COP 3402 Systems Software

Fall 2016

Instructor: Dr. Pawel Wocjan

Midterm Exam

Wednesday 10/26/2016

There are 4 problems. Each problem is worth 5 points.

First Name: _____

Last Name: _____

PID: _____

Problem

1 _____

2 _____

3 _____

4 _____

Total _____

Problem 1 – PM/0 machine

Name three registers of the PM/0 machine and briefly describe their purpose.

Register	Purpose

What is stored in an activation record? Name two entries and describe briefly their purpose.

Entry	Purpose

Problem 2 – Lexical analysis

Output the token tables for the following two PL/0 program fragments. Stop scanning if a lexicographical error occurs and report a suitable corresponding error message.

```
identsym,    numbersym,   plussym,     minussym,    multsym,     slashsym,
oddsym,      eqlsym,      neqsym,      lessym,      leqsym,      gtrsym,
geqsym,      lparentsym,  rparentsym,  commasym,    semicolonsym, periodsym,
becomessym,  beginsym,    endsym,      ifsym,       thensym,     whilesym,
dosym,       callsym,     constsym,    varsym,      procsym,     writesym,
readsym,     elsesym
```

Identifiers can be a maximum of 12 characters in length. Identifiers must start with a letter symbol. Numbers must be smaller than 2^16.

Program 1:

```
x := y;
if x <> 5 then
```

Table 1:

Token type	Semantic value

Error: _____

Program 2:

```
const one = 1;
var num, 2num;
```

Table 2:

Token type	Semantic value

Error: _____

Problem 3 – Ambiguous grammars / Parse trees

Show that the following grammar is ambiguous by choosing an appropriate string of the language generated by the grammar and by drawing two different parse trees for your string.

```
S -> A

A -> A + A | A – A | id
```

Problem 4 – Extended Backus Naur Form / Syntax diagram / Recursive decent parser

You are given the following portion of the EBNF specification of PL/0:

```
<var-declaration> ::= [ “var” <ident> { “,” <ident> } “;” ]
```

a) Draw the corresponding syntax diagram.

<var-declaration> ::= [“var” <ident> { “,” <ident> } “;”]

b) Using C-like pseudocode, implement the function of the recursive descent parser that parses **<var-declaration>**. Assume that there is a global variable `tok` and that the function `advance()` loads the next token into `tok`. Use the token types listed in Problem 2 in your pseudocode.

Scratch paper: