctly compiled in the mentioned environment and by the given command before submitting your code. It is your responsibility to make sure that your code works properly using the mentioned OS and Python interpreter.
- Submitted codes will be tested and graded using several different test cases (i.e., several **'input.txt**' files). Your compiler should read **'input.txt**' from the same working directory as that of '**compiler.py**'. In the case of a compile or run-time error for a test case, a grade of zero will be assigned to the submitted code for that test case. Similarly, if the code cannot produce the expected output (i.e., '**output.txt**') for a test case, or if executing **'output.txt**' by the **Tester** program does not produce the **expected** value, again a grade of zero will be assigned to the code for that test case. Therefore, it is recommended that you test your programs on several different random test cases before submitting your code. If you decided to implement the optional part of the assignment, your compiler will also be tested on a number relevant input. Please note that the test cases of compulsory part of the assignment will be a fully correct C-minus program. The print outs of your generated code will be checked against the '**extected.txt**' file.
- In a couple of days, you will also receive **13** input-output sample files (**Ten** cases for the compulsory part and **Three** cases for the optional part).

- Your Compiler will be evaluated by the Quera's Judge System (QJS). These 13 samples will be added to QJS. After the assignment's deadline is passed, **four** new test cases (**three new** cases for the compulsory part and **one new** case for the optional part) will be substituted for some the sample cases of the Quera and your compiler will be rejudged again.

Good Luck!

Ghassem Sani, Gholamreza