University of Central Florida
Department of Computer Science
COP 3402: System Software
Fall 2021

Homework #4 (PL/0 Compiler)

Due November 17th 2021 by 11:59 p.m.

REQUIRMENT:
All assignments must compile and run on the Eustis server. Please see course website for details concerning use of Eustis.

Objective:
In this assignment, you must implement a PL/0 compiler as you did in HW3, but this time using your own VM (HW1) and your own Scanner (HW2). In this implementation, you must do some modifications to your VM, to the grammar, and to the scanner.

1) In your VM (HW1), you must modify the instruction **call** and **return** as shown below**.** The AR of the modified VM must have: FV, SL, DL, and RA (in that order).

CAL L, M
```
stack[sp - 1] ← 0              /* Functional Value(FV)
stack[sp - 2] ← base(L);       /* static link (SL)
stack[sp - 3] ← bp;            /* dynamic link (DL)
stack[sp - 4] ← pc             /* return address (RA)
bp ← sp - 1;
pc ← M;
```

RTN 0, 0
```
sp ← bp + 1
pc ← stack[sp - 4]
bp ← stack[sp - 3])
```

In HW3 for Symbol table instead of adding 3, add 4
Change increment
Symbol table address
Tokens match for if/when

2) The modified PL/0 grammar is defined as:

EBNF of tiny PL/0:

program ::= block "**.**" **.**
block ::= const-declaration var-declaration procedure-declaration statement**.**
const-declaration ::= ["**const**" ident ":=" number {"**,**" ident ":=" number} "**;**"]**.**
var-declaration ::= ["**var** "ident {"**,**" ident} "**;**"]**.**
procedure-declaration ::= { "**procedure**" ident "**;**" block "**;**" }.
statement ::= [ident "**:=**" expression
| "**call**" ident
| "**do**" statement { "**;**" statement } "**od**"
| "**when**" condition "**do**" statement ["**elsedo**" statement]
| "**while**" condition "**do**" statement
| "**read**" ident
| "**write**" expression
| **ε**] **.**
condition ::= "**odd**" expression
| expression rel-op expression**.**
rel-op ::= "**=**"|"**!=**"|"**<**"|"**<=**"|"**>**"|"**>=**"**.**
expression ::= ["+"|"-"] term { ("+"|"-") term}**.**
term ::= factor {("***"|"/"|"%") factor}**.**
factor ::= ident | number | "**(**" expression "**)**"**.**
number ::= digit {digit}**.**
ident ::= letter {letter | digit}**.**
digit ;;= "**0**" | "**1**" | "**2**" | "**3**" | "**4**" | "**5**" | "**6**" | "**7**" | "**8**" | "**9**"**.**
letter ::= "**a**" | "**b**" | … | "**y**" | "**z**" | "**A**" | "**B**" | ... | "**Y**" | "**Z**"**.**

Based on Wirth's definition for EBNF we have the following rule:
[ ] means an optional item.
{ } means repeat 0 or more times.
Terminal symbols are enclosed in quote marks.
A period is used to indicate the end of the definition of a syntactic class.

Replace if for do, and the rest of the tokens. Do not add them to the allowed symbols. For example, "if" was a reserved word before, now "if" would be an identifier

If you follow the grammar, the number of errors that you must handle will emerge in a natural way. When creating your error list, pay attention to this "augmented Error List" example.

1. Program must be closed by a period
2. Constant declarations should follow the pattern **ident ":=" number {"," ident ":=" number}**
3. Variable declarations should follow the pattern **ident {"," ident}**
4. Procedure declarations should follow the pattern **ident ";"**
5. Variables must be assigned using :=
6. Only variables may be assigned to or read
7. call must be followed by a procedure identifier
8. when must be followed by do - found in **statement** in the when case when flow of control returns from **condition** and the current symbol is not do.
9. while must be followed by do
10. Relational operator missing from condition
11. Arithmetic expressions may only contain arithmetic operators, numbers, parentheses, constants, and variables
12. "(" must be followed by)
13. Multiple symbols in variable and constant declarations must be separated by commas
14. Symbol declarations should close with a semicolon
15. Statements within do-od must be separated by a semicolon - found in **statement** when the od symbol is expected but one of the following is found instead: identifier, read, write, do, call, when, or while
16. do must be followed by od - found in **statement** when the od symbol is expected and the symbol present is neither od, identifier, read, write, do, call, when, nor while
17. Bad arithmetic
18. Conflicting symbol declarations
19. Undeclared identifier

Note: you are free to add, delete, extend, and or modify the list of errors.

In the next page you will find two program examples.

Program examples:

Example of a program written in modified PL/0:

```
var x, w;
do
    x:= 4;
    read w;
    when w > x do
            w:= w + 1
    elsedo
            w:= x;
    write w
od.
```

Example 2: You can use this example (recursive program) to test your compiler:

```
var f, n;
procedure fact;
        var ans1;
        do
                ans1:=n;
                n:= n-1;
                when n = 0 do f := 1;
                when n > 0 do call fact;
                f:=f*ans1;
        od;

do
    n:=3;
    call fact;
    write f
od.
```

3) Modify your scanner (HW2) to include tokens associated to the new keywords as defined in the modified grammar.

Reminder:

The compiler must read a program written in modified PL/0 and generate code for the Virtual Machine (VM) you implemented in HW1. Your compiler must neither parse nor generate code for programming constructs that are not in the grammar described above. **For example, if your compiler parse and generate code for "If x >2 then x:= x + 1;" and it runs, your grade will be zero.**

Rubric

15 – Compiles
05 – README.txt containing author names
20 - produces correct token table and lexeme list
20 - produces correct symbol table and handles errors correctly
20 - produces correctly parsed code
20 - executes code correctly

Except for the modifications described in this document, The rule of the game for this assignment (HW4) are the same used in HW3.

To Do List:

1. Correct your HW1
2. Make the adjustments to return and call in HW1 specified above.
3. Transfer your HW1 to the skeleton vm.c and make sure it works.
4. Correct your HW2
5. Make adjustments to reserved words list as specified above in the grammar and in the enum.
6. Correct your HW3
7. Make adjustments to your HW3 to reflect the changes (HINT: if your HW3 was perfect, you should only need to do two things: update the token type names AND adjust the emit INC call in block for subprocedures to reflect the larger AR size).

Important: Do not forget to pay attention to the addresses in the symbol table because the AR is larger now.