CS402 – Compiler Design

Homework 3

Fall 2014-2015

1 Description

In this homework, you will implement a compiler from a programming language called PL4 into the three address code (TAC) that we have studied in our lectures.

You are expected to write a flex/bison scanner parser, that parses PL4 programs, and a code generator which translates the input PL4 program into an equivalent TAC program. The grammars for these languages are given in Section 3 and Section 4.

An optimization as explained in Section 6 will let you get 25 points bonus.

2 Input and output

The input to your parser will be PL4 programs. Your parser should accept the name of the input file and the name of the output file as command line arguments. The usage of your program will be

```
$ > pl4 inputfile outputfile
```

3 PL4 Grammar

$$
\begin{array}{rcl}
\hline
\mathit{Prgrm} & \rightarrow & \mathit{StmtBlk} \\
\mathit{StmtBlk} & \rightarrow & \textbf{begin}\ \mathit{StmtLst}\ \textbf{end} \\
\mathit{StmtLst} & \rightarrow & \mathit{Stmt}\ \mathit{StmtLst} \mid \mathit{Stmt} \\
\mathit{Stmt} & \rightarrow & \mathit{AsgnStmt} \mid \mathit{IfStmt} \mid \mathit{WhlStmt} \\
\mathit{AsgnStmt} & \rightarrow & \textbf{id} = \mathit{Expr}\ ; \\
\mathit{IfStmt} & \rightarrow & \textbf{if}\ (\ \mathit{Expr}\ )\ \textbf{then}\ \mathit{StmtBlk}\ \textbf{else}\ \mathit{StmtBlk} \\
\mathit{WhlStmt} & \rightarrow & \textbf{while}\ (\ \mathit{Expr}\ )\ \mathit{StmtBlk} \\
\mathit{Expr} & \rightarrow & \textbf{intnum} \mid \textbf{false} \mid \textbf{true} \mid \textbf{id} \mid (\mathit{Expr}) \mid \mathit{Expr}\ \mathit{Op}\ \mathit{Expr} \\
\mathit{Op} & \rightarrow & + \mid * \mid < \mid == \mid \textbf{and} \\
\hline
\end{array}
$$

Tokens:

- Integer constant (**intnum**): sequence of digits
- Identifier (**id**) : a letter followed by a sequence of letters and digits
- For other tokens, the lexemes are the same as the token names

Operator precedences (from highest to lowest): $*$, $+$, $<$, $==$, **and**
Operators are left associative.

4 TAC Grammar

$$\begin{array}{rcl}
\hline
Prgrm & \rightarrow & StmtLst \\
StmtLst & \rightarrow & LStmt\ StmtLst \mid LStmt \\
LStmt & \rightarrow & Lbl\ \textbf{:}\ Stmt \mid Stmt \\
Stmt & \rightarrow & AsgnStmt \mid CondJump \mid Jump \\
AsgnStmt & \rightarrow & \textbf{id} = Expr \\
IfStmt & \rightarrow & \textbf{if}\ (\ \textbf{id}\ )\ \textbf{GOTO}\ Lbl \\
Jump & \rightarrow & \textbf{GOTO}\ Lbl \\
Lbl & \rightarrow & \textbf{id} \\
Expr & \rightarrow & Operand \mid Operand\ Op\ Operand \\
Operand & \rightarrow & \textbf{intnum} \mid \textbf{false} \mid \textbf{true} \mid \textbf{id} \\
Op & \rightarrow & + \mid * \mid < \mid == \mid \textbf{and} \\
\hline
\end{array}$$

5 Output Format

As mentioned above, the output of your program will be an equivalent TAC program. Note that, in TAC grammar, we require the condition of a conditional jump to be an identifier. This requirement, together with the fact that PL4 expressions can have more than one operator, will force you to introduce temporary variables. Assume that an optimizer will be applied to your TAC program to reduce the number of temporary variables. Hence, use a fresh one every time you need a temporary variable.

In your output program, every TAC statement must be emitted on a separate line.

6 Optimization

While introducing temporary variables, we depend on an optimizer, and hence be ignorant to the number of temporary variables inserted.

You may implement a TAC optimizer for an additional 25 points. TAC optimizer should accept a TAC program as input, and produce an equivalent TAC program which uses possibly less number of temporary variables.

You may assume that, the inputs to the TAC optimizer will be the TAC programs produced by your PL4 compiler. Please notice that, the way we generate temporary variables in the PL4 compiler allows a very simple optimization. The life time of a temporary variable ends when it is used. There will not be another use of it in the remaining statements. Hence after the statement that uses the temporary variable, that temporary variable can be reused for some other purpose. There is the problem of type consistency (e.g. a temporary variable is used for an integer value, then for a boolean value). However, do not pay attention to this problem.

Your optimizer's usage must be

```
$> tacopt inputfile outputfile
```

7 How to submit

Submit a single tar.gz package on SUCourse that contains

- all the sources you've developed,
- Makefile that can be used to build your parser
- A word document containing your grammar (whether you modify it or not).

Your tar.gz file must be named as id.tar.gz, where id is your student id.

The Makefile should produce an executable named as **pl4** (and also **tacopt** if you decide to go for the bonus). Note that, the executables will be tested on `flow.sabanciuniv.edu`, so we recommend that you, at least, test your implementation on this machine before submitting.

8 Notes

- **Important**: SUCourse's clock may be off a couple of minutes. Take this into account to decide when to submit.
- No homework will be accepted if it is not submitted using SUCourse.
- Start working on the homework immediately.