**COP 3402 Systems Software**

**Summer 2015**

**Instructor: Dr. Pawel Wocjan**

**Second Exam**

**Wednesday 07/22/2015**

There are 6 problems (5 problems + 1 bonus problem). Each problem is worth 5 points.

**First Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Last Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Problem 1: \_\_\_\_\_\_\_\_\_**

**Problem 2: \_\_\_\_\_\_\_\_\_**

**Problem 3: \_\_\_\_\_\_\_\_\_**

**Problem 4: \_\_\_\_\_\_\_\_\_**

**Problem 5: \_\_\_\_\_\_\_\_\_**

**Problem 6: \_\_\_\_\_\_\_\_\_**

**Total: \_\_\_\_\_\_\_\_\_**

**First Name/Last Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Problem 1 – Parsing and Code Generation Errors**

Indicate for the following five tiny PL/0 programs if

**(A)** PM/0 code can be successfully generated,

**(B)** there is a parsing error, or

**(C)** there is a code generation error.

If an error occurs, give a short description of the error. Refer to Appendix A for the grammar for tiny PL/0.

var a;

begin

read a;

write a;

end **( \_\_ ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

const b = 10;

var a, b;

begin

read a;

a = a + b;

write a;

end **.** **( \_\_ ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

const a;

var b;

begin

read b;

b = b + a;

write b;

end **.** **( \_\_ ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

var in, out;

begin

read in;

out = 0;

if in > 10 then out = 1;

write out;

end **.** **( \_\_ ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

const a = 10;

var b;

begin

read a;

b = a + 2;

write b;

end **.** **( \_\_ ) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**First Name/Last Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Problem 2 – Symbol Table**

Give the contents of the symbol table for the tiny PL/0 program by completing the provided table. List the symbols in the symbol table in the order in which they are declared in the PL/0 program.

typedef struct symbol {

int kind; // const = 1, var = 2, proc = 3

char name[12]; // name up to 11 chars

int val; // number (ASCII value)

int level; // L level

int addr; // M address

} symbol;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | first symbol | second symbol | third symbol | fourth symbol | fifth symbol |
| kind |  |  |  |  |  |
| name |  |  |  |  |  |
| val |  |  |  |  |  |
| level |  |  |  |  |  |
| addr |  |  |  |  |  |

const secret = 10, max = 5;

var guess, num;

begin

num = 1;

read guess;

while guess <> secret do begin

num = num + 1;

read guess;

end;

if num <= max write num;

end **.**

**First Name/Last Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Problem 3 – Code Generation**

Generate the PM/0 code for the tiny PL/0 program, g. Refer to Appendix B for the definition of the PM/0 instructions. **You must use the symbolic names**. For instance, write **LIT 0 2** instead of 01 0 2 and **OPR 0 ADD** instead of 02 0 2.

const s = 9, one = 1;

var g;

begin

read g;

if g = s then write one;

end **.**

Address: Instruction:

0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

9 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

10 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

11 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

12 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

13 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

14 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

15 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**First Name/Last Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Problem 4 – Ambiguous Grammars**

Show that the following grammar is ambiguous (1) by choosing an appropriate string of the language generated by the grammar and (2) by presenting two different parse trees for your string.

**S -> A**

**A -> A + A | A – A | id**

**First Name/Last Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Problem 5 – Parser Code**

Given the following row of a predictive parsing table, write C-like pseudocode for the corresponding parsing function for a recursive-descent parser.

Assume that the function void getToken() gets the next token and stores it in the global int variable token. For instance, if you wanted to check if the current token is equal to ‘\*’, you could write if (token == multsym).

The other constants are: identsym, plussym, lparensym, rparensym.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **id** | **+** | **\*** | **(** | **)** |
| **F** | **F->id** |  |  | **F->(id)** |  |

**First Name/Last Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Problem 6 (Bonus) – Stack Trace of PM/0 Program**

Given the program in interpreted assembly language for virtual machine PM/0 below, show the stack and register values (pc, bp, and sp) at the end of the execution.

If there are multiple Activation Records, separate them with the symbol “|”. Do not show any stack content above sp. Refer to Appendix B if needed. Initial values for the PM/0 CPU registers: sp = 0; bp = 1; pc = 0.

**Write your final answer into the slots below:**

**pc: \_\_\_\_\_\_ bp: \_\_\_\_\_\_ sp: \_\_\_\_\_\_ stack: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Line OP L M

0 JMP 0 8

1 JMP 0 2

2 INC 0 4

3 LOD 1 4

4 LOD 1 5

5 OPR 0 ADD

6 STO 1 4

7 OPR 0 RET

8 INC 0 4

9 LIT 0 4

10 LIT 0 3

11 CAL 0 2

12 SIO 0 2

**Appendix A** (grammar for tiny PL/0 in EBNF)

program ::= block “.**”** **.**

block ::= const-declaration var-declaration statement **.**

const-declaration ::= [ “**const”** ident “**=**” number {“**,**” ident “**=**”

number} “**;**” ] **.**

var-declaration ::= [ “**var”** ident { “**,**” ident } “**;**” ] **.**

statement ::= [ ident “**:=**" expression

| “**begin**” statement { “**;**” statement } “**end**”

| “**if**” condition “**then**” statement

| “**while**” condition “**do**” statement

| “**read**” ident

| “**write**” ident

| **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**” **.**

**Appendix B** (PM/0 instructions)

01 **LIT 0 M**

sp = sp + 1;

stack[sp] = **M**;

02 **OPR** **0 M**

0 **RET** sp = bp – 1; pc = stack[sp + 4]; bp = stack[sp + 3];

1 **NEG** stack[sp] = -stack[sp];

2 **ADD** sp = sp – 1; stack[sp] = stack[sp] + stack[sp + 1];

3 **SUB** sp = sp – 1; stack[sp] = stack[sp] - stack[sp + 1];

4 **MUL** sp = sp – 1; stack[sp] = stack[sp] \* stack[sp + 1];

5 **DIV** sp = sp – 1; stack[sp] = stack[sp] / stack[sp + 1];

6 **ODD** stack[sp] = stack[sp] mod 2;

7 **MOD** sp = sp – 1; stack[sp] = stack[sp] mod stack[sp + 1];

8 **EQL** sp = sp – 1; stack[sp] = stack[sp] == stack[sp + 1];

9 **NEQ** sp = sp – 1; stack[sp] = stack[sp] != stack[sp + 1];

10 **LSS** sp = sp – 1; stack[sp] = stack[sp] < stack[sp + 1];

11 **LEQ** sp = sp – 1; stack[sp] = stack[sp] <= stack[sp + 1];

12 **GTR** sp = sp – 1; stack[sp] = stack[sp] > stack[sp + 1];

13 **GEQ** sp = sp – 1; stack[sp] = stack[sp] >= stack[sp + 1];

03 **LOD L M**

sp = sp + 1;

stack[sp] = stack[ base(**L, bp**) + **M**];

04 **STO** **L M**

stack[ base(**L, bp**) + **M**] = stack[ sp ];

sp = sp - 1;

05 **CAL L M**

stack[sp + 1] = 0;

stack[sp + 2] = base(**L, bp**);

stack[sp + 3] = bp;

stack[sp + 4] = pc;

bp = sp + 1;

pc = **M**;

06 **INC 0 M**

sp = sp + **M**;

07 **JMP 0 M**

pc = **M**;

08 **JPC 0 M**

**if** (stack[ sp ] == 0 ) **then {** pc = **M; }**

sp = sp - 1;

09 **SIO** **0 0**

print(stack[ sp ]);

sp = sp – 1;

09 **SIO 0 1**

sp = sp + 1;

read(stack[ sp ]);

09 **SIO 0 2** halt;

**NOTE**: The result of a logical operation such as (A > B) is defined as 1 if  
the condition was met and 0 otherwise.