Jonathan Buckner
COSC 4316 01 Java0 Language Translator
Option B-A

Translator successfully implements:

- Table driven FSA & PDA
- Symbolic and Numeric addresses
- Declarations (CONST, VAR) and Numeric Literals updating in Symbol table, as well as updating the .data and .bss section in Assembly from Symbol table
- Compound statements, source listing
- Integer I/O
- Arithmetic Assignment Statements
- IF/THEN/ELSE statements that can be nested
- WHILE/DO statements that can be nested
- All relational operators
- Some error checking

Table of Contents

Pass One State Table	3
Pass Two State Table	5
Syntax Precedence Table	6
Pass One Diagram	10
Pass Two Diagram	11
Full Translator Code	12
Example Programs	36
Question One (Arithmetic)	37
Question Two (Nested IF/THEN/ELSE)	46
Question Three/Five (NFactorial Using a Function)	56
Question Four (Nested Whiles)	64
Bonus Question Five (5 Nested IFS)	73

Pass One State Table:

state/token	space	*	digit	letter	/	"="	"<"	">"
0	0	2	3	5	7	11	14	17
1 (error)	0	0	0	0	0	0	0	0
2(create "*")	0	0	0	0	0	0	0	0
3	4	4	3	4	4	4	4	4
4 (create <int>)	0	0	0	0	0	0	0	0
5	6	6	5	5	6	6	6	6
6(create <var>	0	0	0	0	0	0	0	0
7	10	8	10	10	10	10	10	10
8	8	9	8	8	8	8	8	8
9	8	8	8	8	0	8	8	8
10(create "/")	0	0	0	0	0	0	0	0
11	12	12	12	12	12	13	12	12
12(create "=")	0	0	0	0	0	0	0	0
13(create "==")	0	0	0	0	0	0	0	0
14	15	15	15	15	15	16	15	15
15(create "<")	0	0	0	0	0	0	0	0
16(create "<=")	0	0	0	0	0	0	0	0
17	18	18	18	18	18	19	18	18
18 (create ">")	0	0	0	0	0	0	0	0
19 (create ">=")	0	0	0	0	0	0	0	0
20 (create ",")	0	0	0	0	0	0	0	0
21 (create ";")	0	0	0	0	0	0	0	0
22(create "+")	0	0	0	0	0	0	0	0
23 (create "-")	0	0	0	0	0	0	0	0
24 (create "(")	0	0	0	0	0	0	0	0
25(create ")")	0	0	0	0	0	0	0	0
26(create"{")	0	0	0	0	0	0	0	0
27(create "}")	0	0	0	0	0	0	0	0
28	1	1	1	1	1	29	1	1
29(create "!=")	0	0	0	0	0	0	0	0

state/token	","	";"	"+"	"-"	"("	")"	"{"	"}"
0	20	21	22	23	24	25	26	27
1 (error)	0	0	0	0	0	0	0	0
2(create "*")	0	0	0	0	0	0	0	0
3	4	4	4	4	4	4	4	4
4 (create <int>)	0	0	0	0	0	0	0	0
5	6	6	6	6	6	6	6	6
6(create <var>	0	0	0	0	0	0	0	0
7	10	10	10	10	10	10	10	10
8	8	8	8	8	8	8	8	8
9	8	8	8	8	8	8	8	8
10(create "/")	0	0	0	0	0	0	0	0
11	12	12	12	12	12	12	12	12
12(create "=")	0	0	0	0	0	0	0	0
13(create "==")	0	0	0	0	0	0	0	0
14	15	15	15	15	15	15	15	15
15(create "<")	0	0	0	0	0	0	0	0
16(create "<=")	0	0	0	0	0	0	0	0
17	18	18	18	18	18	18	18	18
18 (create ">")	0	0	0	0	0	0	0	0
19 (create ">=")	0	0	0	0	0	0	0	0
20 (create ",")	0	0	0	0	0	0	0	0
21 (create ";")	0	0	0	0	0	0	0	0
22(create "+")	0	0	0	0	0	0	0	0
23 (create "-")	0	0	0	0	0	0	0	0
24 (create "(")	0	0	0	0	0	0	0	0
25(create ")")	0	0	0	0	0	0	0	0
26(create"{")	0	0	0	0	0	0	0	0
27(create "}")	0	0	0	0	0	0	0	0
28	1	1	1	1	1	1	1	1
29(create "!=")	0	0	0	0	0	0	0	0

state/token	"!"
0	28
1 (error)	0
2(create "*")	0
3	4
4 (create <int>)	0
5	6
6(create <var>	0
7	10
8	8
9	8
10(create "/")	0
11	12
12(create "=")	0
13(create "==")	0
14	15
15(create "<")	0
16(create "<=")	0
17	18
18 (create ">")	0
19 (create ">=")	0
20 (create ",")	0
21 (create ";")	0
22(create "+")	0
23 (create "-")	0
24 (create "(")	0
25(create ")")	0
26(create"{")	0
27(create "}")	0
28	1
29(create "!=")	0

Pass Two State Table:

state/token	class	<var>	${	CONST	$=	<int>	$;	VAR
0	1	13	13	13	13	13	13	13
1	13	2	13	13	13	13	13	13
2	13	13	3	13	13	13	13	13
3	13	10	13	4	13	13	13	8
4	13	5	13	13	13	13	13	13
5	13	13	13	13	6	13	13	13
6	13	13	13	13	13	7	13	13
7	13	13	13	13	13	13	3(add constant)	13
8	13	9	13	13	13	13	13	13
9	13	13	13	13	13	13	3(add variable)	13
10	10	10	10	10	10	11(add int literal)	10	10
11	10	10	10	10	10	10	10	10
12(end state)	-1	-1	-1	-1	-1	-1	-1	-1
13(error case)	-1	-1	-1	-1	-1	-1	-1	-1
14 procedure	13	15	13	13	13	13	13	13
15	13	13	13	13	13	13	13	13
16	13	13	13	13	13	13	13	13
17	13	13	3	13	13	13	13	13

state/token	$,	EOF	anything else	PROCEDURE	$(	$)	READ
0	13	13	13	13	13	13	13
1	13	13	13	13	13	13	13
2	13	13	13	13	13	13	13
3	13	13	10	14	13	13	10
4	13	13	13	13	13	13	13
5	13	13	13	13	13	13	13
6	13	13	13	13	13	13	13
7	4(add constant)	13	13	13	13	13	13
8	13	13	13	13	13	13	13
9	8(add variable)	13	13	13	13	13	13
10	10	12	10	14	10	10	10
11	10	12	10	10	10	10	13
12(end state)	-1	-1	-1	-1	13	13	13
13(error case)	-1	-1	-1	-1	13	13	13
14 procedure	13	13	13	13	13	13	13
15	13	13	13	13	16	13	13
16	13	13	13	13	13	17	13
17	13	13	13	13	13	18	13

Syntax Analyzer Precedence Table:

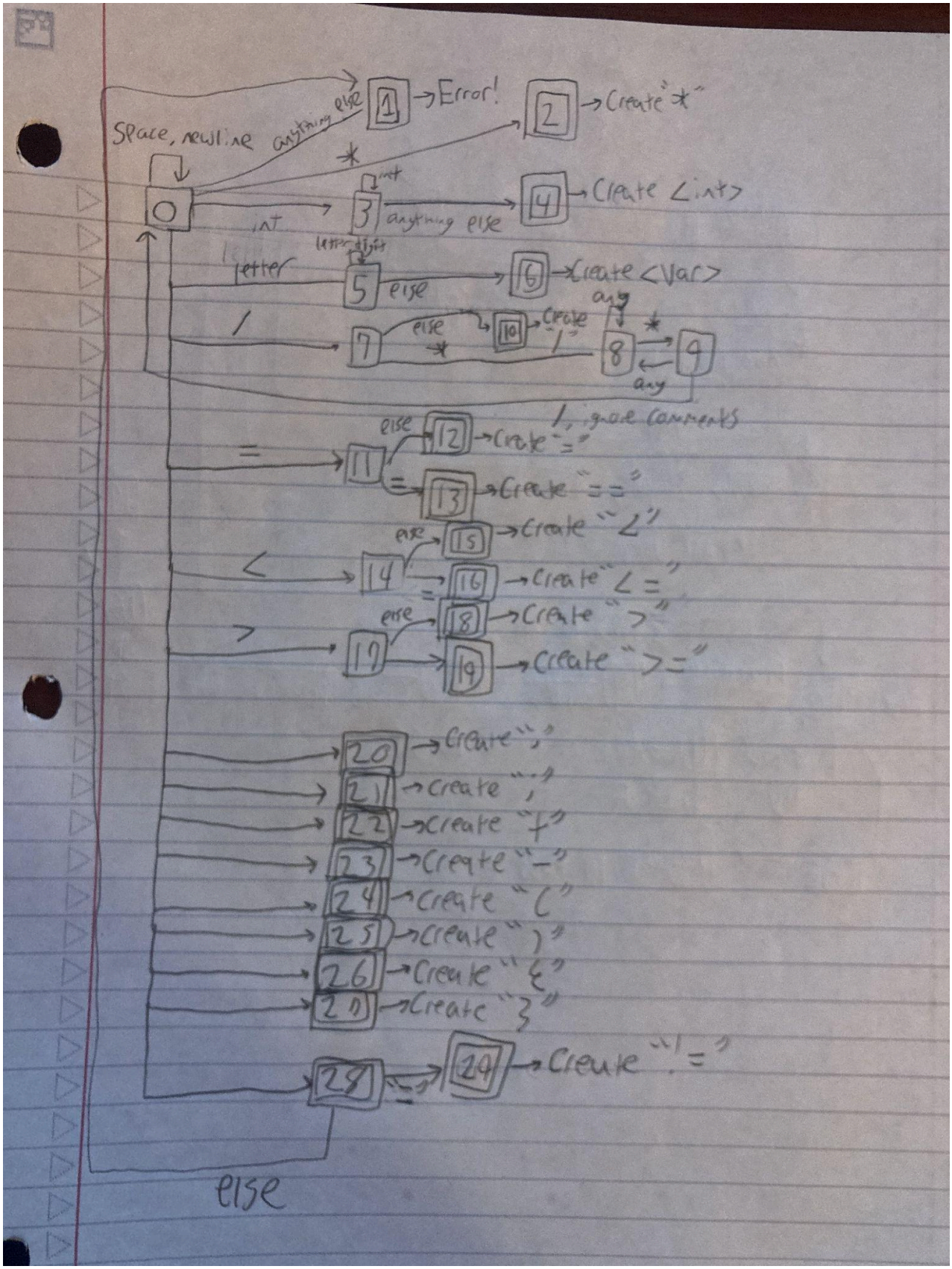
tokens	";"	"="	"+"	"-"	"("	")"	"*"	"/"
";"	" "	<	" "	" "	" "	" "	" "	" "
"="	>	" "	<	<	<	" "	<	<
"+"	>	" "	>	>	<	>	<	<
"-"	>	" "	>	>	<	>	<	<
"("	">"	" "	<	<	<	"="	<	<
")"	>	" "	>	>	" "	>	>	>
"*"	>	" "	>	>	<	>	>	>
"/"	>	" "	>	>	<	>	>	>
"IF"	" "	" "	<	<	<	" "	<	<
"THEN"	" "	<	" "	" "	" "	" "	" "	" "
"ODD"	" "	" "	<	<	<	" "	<	<
"=="	" "	" "	<	<	<	" "	<	<
"!="	" "	" "	<	<	<	" "	<	<
">"	" "	" "	<	<	<	" "	<	<
"<"	" "	" "	<	<	<	" "	<	<
">="	" "	" "	<	<	<	" "	<	<
"<="	" "	" "	<	<	<	" "	<	<
"{"	" "	" "	" "	" "	" "	" "	" "	" "
"}"	" "	" "	" "	" "	" "	" "	" "	" "
"CALL"	" "	" "	" "	" "	" "	" "	" "	" "
var	">"	">"	">"	">"	">"	">"	">"	">"
WHILE	" "	" "	"<"	"<"	"<"	" "	"<"	"<"
DO	" "	"<"	"<"	"<"	" "	" "	"<"	"<"
EOF	" "	" "	" "	" "	" "	" "	" "	" "
READ	">"	" "	" "	" "	" "	" "	" "	" "
PRINTNUMBER	">"	" "	" "	" "	" "	" "	" "	" "
PRINTSTRING	">"	" "	" "	" "	" "	" "	" "	" "
ELSE	" "	" "	<	<	<	" "	<	<

tokens	"IF"	"THEN"	"ODD"	"=="	"!="	">"	"<"	">="
","	" "	" "	" "	" "	" "	" "	" "	" "
"="	" "	" "	" "	" "	" "	" "	" "	" "
"+"	" "	>	" "	>	>	>	>	>
"-"	" "	>	" "	>	>	>	>	>
"("	" "	" "	" "	" "	" "	" "	" "	" "
")"	" "	" "	" "	" "	" "	" "	" "	" "
"*"	" "	>	" "	>	>	>	>	>
"/"	" "	>	" "	>	>	>	>	>
"IF"	" "	"="	<	<	<	<	<	<
"THEN"	<	" "	" "	" "	" "	" "	" "	" "
"ODD"	" "	>	" "	" "	" "	" "	" "	" "
"=="	" "	>	" "	" "	" "	" "	" "	" "
"!="	" "	>	" "	" "	" "	" "	" "	" "
">"	" "	>	" "	" "	" "	" "	" "	" "
"<"	" "	>	" "	" "	" "	" "	" "	" "
">="	" "	>	" "	" "	" "	" "	" "	" "
"<="	" "	>	" "	" "	" "	" "	" "	" "
"{"	" "	" "	" "	" "	" "	" "	" "	" "
"}"	" "	" "	" "	" "	" "	" "	" "	" "
"CALL"	" "	" "	" "	" "	" "	" "	" "	" "
var	">"	">"	">"	">"	">"	">"	">"	">"
WHILE	" "	" "	" "	"<"	"<"	"<"	"<"	"<"
DO	"<"	" "	" "	" "	" "	" "	" "	" "
EOF	" "	" "	" "	" "	" "	" "	" "	" "
READ	" "	" "	" "	" "	" "	" "	" "	" "
PRINTNUMBER	" "	" "	" "	" "	" "	" "	" "	" "
PRINTSTRING	" "	" "	" "	" "	" "	" "	" "	" "
ELSE	" "	"="	<	<	<	<	<	<

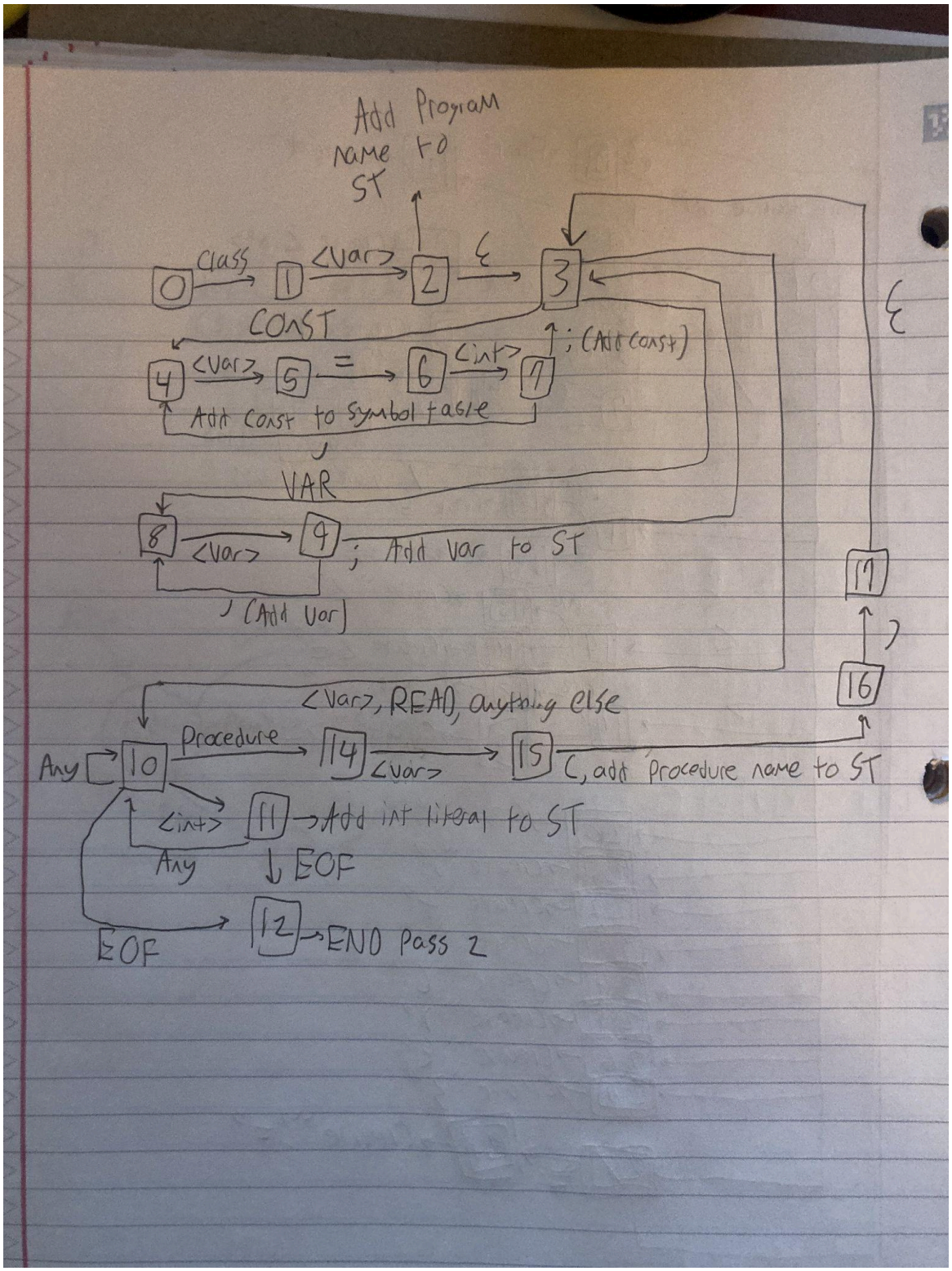
tokens	"<="	"{"	"}"	"CALL"	VAR	WHILE	DO	EOF
";"	" "	" "	" "	" "	"<"	" "	" "	" "
"="	" "	" "	" "	" "	"<"	" "	" "	">"
"+"	>	" "	" "	" "	"<"	" "	" "	">"
"-"	>	" "	" "	" "	"<"	" "	" "	">"
"("	" "	" "	" "	" "	"<"	" "	" "	">"
")"	" "	" "	" "	" "	"<"	" "	" "	">"
"*"	>	" "	" "	" "	"<"	" "	" "	">"
"/"	>	" "	" "	" "	"<"	" "	" "	">"
"IF"	<	" "	" "	" "	"<"	" "	" "	">"
"THEN"	" "	<	" "	<	"<"	" "	" "	">"
"ODD"	" "	" "	" "	" "	"<"	" "	">"	">"
"=="	" "	" "	" "	" "	"<"	" "	">"	">"
"!=	" "	" "	" "	" "	"<"	" "	">"	">"
">"	" "	" "	" "	" "	"<"	" "	">"	">"
"<"	" "	" "	" "	" "	"<"	" "	">"	">"
">="	" "	" "	" "	" "	"<"	" "	">"	">"
"<="	" "	" "	" "	" "	"<"	" "	">"	">"
"{"	" "	" "	">"	" "	"<"	" "	" "	">"
"}"	" "	" "	" "	" "	"<"	" "	" "	">"
"CALL"	" "	" "	" "	" "	"<"	" "	" "	">"
var	">"	">"	">"	">"	" "	">"	">"	">"
WHILE	"<"	" "	">"	" "	"<"	" "	=	">"
DO	" "	"<"	">"	" "	"<"	"<"	" "	">"
EOF	" "	" "	" "	" "	" "	" "	" "	" "
READ	" "	" "	" "	" "	" "	" "	" "	" "
PRINTNUMBER	" "	" "	" "	" "	" "	" "	" "	" "
PRINTSTRING	" "	" "	" "	" "	" "	" "	" "	" "
ELSE	<	" "	">"	" "	"<"	" "	" "	">"

tokens	READ	PRINTNUMBER	PRINTSTRING	ELSE
";"	"<"	"<"	"<"	" "
"="	" "	" "	" "	" "
"+"	" "	" "	" "	" "
"-"	" "	" "	" "	" "
"("	" "	" "	" "	" "
")"	" "	" "	" "	" "
"*"	" "	" "	" "	" "
"/"	" "	" "	" "	" "
"IF"	" "	" "	" "	" "
"THEN"	" "	" "	" "	<
"ODD"	" "	" "	" "	" "
"=="	" "	" "	" "	" "
"!=	" "	" "	" "	" "
">"	" "	" "	" "	" "
"<"	" "	" "	" "	" "
">="	" "	" "	" "	" "
"<="	" "	" "	" "	" "
"{"	" "	" "	" "	" "
"}"	" "	" "	" "	" "
"CALL"	" "	" "	" "	" "
var	" "	" "	" "	">"
WHILE	" "	" "	" "	" "
DO	" "	" "	" "	"<"
EOF	" "	" "	" "	" "
READ	" "	" "	" "	" "
PRINTNUMBER	" "	" "	" "	" "
PRINTSTRING	" "	" "	" "	" "
ELSE	" "	" "	" "	" "

Pass One Diagram:


Space, newline
anything else
1
Error!
2
Create "*"
*
0
int
3
4
Create <int>
int
anything else
letter digit
letter
5
else
6
Create <var>
/
7
else
*
10
Create "/"
any
*
8
9
any
/, make comments
else
12
Create "="
=
11
=
13
Create "=="
else
15
Create "<"
<
14
=
16
Create "<="
else
18
Create ">"
>
17
19
Create ">="
20
Create ";"
21
Create ","
22
Create "+"
23
Create "-"
24
Create "("
25
Create ")"
26
Create "{"
27
Create "}"
28
=
29
Create "!="
else

Pass Two Diagram:



Full Language Translator Code

```
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <stack>

using namespace std;

void ScannerPassOne(string filename, string pass1) {
    ofstream outFile(pass1);
    ifstream inFile(filename);
    if (inFile.is_open() && outFile.is_open()) { // if it's open, do function, else, report failure
        cout << "files successfully open!" << endl;
        string line;
        string tokenFlag = "";
        char letters[52] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
                            'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z' };
        int digits[10] = { 0,1,2,3,4,5,6,7,8,9 };
        int stateTable[30][17] = {
      {0, 2, 3, 5, 7, 11, 14, 17, 20, 21, 22, 23, 24, 25, 26, 27, 28},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {4, 4, 3, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {6, 6, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {10, 8, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10},
      {8, 9, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8},
      {8, 8, 8, 8, 0, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {12, 12, 12, 12, 12, 13, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {15, 15, 15, 15, 15, 16, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {18, 18, 18, 18, 18, 19, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
      {1, 1, 1, 1, 1, 29, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
      {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
        };
        int previousState = 0;
        int state = 0;
        int col = 0;
```

```
        string completedString = "";
        bool finished = false;
        while (getline(inFile, line)) {
            for (int i = 0; i < line.size() + 1; i++) {
                char token = line[i];
                if (token == ' ' || token == '\n') col = 0;
                else if (token == '*') col = 1;
                else if (find(begin(digits), end(digits), token - '0') != end(digits)) col = 2; //check
if its a digit
                else if (find(begin(letters), end(letters), token) != end(letters)) col = 3; //check if
its a letter
                else if (token == '/') col = 4;
                else if (token == '=') col = 5;
                else if (token == '<') col = 6;
                else if (token == '>') col = 7;
                else if (token == ',') col = 8;
                else if (token == ';') col = 9;
                else if (token == '+') col = 10;
                else if (token == '-') col = 11;
                else if (token == '(') col = 12;
                else if (token == ')') col = 13;
                else if (token == '{') col = 14;
                else if (token == '}') col = 15;
                else if (token == '!') col = 16;
                else {
                    col = 0;
                }

                switch (previousState) {
                case 0:
                    previousState = stateTable[previousState][col];
                    break;
                case 1:
                    cout << "Error! Illegal token"; //error case
                    previousState = stateTable[previousState][col];
                    break;
                case 2: //append * case
                    finished = true;
                    tokenFlag = "$*";
                    break;
                case 3:
                    previousState = stateTable[previousState][col];
                    if (previousState == 3) { //does case 4 if we are no longer in case 3
                        break;
                    }
                case 4: //append <int> case
                    finished = true;
                    tokenFlag = "<int>";
                    break;
                case 5:
                    previousState = stateTable[previousState][col];
                    if (previousState == 5) { // does case 6 if we are no longer in case 5
                        break;
                    }
                case 6: //append <var> case
                    finished = true;
                    tokenFlag = "<var>";
                    break;
                case 7:
```

```
        previousState = stateTable[previousState][col];
        break;
    case 8:
        previousState = stateTable[previousState][col]; //dont append if its a comment
        break;
    case 9:
        previousState = stateTable[previousState][col]; //dont append if its a comment again
        if (previousState == 0) { //wipe string if comment is finished
            completedString = "";
            col = 0; //reset token as its currently a /
        }
        break;
    case 10: //append / case
        finished = true;
        tokenFlag = "$/";
        break;
    case 11:
        previousState = stateTable[previousState][col];
        break;
    case 12:
        finished = true;
        tokenFlag = "$="; //append assignment case
        break;
    case 13://append comparison case
        finished = true;
        tokenFlag = "$==";
        break;
    case 14:
        previousState = stateTable[previousState][col];
        break;
    case 15: //append < case
        finished = true;
        tokenFlag = "$<";
        break;
    case 16: //append <= case
        finished = true;
        tokenFlag = "$<=";
        break;
    case 17:
        previousState = stateTable[previousState][col];
        break;
    case 18://append > case
        finished = true;
        tokenFlag = "$>";
        break;
    case 19://append >= case
        finished = true;
        tokenFlag = "$>=";
        break;
    case 20://append , case
        finished = true;
        tokenFlag = "$,";
        break;
    case 21://append ; case
        finished = true;
        tokenFlag = "$;";
        break;
    case 22://append + case
        finished = true;
```

```
    tokenFlag = "$+";
    break;
case 23://append = case
    finished = true;
    tokenFlag = "$-";
    break;
case 24://append ( case
    finished = true;
    tokenFlag = "$(";
    break;
case 25://append ) case
    finished = true;
    tokenFlag = "$)";
    break;
case 26://append { case
    finished = true;
    tokenFlag = "${";
    break;
case 27://append } case
    finished = true;
    tokenFlag = "$}";
    break;
case 28:
    previousState = stateTable[previousState][col];
    break;
case 29://append != case
    finished = true;
    tokenFlag = "$!=";
    break;
}
if (finished) { //new token condition
    if (tokenFlag == "<var>") { //reserved word check
        if (completedString == "CONST") {
            outFile << completedString + " " + "CONST" << endl;
        }
        else if (completedString == "IF") {
            outFile << completedString + " " + "IF" << endl;
        }
        else if (completedString == "VAR") {
            outFile << completedString + " " + "VAR" << endl;
        }
        else if (completedString == "THEN") {
            outFile << completedString + " " + "THEN" << endl;
        }
        else if (completedString == "ELSE") {
            outFile << completedString + " " + "ELSE" << endl;
        }
        else if (completedString == "PROCEDURE") {
            outFile << completedString + " " + "PROCEDURE" << endl;
        }
        else if (completedString == "WHILE") {
            outFile << completedString + " " + "WHILE" << endl;
        }
        else if (completedString == "CALL") {
            outFile << completedString + " " + "CALL" << endl;
        }
        else if (completedString == "DO") {
            outFile << completedString + " " + "DO" << endl;
        }
```

```
                        else if (completedString == "ODD") {
                            outFile << completedString + " " + "ODD" << endl;
                        }
                        else if (completedString == "CLASS") {
                            outFile << completedString + " " + "CLASS" << endl;
                        }
                        else if (completedString == "PRINTNUMBER") {
                            outFile << completedString + " " + "PRINTNUMBER" << endl;
                        }
                        else if (completedString == "PRINTSTRING") {
                            outFile << completedString + " " + "PRINTSTRING" << endl;
                        }
                        else if (completedString == "READ") {
                            outFile << completedString + " " + "READ" << endl;
                        }
                        else {
                            outFile << completedString + " " + tokenFlag << endl; //if its not a
reserved word, add it as a <var>
                        }
                    }
                    else { //if its not a <var>, append with the current flag
                        outFile << completedString + " " + tokenFlag << endl;
                    }
                    completedString = ""; //reset the string to append, the token flag, set the state
back to 0, and reset finished flag
                    tokenFlag = "";
                    previousState = stateTable[0][col];
                    finished = false;

                }
                if (col != 0) { completedString += token; } //append token unless its a space
            }
        }
        outFile << "EOF $EOF";
        inFile.close();
        outFile.close();
    }
    else { //catch statement
        cout << "File in pass 1 failed to open!" << endl;
    }
}

void ScannerPassTwo(string pass1, string pass2) {
    ifstream inFile(pass1);
    ofstream outFile(pass2);
    if (inFile.is_open() && outFile.is_open()) { // if it's open, do function, else, report failure
        cout << "files in pass 2 successfully open!" << endl;
        string line = "";
        int col = 0;
        int row = 0;
        int counter = 0;
        int address = 0;
        const int max_duplicates = 15;
        int total_duplicates = 0;
        string duplicates[max_duplicates]; //array to store duplicate literals, setting it to 15 for now
        string variablename = "";
        string segment = "";
        string previousType = "";
        string previousSymbol = "";
```

```
int stateTable[18][15] = {
{1, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13},
{13, 2, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13},
{13, 13, 3, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13},
{13, 10, 13, 4, 13, 13, 13, 8, 13, 13, 10, 14, 13, 13, 10},
{13, 5, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13},
{13, 13, 13, 13, 6, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13},
{13, 13, 13, 13, 13, 7, 13, 13, 13, 13, 13, 13, 13, 13, 13},
{13, 13, 13, 13, 13, 13, 3, 13, 4, 13, 13, 13, 13, 13, 13},
{13, 9, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13},
{13, 13, 13, 13, 13, 13, 3, 13, 8, 13, 13, 13, 13, 13, 13},
{10, 10, 10, 10, 10, 11, 10, 10, 10, 12, 10, 14, 10, 10, 10},
{10, 10, 10, 10, 10, 10, 10, 10, 10, 12, 10, 10, 10, 10, 13},
{-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 13, 13, 13},
{-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 13, 13, 13},
{13, 15, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13},
{13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 16, 13, 13},
{13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 17, 13},
{13, 13, 3, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13},
};
int previousState = 0;
while (getline(inFile, line)) {
    bool next = false;
    bool addToken = false;
    string Currentsymbol = "";
    string Currenttype = "";
    for (int i = 0; i < line.size(); i++) {
        char token = line[i];
        if (token == ' ') {
            next = true;
        }
        else {
            if (!next) { //first half before delimiter
                Currentsymbol += token;
            }
            else { //second half
                Currenttype += token;
            }
        }
    }
    if (Currenttype == "CLASS") {
        col = 0;
    }
    else if (Currenttype == "<var>") {
        col = 1;
    }
    else if (Currenttype == "${") {
        col = 2;
    }
    else if (Currenttype == "CONST") {
        col = 3;
    }
    else if (Currenttype == "$=") {
        col = 4;
    }
    else if (Currenttype == "<int>") {
        col = 5;
    }
    else if (Currenttype == "$;") {
```

```
                col = 6;
            }
            else if (Currenttype == "VAR") {
                col = 7;
            }
            else if (Currenttype == "$,") {
                col = 8;
            }
            else if (Currenttype == "EOF") {
                col = 9;
            }
            else if (Currenttype == "PROCEDURE") {
                col = 11;
            }
            else if (Currenttype == "$(") {
                col = 12;
            }
            else if (Currenttype == "$)") {
                col = 13;
            }
            else {
                col = 10;
            }
            switch (previousState) {
            case 0:
                previousState = stateTable[previousState][col];
                break;
            case 1:
                previousState = stateTable[previousState][col];
                variablename = Currentsymbol;
                previousType = "<PROGRAM NAME>";
                addToken = true; // add the program name directly after "class"
                break;
            case 2:
                previousState = stateTable[previousState][col];
                break;
            case 3:
                previousState = stateTable[previousState][col];
                previousType = ""; //state 3 should be resetting types as its a transition between const
and var declarations
                break;
            case 4:
                previousState = stateTable[previousState][col];
                previousType = "ConstVar";
                variablename = Currentsymbol; // get the name for the constant here
                break;
            case 5:
                previousState = stateTable[previousState][col];
                break;
            case 6:
                previousState = stateTable[previousState][col];
                previousSymbol = Currentsymbol; //get the value for the constant here
                break;
            case 7://add constant case
                previousState = stateTable[previousState][col];
                addToken = true;
                break;
            case 8:
                previousState = stateTable[previousState][col];
```

```
    variablename = Currentsymbol;
    previousType = "Var";
    break;
case 9:
    previousState = stateTable[previousState][col]; //add undeclared VAR to table case
    addToken = true;
    break;
case 10:
    previousState = stateTable[previousState][col];
    if (previousState == 11) { //if the next state is 11, add a numeric literal to table
        bool isDuplicate = false;
        for (int i = 0; i < total_duplicates; i++) { // check for duplicate num lits
            if (duplicates[i] == Currentsymbol) {
                isDuplicate = true;
                break;
            }
        }
        if (isDuplicate) {
            if (total_duplicates >= max_duplicates) { //make sure to not go out of bounds
                cout << "MAX DUPLICATES REACHED!" << endl;
            }
        }
        else {
            variablename = Currentsymbol;
            previousSymbol = Currentsymbol;
            previousType = "NumericLiteral";
            addToken = true;
            duplicates[total_duplicates] = Currentsymbol;
            total_duplicates++; //increment total duplicates
        }
    }
    break;
case 11:
    previousState = stateTable[previousState][col];
    break;
case 12:
    break; //eof case
case 13: //error case
    cout << "Error case reached! " << endl;
case 14: //procedure case
    previousState = stateTable[previousState][col];
    variablename = Currentsymbol; //add the name of the procedure to the symbol table
    previousType = "<PROCEDURE NAME>";
    addToken = true;
    break;
case 15:
    previousState = stateTable[previousState][col];
    break;
case 16:
    previousState = stateTable[previousState][col];
    break;
case 17:
    previousState = stateTable[previousState][col];
    break;
}


if (addToken) {
    if (previousType == "<PROGRAM NAME>" || previousType == "<PROCEDURE NAME>") { //if its a
```

```
class or proc name
                    segment = "CS";
                }
                else {
                    segment = "DS";
                }
                if (previousSymbol == "") { //if we never set the symbol
                    previousSymbol = "Null";
                }
                outFile << counter << " " << variablename << " " << previousType << " " <<
previousSymbol << " " << address << " " << segment << endl;
                counter += 1;//row number
                if (segment == "DS") { //increment address only if its a data section
                    address += 2;
                }
                variablename = ""; //reset name and symbol
                previousSymbol = "";
                if (previousState == 3) { //reset type only if we are going back to case 3
                    previousType = "";
                }
            }
        }
        // add x amount of temp variables at the end:
        previousType = "Var";
        previousSymbol = "Null";
        segment = "DS";
        int totaltemps = 3; //can easily change this to utilize I/O if we want user inputed temp
variables
        for (int i = 0; i < totaltemps; i++) {
            variablename = "T" + to_string(i + 1);
            outFile << counter << " " << variablename << " " << previousType << " " << previousSymbol <<
" " << address << " " << segment << endl;
            address += 2;
        }
        inFile.close();
        outFile.close();
    }
    else {
        cout << "Files in pass 2 failed to open!" << endl;
    }


}

class stackElement { // objects to push into stack
public:
    string token = "";
    string elemClass = "";
    int colValue = 0; // we need this value to use the precedence table, we get it from the if
statements below
};

string quadGenerator(string previousOperator, stackElement currentOp, stack<stackElement>& myStack,
stack<stackElement>& tempStack,
    stack<stackElement>& operatorStack, ofstream& outFile, stack<stackElement>& endStack,
stack<stackElement>& startStack, int& totalJumps,
int totalTemps) {

    string quadReturn = "";
```

```
    stackElement tempStackElement;
    if (previousOperator == "=") {
        stackElement rightValue = myStack.top();
        myStack.pop();
        stackElement leftValue = myStack.top();
        myStack.pop();
        if (leftValue.elemClass == "<int>") { //if they're numlits set them to numlits
            leftValue.token = "lit" + leftValue.token;
        }
        if (rightValue.elemClass == "<int>") {
            rightValue.token = "lit" + rightValue.token;
        }
        quadReturn = previousOperator + " " + leftValue.token + " " + rightValue.token + " " + "-";
        for (int i = 1; i <= totalTemps; ++i) { //check if the right is a temp value, if it is, we can
reuse it
            string tempToken = "T" + to_string(i);
            if (rightValue.token == tempToken) {
                tempStackElement.token = tempToken;
                tempStackElement.colValue = 20;
                tempStackElement.elemClass = "Temp Value";
                tempStack.push(tempStackElement);
            }
        }
    }
    if (previousOperator == "+" || previousOperator == "-" || previousOperator == "*" ||
previousOperator == "/") {
        stackElement rightValue = myStack.top();
        myStack.pop();
        stackElement leftValue = myStack.top();
        myStack.pop();
        string currentTemp = tempStack.top().token;
        if (leftValue.elemClass == "<int>") { //if they're numlits set them to numlits
            leftValue.token = "lit" + leftValue.token;
        }
        if (rightValue.elemClass == "<int>") {
            rightValue.token = "lit" + rightValue.token;
        }
        quadReturn = previousOperator + " " + leftValue.token + " " + rightValue.token + " " +
currentTemp;
        tempStack.pop();
        for (int i = 1; i <= totalTemps; ++i) { //check if the left or right are temp values, if they
are, we can reuse them
            string tempToken = "T" + std::to_string(i);
            if (rightValue.token == tempToken || leftValue.token == tempToken) {
                tempStackElement.token = tempToken;
                tempStackElement.colValue = 20;
                tempStackElement.elemClass = "Temp Value";
                tempStack.push(tempStackElement);
            }
        }
        rightValue.token = currentTemp;  // reuse a token to make it the new tempVariable
        rightValue.elemClass = "<var>";
        myStack.push(rightValue); //push temp variable into the stack
    }
    if (previousOperator == "==" || previousOperator == "!=" || previousOperator == ">" ||
previousOperator == "<" || previousOperator == ">=" || previousOperator == "<=") {
        stackElement rightValue = myStack.top();
        myStack.pop();
        stackElement leftValue = myStack.top();
```

```
        myStack.pop();
        if (leftValue.elemClass == "<int>") { //if they're num lits, set them to numlits
            leftValue.token = "lit" + leftValue.token;
        }
        if (rightValue.elemClass == "<int>") {
            rightValue.token = "lit" + rightValue.token;
        }
        quadReturn = previousOperator + " " + leftValue.token + " " + rightValue.token + " " + "-";
        outFile << quadReturn << +"\n";
        if (previousOperator == "==") { //, rel operation complete, now for jumps, as you wouldnt use
rel ops without with if or while
            quadReturn = currentOp.token + " L" + to_string(totalJumps) + " JNE -";
        }
        if (previousOperator == "!=") { //, rel operation complete, now for jumps, as you wouldnt use
rel ops without with if or while
            quadReturn = currentOp.token + " L" + to_string(totalJumps) + " JE -";
        }
        if (previousOperator == ">") { //, rel operation complete, now for jumps, as you wouldnt use rel
ops without with if or while
            quadReturn = currentOp.token + " L" + to_string(totalJumps) + " JLE -";
        }
        if (previousOperator == "<") { //, rel operation complete, now for jumps, as you wouldnt use rel
ops without with if or while
            quadReturn = currentOp.token + " L" + to_string(totalJumps) + " JGE -";
        }
        if (previousOperator == ">=") { //, rel operation complete, now for jumps, as you wouldnt use
rel ops without with if or while
            quadReturn = currentOp.token + " L" + to_string(totalJumps) + " JL -";
        }
        if (previousOperator == "<=") { //, rel operation complete, now for jumps, as you wouldnt use
rel ops without with if or while
            quadReturn = currentOp.token + " L" + to_string(totalJumps) + " JG -";
        }
    }
    if (previousOperator == "(" || previousOperator == "{") { //if its matching brackets/parens,
recognize and pop stack
        cout << "matching parenthesis detected! popping stack!" << endl; //this will pop stack in the
syntax analyzer
    }
    if (previousOperator == "DO") {
        stackElement whileLabel = startStack.top();
        startStack.pop();
        stackElement jumpLabel = endStack.top();
        endStack.pop();
        outFile << whileLabel.token + " JMP - -" << endl;
        quadReturn = jumpLabel.token + " NOP - - ";
        operatorStack.pop(); // we can pop the DO and the WHILE now
        totalJumps--; // now that we've finished a jump, we can decrement the jump to match the labels
    }
    if (previousOperator == "THEN") {
        quadReturn = startStack.top().token + " NOP - -"; //print label quad
        startStack.pop();
        operatorStack.pop(); //pop IF THEN
    }
    if (previousOperator == "ELSE") {
        quadReturn = endStack.top().token + " NOP - -"; //print label quad
        endStack.pop();
        operatorStack.pop(); //pop IF THEN ELSE
        operatorStack.pop();
```

```
    }
    if (previousOperator == "READ") {
        stackElement variable = myStack.top();
        myStack.pop();
        quadReturn = "READ " + variable.token + " - -";
    }
    if (previousOperator == "PRINTNUMBER") {
        stackElement variable = myStack.top();
        if (variable.elemClass == "<int>") { //if its a numlit, set it to numlit
            variable.token = "lit" + variable.token;
        }
        myStack.pop();
        quadReturn = "PRINTNUMBER " + variable.token + " - -";
    }
    if (previousOperator == "PRINTSTRING") {
        stackElement variable = myStack.top();
        myStack.pop();
        quadReturn = "PRINTSTRING " + variable.token + " - -";
    }
    return quadReturn;
}


void syntaxAnalyzer(string pass1, string pass2, string syntaxPass) {
    ifstream inFile1(pass1); //token list
    ifstream inFile2(pass2); //symbol table
    ofstream outFile(syntaxPass); //output
    if (inFile1.is_open() && inFile2.is_open() && outFile.is_open()) { // if it's open, do function,
else, report failure
        cout << "files in syntax successfully open!" << endl;
        stack<stackElement> myStack; // see stack logic above, max size is set to 100
        string precedence = "";
        string line = "";
        int col = 0;
        int row = 0;
        int tempFlag = 0;
        string quad = "";
        stackElement op;
        op.token = "";
        stack<stackElement> previousOperator; //operator stack
        stack<stackElement> tempStack; //stack to hold temp values for efficent use
        stack<stackElement> startStack; //start stack for while loops
        stack<stackElement> endStack; //end stack (fixup stack)
        int totalJumps = 0;
        int totalTemps = 3;
        for (int i = totalTemps; i > 0; i--) { //add however many total temps to the tempStack
            stackElement tempStackEl;
            tempStackEl.token = "T" + to_string(i);
            tempStackEl.colValue = 20;
            tempStackEl.elemClass = "Temp Value";
            tempStack.push(tempStackEl);
        }

        string precedenceTable[28][28] = {
        {" ", "<", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", "<", " ", " ", " ", "<", "<", "<", " "},
        {">", " ", "<", "<", "<", " ", "<", "<", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", "<", " ", " ", ">", " ", " ", " ", " ", " "},
        {">", " ", ">", ">", "<", ">", "<", "<", " ", ">", " ", ">", ">", ">", ">", ">", ">", " ", " ",
```

```
" ", "<", " ", " ", ">", " ", " ", " ", " "},
        {">", " ", ">", ">", "<", ">", "<", "<", " ", ">", " ", ">", ">", ">", ">", ">", ">", " ", " ",
" ", "<", " ", " ", ">", " ", " ", " ", " "},
        {">", " ", "<", "<", "<", "=", "<", "<", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", "<", " ", " ", ">", " ", " ", " ", " "},
        {">", " ", ">", ">", " ", ">", ">", ">", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", "<", " ", " ", ">", " ", " ", " ", " "},
        {">", " ", ">", ">", "<", ">", ">", ">", " ", ">", " ", ">", ">", ">", ">", ">", ">", " ", " ",
" ", "<", " ", " ", ">", " ", " ", " ", " "},
        {">", " ", ">", ">", "<", ">", ">", ">", " ", ">", " ", ">", ">", ">", ">", ">", ">", " ", " ",
" ", "<", " ", " ", ">", " ", " ", " ", " "},
        {" ", " ", "<", "<", "<", " ", "<", "<", " ", "=", "<", "<", "<", "<", "<", "<", "<", " ", " ",
" ", "<", " ", " ", ">", " ", " ", " ", " "},
        {" ", "<", " ", " ", " ", " ", " ", " ", "<", " ", " ", " ", " ", " ", " ", " ", " ", "<", " ",
"<", "<", " ", " ", ">", " ", " ", " ", "<"},
        {" ", " ", "<", "<", "<", " ", "<", "<", " ", ">", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", "<", " ", ">", ">", " ", " ", " ", " "},
        {" ", " ", "<", "<", "<", " ", "<", "<", " ", ">", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", "<", " ", ">", ">", " ", " ", " ", " "},
        {" ", " ", "<", "<", "<", " ", "<", "<", " ", ">", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", "<", " ", ">", ">", " ", " ", " ", " "},
        {" ", " ", "<", "<", "<", " ", "<", "<", " ", ">", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", "<", " ", ">", ">", " ", " ", " ", " "},
        {" ", " ", "<", "<", "<", " ", "<", "<", " ", ">", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", "<", " ", ">", ">", " ", " ", " ", " "},
        {" ", " ", "<", "<", "<", " ", "<", "<", " ", ">", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", "<", " ", ">", ">", " ", " ", " ", " "},
        {" ", " ", "<", "<", "<", " ", "<", "<", " ", ">", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", "<", " ", ">", ">", " ", " ", " ", " "},
        {" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", ">",
" ", "<", " ", " ", ">", " ", " ", " ", " "},
        {">", " ", ">", ">", " ", ">", ">", ">", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", "<", " ", " ", ">", " ", " ", " ", " "},
        {" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", "<", " ", " ", ">", " ", " ", " ", " "},
        {">", ">", ">", ">", ">", ">", ">", ">", ">", ">", ">", ">", ">", ">", ">", ">", ">", ">", ">",
">", " ", ">", ">", ">", " ", " ", " ", ">"},
        {" ", " ", "<", "<", " ", " ", "<", "<", " ", " ", " ", "<", "<", "<", "<", "<", "<", " ", ">",
" ", "<", " ", "=", ">", " ", " ", " ", " "},
        {" ", "<", "<", "<", " ", " ", "<", "<", "<", " ", " ", " ", " ", " ", " ", " ", " ", "<", ">",
" ", "<", "<", " ", ">", " ", " ", " ", "<"},
        {" ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", " ", " ", " ", ">", " ", " ", " ", " "},
        {">", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", " ", " ", " ", ">", " ", " ", " ", " "},
        {">", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", " ", " ", " ", ">", " ", " ", " ", " "},
        {">", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
" ", " ", " ", " ", ">", " ", " ", " ", " "},
        {" ", " ", "<", "<", "<", " ", "<", "<", " ", "=", "<", "<", "<", "<", "<", "<", "<", " ", ">",
" ", "<", " ", " ", ">", " ", " ", " ", " "},
        };
        int previousState = 0;
        while (getline(inFile1, line)) {
            stackElement curr;
            bool next = false;
            bool addToken = false;
            string currentToken = "";
            string Currenttype = "";
```

```
for (int i = 0; i < line.size(); i++) {
    char token = line[i];
    if (token == ' ') {
        next = true;
    }
    else {
        if (!next) { //first half before delimiter
            currentToken += token;
        }
        else { //second half
            Currenttype += token;
        }
    }
}
if (Currenttype == "CONST" || Currenttype == "VAR") { //skip through declarations section
    while (getline(inFile1, line)) {
        currentToken = "";
        Currenttype = "";
        next = false;
        for (int i = 0; i < line.size(); i++) {
            char token = line[i];
            if (token == ' ') {
                next = true;
            }
            else {
                if (!next) {
                    currentToken += token;
                }
                else {
                    Currenttype += token;
                }
            }
        }
        // stop once we're at a semicolon or the eof
        if (Currenttype == "$;" || Currenttype == "$EOF") {
            break;
        }
    }
}
if (Currenttype == "$;") {
    col = 0;
    op.colValue = 0;
    op.elemClass = "$;";
    op.token = ";";
}
else if (Currenttype == "$=") {
    col = 1;
    op.colValue = 1;
    op.elemClass = "$=";
    op.token = "=";
}
else if (Currenttype == "$+") {
    col = 2;
    op.colValue = 2;
    op.elemClass = "$+";
    op.token = "+";
}
else if (Currenttype == "$-") {
    col = 3;
```

```
    op.colValue = 3;
    op.elemClass = "$-";
    op.token = "-";
}
else if (Currenttype == "$(") {
    col = 4;
    op.colValue = 4;
    op.elemClass = "$(";
    op.token = "(";
}
else if (Currenttype == "$)") {
    col = 5;
    op.colValue = 5;
    op.elemClass = "$)";
    op.token = ")";
}
else if (Currenttype == "$*") {
    col = 6;
    op.colValue = 6;
    op.elemClass = "$*";
    op.token = "*";
}
else if (Currenttype == "$/") {
    col = 7;
    op.colValue = 7;
    op.elemClass = "$/";
    op.token = "/";
}
else if (Currenttype == "IF") {
    totalJumps++; //increase total jumps
    col = 8;
    op.colValue = 8;
    op.elemClass = "IF";
    op.token = "L" + to_string(totalJumps); //add a label to the fixup stack
    startStack.push(op);
    outFile << "IF - - -" << "\n"; //set up quad immediately
    op.token = "IF";
}
else if (Currenttype == "THEN") {
    col = 9;
    op.colValue = 9;
    op.elemClass = "THEN";
    op.token = "THEN";
}
else if (Currenttype == "ODD") {
    col = 10;
    op.colValue = 10;
    op.elemClass = "ODD";
    op.token = "ODD";
}
else if (Currenttype == "$==") {
    col = 11;
    op.colValue = 11;
    op.elemClass = "$==";
    op.token = "==";
}
else if (Currenttype == "$!=") {
    col = 12;
    op.colValue = 12;
```

```
    op.elemClass = "$!=";
    op.token = "!=";
}
else if (Currenttype == "$>") {
    col = 13;
    op.colValue = 13;
    op.elemClass = "$>";
    op.token = ">";
}
else if (Currenttype == "$<") {
    col = 14;
    op.colValue = 14;
    op.elemClass = "$<";
    op.token = "<";
}
else if (Currenttype == "$>=") {
    col = 15;
    op.colValue = 15;
    op.elemClass = "$>=";
    op.token = ">=";
}
else if (Currenttype == "$<=") {
    col = 16;
    op.colValue = 16;
    op.elemClass = "$<=";
    op.token = "<=";
}
else if (Currenttype == "${") {
    col = 17;
    op.colValue = 17;
    op.elemClass = "${";
    op.token = "{";
}
else if (Currenttype == "$}") {
    col = 18;
    op.colValue = 18;
    op.elemClass = "$}";
    op.token = "}";
}
else if (Currenttype == "CALL") {
    col = 19;
    op.colValue = 19;
    op.elemClass = "CALL";
    op.token = "CALL";
}
else if (Currenttype == "$,") { //skip commas
    continue;
}
else if (Currenttype == "WHILE") {
    totalJumps++; //increase total jumps
    col = 21;
    op.colValue = 21;
    op.elemClass = "WHILE";
    op.token = "W" + to_string(totalJumps); //add a while to the startstack
    startStack.push(op);
    outFile << "WHILE " << op.token << " - -" << "\n"; //set up quad immediately
    op.token = "WHILE";
}
else if (Currenttype == "DO") {
```

```
            col = 22;
            op.colValue = 22;
            op.elemClass = "DO";
            op.token = "L" + to_string(totalJumps);
            endStack.push(op);
            op.token = "DO";
        }
        else if (Currenttype == "$EOF") {
            col = 23;
            op.colValue = 23;
            op.elemClass = "$EOF";
            op.token = "EOF";
        }
        else if (Currenttype == "READ") {
            col = 24;
            op.colValue = 24;
            op.elemClass = "READ";
            op.token = "READ";
        }
        else if (Currenttype == "PRINTNUMBER") {
            col = 25;
            op.colValue = 25;
            op.elemClass = "PRINTNUMBER";
            op.token = "PRINTNUMBER";
        }
        else if (Currenttype == "PRINTSTRING") {
            col = 26;
            op.colValue = 26;
            op.elemClass = "PRINTSTRING";
            op.token = "PRINTSTRING";
        }
        else if (Currenttype == "ELSE") {
            totalJumps++; // increment jumps
            col = 27;
            op.colValue = 27;
            op.elemClass = "ELSE";
            op.token = "L" + to_string(totalJumps); // generate new jump to end, push into end stack
            outFile << op.token + " JMP - -" << endl;
            endStack.push(op);
            outFile << startStack.top().token + " NOP - -" << endl; // pop fixup stack and set
location
            startStack.pop();
            op.token = "ELSE"; // now push else into op stack
        }
        else { //assume its not an operator
            curr.colValue = 20; // set it to VAR and push it in our variable stack
            curr.token = currentToken;
            curr.elemClass = Currenttype;
            myStack.push(curr);
            continue;//we don't need to change row value because we're never comparing nonterminals
to anything
        }
        curr.colValue = col;
        curr.token = currentToken;
        curr.elemClass = Currenttype;
        if (previousOperator.empty()) { //if operator stack is empty, push token and move on
            previousOperator.push(op);
            row = op.colValue;
```

```
            }
            else {
                precedence = precedenceTable[row][op.colValue]; //generate current precedence
                while (precedence == ">" && !previousOperator.empty()) {  //found the tail, loop
                    if (!previousOperator.empty()) {
                        quad = quadGenerator(previousOperator.top().token, op, myStack, tempStack,
previousOperator, outFile, endStack, startStack, totalJumps, totalTemps);
                        previousOperator.pop();
                        if (!previousOperator.empty()) { //edge case at the end of the file
                            row = previousOperator.top().colValue; //reassign row value based on whats
on top of the stack afterwards
                        }
                        if (quad != "") { // if the quad generator returned a valid quad
                            outFile << quad + "\n";
                        }
                        precedence = precedenceTable[row][col];
                    }
                }
                if (op.colValue != 0 && op.colValue != 18) { //because ; will never match in our PDA, we
don't put them in the op stack
                    previousOperator.push(op);
                }
                if (!previousOperator.empty()) { //edge case at the end of the file
                    row = previousOperator.top().colValue; //reassign row value based on whats on top of
the stack afterwards
                }
                else {
                    row = 23; // if it is empty, you're likely at the end of the file at this point.
setting to 0 and reporting end of file.
                    cout << "Program fully recognized in Syntax Analyzer!" << endl;
                }
            }
        }
        inFile1.close();
        inFile2.close();
        outFile.close();
    }
    else {
        cout << "Files in syntax failed to open!" << endl;
    }
}

class quad { // objects to store quads
public:
    string posOne = "";
    string posTwo = "";
    string posThree = "";
    string posFour = "";

    void posTracker(int position, string tokenString) {
        if (position == 0) { // i needed a way to fill up each position in the quad... there is likely a
better way to do this
            posOne = tokenString;
        }
        if (position == 1) {
            posTwo = tokenString;
        }
        if (position == 2) {
            posThree = tokenString;
```

```
        }
        if (position == 3) {
            posFour = tokenString;
        }
    }
};



void semanticsAnalyzer(string quads, string semanticOutput) {

    ifstream inFile(quads);
    ofstream outFile(semanticOutput);
    if (inFile.is_open() && outFile.is_open()) { // if it's open, do function, else, report failure
        cout << "in semantics! files successfully opened!" << endl;
        string line;
        while (getline(inFile, line)) { // read line by line
            int totalSpaces = 0;
            string tempString = "";
            quad currQuad;
            for (int i = 0; i < line.size(); i++) {
                char token = line[i];
                if (token == ' ') {
                    currQuad.posTracker(totalSpaces, tempString);
                    totalSpaces++;
                    tempString = "";
                }
                else {
                    tempString += token;
                }
            }
            currQuad.posTracker(totalSpaces, tempString); // add the last section after for loop ends at
the end of the line

            if (currQuad.posOne == "=") {
                outFile << "mov ax, [" << currQuad.posThree << "]" << endl;
                outFile << "mov [" << currQuad.posTwo << "], ax" << endl;
            }
            else if (currQuad.posOne == "+") {
                outFile << "mov ax,[" << currQuad.posTwo << "]" << endl;
                outFile << "add ax,[" << currQuad.posThree << "]" << endl;
                outFile << "mov [" << currQuad.posFour << "], ax" << endl;
            }
            else if (currQuad.posOne == "-") {
                outFile << "mov ax,[" << currQuad.posTwo << "]" << endl;
                outFile << "sub ax,[" << currQuad.posThree << "]" << endl;
                outFile << "mov [" << currQuad.posFour << "], ax" << endl;
            }
            else if (currQuad.posOne == "/") {
                outFile << "mov ax,[" << currQuad.posTwo << "]" << endl;
                outFile << "mov bx,[" << currQuad.posThree << "]" << endl;
                outFile << "div bx" << endl;
                outFile << "mov [" << currQuad.posFour << "],ax" << endl;
            }
            else if (currQuad.posOne == "*") {
                outFile << "mov ax,[" << currQuad.posTwo << "]" << endl;
                outFile << "mul word [" << currQuad.posThree << "]" << endl;
                outFile << "mov [" << currQuad.posFour << "], ax" << endl;
```

```
            }
            else if (currQuad.posOne == "WHILE") {
                outFile << currQuad.posTwo << " NOP" << endl;
            }
            else if (currQuad.posOne == "DO") {
                outFile << currQuad.posThree << " " << currQuad.posTwo << endl;
            }
            else if (currQuad.posOne == "IF") { //dont need to do anything with the if statements
                continue;
            }
            else if (currQuad.posOne == "THEN") {
                outFile << currQuad.posThree << " " << currQuad.posTwo << endl;
            }
            else if (currQuad.posOne[0] == 'W') {
                outFile << "JMP " << currQuad.posOne << endl;
            }
            else if (currQuad.posOne[0] == 'L') {
                if (currQuad.posTwo == "JMP") {
                    outFile << currQuad.posTwo << " " << currQuad.posOne << endl;
                }
                else {
                    outFile << currQuad.posOne << " NOP" << endl;
                }
            }
            else if (currQuad.posOne == "==" || currQuad.posOne == "!=" || currQuad.posOne == ">" ||
currQuad.posOne == "<" ||
                currQuad.posOne == ">=" || currQuad.posOne == "<=") {
                outFile << "mov ax,[" << currQuad.posTwo << "]" << endl;
                outFile << "cmp ax,[" << currQuad.posThree << "]" << endl;
            }
            else if (currQuad.posOne == "READ") {
                outFile << "call PrintString" << endl;
                outFile << "call GetAnInteger" << endl;
                outFile << "mov ax, [ReadInt]" << endl; //returns read int
                outFile << "mov [" << currQuad.posTwo << "], ax" << endl; //stores read int in variable
            }
            else if (currQuad.posOne == "PRINTNUMBER") {
                outFile << "mov ax, [" << currQuad.posTwo << "]" << endl;
                outFile << "call ConvertIntegerToString" << endl;
                outFile << "mov eax, 4" << endl;
                outFile << "mov ebx, 1" << endl;
                outFile << "mov ecx, Result" << endl;
                outFile << "mov edx, ResultEnd" << endl;
                outFile << "int 80h" << endl;
            }
            else if (currQuad.posOne == "PRINTSTRING") { // i want to implement strings in my language,
so leaving this code here in case i get around to it
                outFile << "mov ax, [" << currQuad.posTwo << "]" << endl;
                outFile << "mov eax, 4" << endl;
                outFile << "mov ebx, 1" << endl;
                outFile << "mov ecx, Result" << endl;
                outFile << "mov edx, ResultEnd" << endl;
                outFile << "int 80h" << endl;
            }
            else {
                cout << "ERROR! QUAD NOT RECOGNIZED!" << endl; //error case
            }
        }
        cout << "Semantics finished!" << endl;
```

```
        inFile.close();
        outFile.close();
    }
    else {
        cout << "files in semantics failed to open!" << endl;
    }
}

class symbolTableLine { // objects to store lines of symbol table
public:
    string posOne = "";
    string posTwo = "";
    string posThree = "";
    string posFour = "";
    string posFive = "";
    string posSix = "";

    void posTracker(int position, string tokenString) {
        if (position == 0) { // i needed a way to fill up each position in the ST.. there is likely a
better way to do this
            posOne = tokenString;
        }
        if (position == 1) {
            posTwo = tokenString;
        }
        if (position == 2) {
            posThree = tokenString;
        }
        if (position == 3) {
            posFour = tokenString;
        }
        if (position == 4) {
            posFive = tokenString;
        }
        if (position == 5) {
            posSix = tokenString;
        }
    }
};

void finalAssembly(string symbolTableString, string assemblyText1, string assemblyText2,
    string assemblyText3, string assemblyText4, string myAssemblyCode, string finalProgram) {

    ifstream symbolTable(symbolTableString);
    ifstream dotDataSection(assemblyText1);
    ifstream dotBSSSection(assemblyText2);
    ifstream startSection(assemblyText3);
    ifstream assemblyPartFour(assemblyText4);
    ifstream myAssemblyFile(myAssemblyCode);
    ofstream finalProgramFile(finalProgram);

    if (finalProgramFile.is_open() && symbolTable.is_open() && dotDataSection.is_open() &&
        dotBSSSection.is_open() && startSection.is_open() && assemblyPartFour.is_open() &&
        myAssemblyFile.is_open()) {
        cout << "In Final Assembly! All files successfully opened!" << endl;
        string line;

        while (getline(dotDataSection, line)) { //write burris section 1 to final program
            finalProgramFile << line << endl;
```

```
        }

        while (getline(symbolTable, line)) { // add constants and numeric lits to .data file
            int totalSpaces = 0;
            string tempString = "";
            symbolTableLine currLine;
            for (int i = 0; i < line.size(); i++) {
                char token = line[i];
                if (token == ' ') {
                    currLine.posTracker(totalSpaces, tempString);
                    totalSpaces++;
                    tempString = "";
                }
                else {
                    tempString += token;
                }
            }
            currLine.posTracker(totalSpaces, tempString); // add the last section after for loop ends at
the end of the line
            if (currLine.posThree == "NumericLiteral") { //add constants to the .data section
                finalProgramFile << "lit" + currLine.posFour << " DW " << currLine.posFour << endl;
            }
            else if (currLine.posThree == "ConstVar") {
                finalProgramFile << currLine.posTwo << " DW " << currLine.posFour << endl;
            }
            else {
                continue;
            }
        }
        symbolTable.close();
        while (getline(dotBSSSection, line)) { // add burris .CSS section
            finalProgramFile << line << endl;
        }

        ifstream symbolTable2(symbolTableString); //new file to reset symbol table
        while (getline(symbolTable2, line)) { // add uninitialized variables here
            int totalSpaces = 0;
            string tempString = "";
            symbolTableLine currLine;
            for (int i = 0; i < line.size(); i++) {
                char token = line[i];
                if (token == ' ') {
                    currLine.posTracker(totalSpaces, tempString);
                    totalSpaces++;
                    tempString = "";
                }
                else {
                    tempString += token;
                }
            }

            currLine.posTracker(totalSpaces, tempString); // add the last section after for loop ends at
the end of the line
            if (currLine.posThree == "Var") { //add constants to the .data section
                finalProgramFile << currLine.posTwo << " RESW 1" << endl;
            }
            else {
                continue;
            }
```

```
        }

        while (getline(startSection, line)) { //add the _start section before main program
            finalProgramFile << line << endl;
        }

        while (getline(myAssemblyFile, line)) { //add our compiled assembly code
            finalProgramFile << line << endl;
        }
        while (getline(assemblyPartFour, line)) {
            finalProgramFile << line << endl;
        }

    }
    else {
        cout << "Final Assmebly files failed to open!" << endl;
    }
    finalProgramFile.close();
    symbolTable.close();
    dotDataSection.close();
    dotBSSSection.close();
    startSection.close();
    assemblyPartFour.close();
    myAssemblyFile.close();
}

int main(int argc, char* argv[]) { // arg[0] is program file, arg[1] is the config file
    ifstream configFile(argv[1]); // i/o redirection file
    if (!configFile) { //config file that contains all file locations for compiler
        cout << "Error opening configuration file!" << endl;
        return 0;
    }
    string passOneFile, passTwoFile, syntaxOutputFile, semanticsOutputFile, java0SourceCode;
    string assemblyText1, assemblyText2, assemblyText3, assemblyText4, finalProgram;
    cout << "input pass 1 file location: " << endl;
    getline(configFile, passOneFile);
    cout << "input pass 2 file location: " << endl;
    getline(configFile, passTwoFile);
    cout << "input syntaxOuput file location: " << endl;
    getline(configFile, syntaxOutputFile);
    cout << "input semanticOutput file location: " << endl;
    getline(configFile, semanticsOutputFile);
    cout << "input java0 file to interpret: " << endl;
    getline(configFile, java0SourceCode);
    cout << "input assemblyText1 file location: " << endl;
    getline(configFile, assemblyText1);
    cout << "input assemblyText2 file location: " << endl;
    getline(configFile, assemblyText2);
    cout << "input assemblyText3 file location: " << endl;
    getline(configFile, assemblyText3);
    cout << "input assemblyText4 file location: " << endl;
    getline(configFile, assemblyText4);
    cout << "input finalProgram.asm file location: " << endl;
    getline(configFile, finalProgram);
    configFile.close();
    ScannerPassOne(java0SourceCode, passOneFile);
    ScannerPassTwo(passOneFile, passTwoFile);
    syntaxAnalyzer(passOneFile, passTwoFile, syntaxOutputFile);
    semanticsAnalyzer(syntaxOutputFile, semanticsOutputFile);
```

```
    finalAssembly(passTwoFile, assemblyText1, assemblyText2, assemblyText3, assemblyText4,
        semanticsOutputFile, finalProgram);

    cout << "scanner passes complete! " << endl;
    return 0;
}
```

Example Programs

The program was run through IO through the command line using a config file that held the locations of all the necessary text files. It was compiled using these commands:

```
cd C:\Users\jonat\source\repos\compilers\CompilerBOption\B-option
g++ -o BurrisCompiler B-option.cpp
BurrisCompiler C:\Users\jonat\Documents\data\config.txt
```

The contents of the config file are stored locally on my machine, and are used to store the outputs of my program. This is its content:

```
C:\Users\jonat\Documents\data\pass1.txt
C:\Users\jonat\Documents\data\pass2.txt
C:\Users\jonat\Documents\data\syntaxOutput.txt
C:\Users\jonat\Documents\data\semanticsOutput.txt
C:\Users\jonat\Documents\data\java0.txt
C:\Users\jonat\Documents\data\assemblyText1.txt
C:\Users\jonat\Documents\data\assemblyText2.txt
C:\Users\jonat\Documents\data\assemblyText3.txt
C:\Users\jonat\Documents\data\assemblyText4.txt
C:\Users\jonat\Documents\data\finalProgram.asm
```

All Assembly was transferred from my Windows 10 Machine to a Linux VPS using “Secure Copy Protocall (SCP)” using this command:

```
scp C:\Users\jonat\Documents\data\finalProgram.asm
developer@144.202.65.55:/home/developer/compilers/
```

and compiled using the following commands:

```
nasm -f elf64 finalProgram.asm -o finalProgram.o
ld -o finalProgram finalProgram.o
./finalProgram
```

Question One

Java0 Source Code:

```
CLASS PGM1{
PROCEDURE questionOne(){
CONST c = 3;
VAR a, Bob, Jane, b, ans;
READ a;
READ b;
READ Bob;
READ Jane;
ans = a * ((Bob + Jane - 10) / 2 * 4) / (b + c);
PRINTNUMBER ans;

/* this program demonstrates order of operations and IO

You can input any numbers, but I will say the numbers in a comment to show my translator's calculations
are correct and to save you time:

If A = 5, B = 2, C = 3, Bob = 30 and Jane = 20, then:
->5 * (( 30 + 20 - 10) / 2 * 4) / ( 2 + 3)
-> 5 * ((40 / 2 * 4) / (5)
-> 5 * 80 / 5
-> Answer is 80.

*/
}
}
```

Pass One Token List:

```
CLASS CLASS
PGM1 <var>
{ ${
PROCEDURE PROCEDURE
questionOne <var>
( $(
) $)
{ ${
CONST CONST
c <var>
= $=
3 <int>
; $;
VAR VAR
a <var>
```

, $,
Bob <var>
, $,
Jane <var>
, $,
b <var>
, $,
ans <var>
; $;
READ READ
a <var>
; $;
READ READ
b <var>
; $;
READ READ
Bob <var>
; $;
READ READ
Jane <var>
; $;
ans <var>
= $=
a <var>
* $*
($(
($(
Bob <var>
+ $+
Jane <var>
- $-
10 <int>
) $)
/ $/
2 <int>
* $*
4 <int>
) $)
/ $/
($(
b <var>
+ $+
c <var>
) $)
; $;
PRINTNUMBER PRINTNUMBER
ans <var>
; $;
} $}

```
} $}
EOF $EOF
```

Pass Two Symbol Table

```
0 PGM1 <PROGRAM NAME> Null 0 CS
1 questionOne <PROCEDURE NAME> Null 0 CS
2 c ConstVar 3 0 DS
3 a Var Null 2 DS
4 Bob Var Null 4 DS
5 Jane Var Null 6 DS
6 b Var Null 8 DS
7 ans Var Null 10 DS
8 10 NumericLiteral 10 12 DS
9 2 NumericLiteral 2 14 DS
10 4 NumericLiteral 4 16 DS
11 T1 Var Null 18 DS
11 T2 Var Null 20 DS
11 T3 Var Null 22 DS
```

Syntax Analyzer Quad Output:

```
READ a - -
READ b - -
READ Bob - -
READ Jane - -
+ Bob Jane T1
- T1 lit10 T2
/ T2 lit2 T1
* T1 lit4 T2
+ b c T1
/ T2 T1 T3
* a T3 T2
= ans T2 -
PRINTNUMBER ans - -
```

Semantics Assembly Output:

```
call PrintString
call GetAnInteger
mov ax, [ReadInt]
mov [a], ax
call PrintString
call GetAnInteger
mov ax, [ReadInt]
mov [b], ax
call PrintString
call GetAnInteger
```

```
mov ax, [ReadInt]
mov [Bob], ax
call PrintString
call GetAnInteger
mov ax, [ReadInt]
mov [Jane], ax
mov ax,[Bob]
add ax,[Jane]
mov [T1], ax
mov ax,[T1]
sub ax,[lit10]
mov [T2], ax
mov ax,[T2]
mov bx,[lit2]
div bx
mov [T1],ax
mov ax,[T1]
mul word [lit4]
mov [T2], ax
mov ax,[b]
add ax,[c]
mov [T1], ax
mov ax,[T2]
mov bx,[T1]
div bx
mov [T3],ax
mov ax,[a]
mul word [T3]
mov [T2], ax
mov ax, [T2]
mov [ans], ax
mov ax, [ans]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
```

Final .ASM file:

```
sys_exit        equ     1
sys_read        equ     3
sys_write       equ     4
stdin           equ     0 ; default keyboard
stdout          equ     1 ; default terminal screen
stderr          equ     3
```

```
section .data                ;used to declare constants
        userMsg                 db      'Enter an integer(less than 32,765): '
        lenUserMsg      equ     $-userMsg
        displayMsg      db      'You entered: '
        lenDisplayMsg   equ     $-displayMsg
        newline         db      0xA      ; 0xA 0xD is ASCII <LF><CR>

        Ten             DW      10  ;Used in converting to base ten.

        printTempchar   db      'Tempchar = : '
        lenprintTempchar        equ     $-printTempchar


        Result          db      'Ans = '
        ResultValue     db      'aaaaa'
                        db      0xA
        ResultEnd       equ     $-Result    ; $=> here, subtract address Result

        num             times 6 db 'ABCDEF'
        numEnd                  equ     $-num
c DW 3
lit10 DW 10
lit2 DW 2
lit4 DW 4
; Start of user variable area    ---------------------------------------------

section  .bss                ;used to declare uninitialized variables

        TempChar        RESB    1           ;temp space for use by GetNextChar
        testchar        RESB    1
        ReadInt         RESW    1           ;Temporary storage GetAnInteger.
        tempint         RESW    1           ;Used in converting to base ten.
        negflag         RESB    1           ;P=positive, N=negative
a RESW 1
Bob RESW 1
Jane RESW 1
b RESW 1
ans RESW 1
T1 RESW 1
T2 RESW 1
T3 RESW 1
        global _start
section .text

_start:  nop
        ; prompt user for positive number

Again:
call PrintString
```

```
call GetAnInteger
mov ax, [ReadInt]
mov [a], ax
call PrintString
call GetAnInteger
mov ax, [ReadInt]
mov [b], ax
call PrintString
call GetAnInteger
mov ax, [ReadInt]
mov [Bob], ax
call PrintString
call GetAnInteger
mov ax, [ReadInt]
mov [Jane], ax
mov ax,[Bob]
add ax,[Jane]
mov [T1], ax
mov ax,[T1]
sub ax,[lit10]
mov [T2], ax
mov ax,[T2]
mov bx,[lit2]
div bx
mov [T1],ax
mov ax,[T1]
mul word [lit4]
mov [T2], ax
mov ax,[b]
add ax,[c]
mov [T1], ax
mov ax,[T2]
mov bx,[T1]
div bx
mov [T3],ax
mov ax,[a]
mul word [T3]
mov [T2], ax
mov ax, [T2]
mov [ans], ax
mov ax, [ans]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
; exit code
fini:
```

```
        mov eax,sys_exit ;terminate, sys_exit = 1
        xor ebx,ebx      ;successfully, zero in ebx indicates success
        int 80h

;       END PgmIO1.asm

;
;     Subroutine to print a string on the display
;
; Input:
;     DS:BX = pointer to the string to print
;
; Output: None
;
; Registers destroyed: none
;
;PrintString    PROC
PrintString:
        push   ax           ;Save registers;
        push   dx
; subpgm:
        ; prompt user
        mov eax, 4                  ;Linux print device register conventions
        mov ebx, 1                  ; print default output device
        mov ecx, userMsg            ; pointer to string
        mov edx, lenUserMsg         ; arg1, where to write, screen
        int      80h                ; interrupt 80 hex, call kernel
        pop    dx           ;Restore registers.
        pop    ax
        ret
;PrintString    ENDP

;%NEWPAGE

;

; Subroutine to get an integer (character string) from the keyboard buffer
;   and convert it to a 16 bit binary number.
;
; Input: none
;
; Output: The integer is returned in the AX register as well as the global
;       variable ReadInt .
;
; Registers Destroyed: AX, BX, CX, DX, SI
;
; Globals Destroyed: ReadInt, TempChar, tempint, negflag
;
;GetAnInteger   PROC
```

```
GetAnInteger:   ;Get an integer as a string
        ;get response
        mov eax,3       ;read
        mov ebx,2       ;device
        mov ecx,num     ;buffer address
        mov edx,6       ;max characters
        int 0x80

        ;print number   ;works
        mov edx,eax     ; eax contains the number of character read including <lf>
        mov eax, 4
        mov ebx, 1
        mov ecx, num
        int 80h

ConvertStringToInteger:
        mov ax,0        ;hold integer
        mov [ReadInt],ax ;initialize 16 bit number to zero
        mov ecx,num     ;pt - 1st or next digit of number as a string
                        ;terminated by <lf>.
        mov bx,0
        mov bl, byte [ecx] ;contains first or next digit
Next:   sub bl,'0'      ;convert character to number
        mov ax,[ReadInt]
        mov dx,10
        mul dx          ;eax = eax * 10
        add ax,bx
        mov [ReadInt], ax

        mov bx,0
        add ecx,1       ;pt = pt + 1
        mov bl, byte[ecx]

        cmp bl,0xA      ;is it a <lf>
        jne Next        ; get next digit   &&&&&&&&&&&&&&&&&&&&&&&&7
        ret
;       ENDP GetAnInteger



;%NEWPAGE
;
; Subroutine to convert a 16 bit integer to a text string
;
; input:
;       AX = number to convert
;       DS:BX = pointer to end of string to store text
;       CX = number of digits to convert
```

```
;
; output: none
;
; Registers destroyed: AX, BX, CX, DX, SI
; Globals destroyed negflag
;
;ConvertIntegerToString PROC

ConvertIntegerToString:
        mov ebx, ResultValue + 4   ;Store the integer as a five
                        ;digit char string at Result for printing

ConvertLoop:
        sub dx,dx  ; repeatedly divide dx:ax by 10 to obtain last digit of number
        mov cx,10  ; as the remainder in the DX register.  Quotient in AX.
        div cx
        add dl,'0' ; Add '0' to dl to convert from binary to character.
        mov [ebx], dl
        dec ebx
        cmp ebx,ResultValue
        jge ConvertLoop

        ret

;ConvertIntegerToString  ENDP
```

Linux VPS output:

```
developer@vultr:~/compilers$ nasm -f elf64 finalProgram.asm -o finalProgram.o
ld -o finalProgram finalProgram.o
./finalProgram
Enter an integer(less than 32,765): 5
5
Enter an integer(less than 32,765): 2
2
Enter an integer(less than 32,765): 30
30
Enter an integer(less than 32,765): 20
20
Ans = 00080
developer@vultr:~/compilers$
```

Question Two

Java0 Source Code:

```
CLASS PGM1{
PROCEDURE questionTwo(){
VAR A, B, C;
READ A;
READ B;
READ C;
IF A > B THEN{
IF A > C THEN{
PRINTNUMBER A;
}
ELSE{
PRINTNUMBER C;
}
}
ELSE{
IF B >= C THEN{
PRINTNUMBER B;
}
ELSE{
PRINTNUMBER C;
}
}
}
}
```

Pass One Token List:

```
CLASS CLASS
PGM1 <var>
{ ${
PROCEDURE PROCEDURE
questionTwo <var>
( $(
) $)
{ ${
VAR VAR
A <var>
, $,
B <var>
, $,
C <var>
; $;
READ READ
A <var>
```

; $;
READ READ
B <var>
; $;
READ READ
C <var>
; $;
IF IF
A <var>
> $>
B <var>
THEN THEN
{ ${
IF IF
A <var>
> $>
C <var>
THEN THEN
{ ${
PRINTNUMBER PRINTNUMBER
A <var>
; $;
} $}
ELSE ELSE
{ ${
PRINTNUMBER PRINTNUMBER
C <var>
; $;
} $}
} $}
ELSE ELSE
{ ${
IF IF
B <var>
>= $>=
C <var>
THEN THEN
{ ${
PRINTNUMBER PRINTNUMBER
B <var>
; $;
} $}
ELSE ELSE
{ ${
PRINTNUMBER PRINTNUMBER
C <var>
; $;
} $}
} $}

```
} $}
} $}
EOF $EOF
```

Pass Two Symbol Table

```
0 PGM1 <PROGRAM NAME> Null 0 CS
1 questionTwo <PROCEDURE NAME> Null 0 CS
2 A Var Null 0 DS
3 B Var Null 2 DS
4 C Var Null 4 DS
5 T1 Var Null 6 DS
5 T2 Var Null 8 DS
5 T3 Var Null 10 DS
```

Syntax Analyzer Quad Output:

```
READ A - -
READ B - -
READ C - -
IF - - -
> A B -
THEN L1 JLE -
IF - - -
> A C -
THEN L2 JLE -
PRINTNUMBER A - -
L3 JMP - -
L2 NOP - -
PRINTNUMBER C - -
L3 NOP - -
L4 JMP - -
L1 NOP - -
IF - - -
>= B C -
THEN L5 JL -
PRINTNUMBER B - -
L6 JMP - -
L5 NOP - -
PRINTNUMBER C - -
L6 NOP - -
L4 NOP - -
```

Semantics Assembly Output:

```
call PrintString
```

```
call GetAnInteger
mov ax, [ReadInt]
mov [A], ax
call PrintString
call GetAnInteger
mov ax, [ReadInt]
mov [B], ax
call PrintString
call GetAnInteger
mov ax, [ReadInt]
mov [C], ax
mov ax,[A]
cmp ax,[B]
JLE L1
mov ax,[A]
cmp ax,[C]
JLE L2
mov ax, [A]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
JMP L3
L2 NOP
mov ax, [C]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
L3 NOP
JMP L4
L1 NOP
mov ax,[B]
cmp ax,[C]
JL L5
mov ax, [B]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
JMP L6
L5 NOP
mov ax, [C]
```

```
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
L6 NOP
L4 NOP
```

Final .ASM file:

```
sys_exit        equ     1
sys_read        equ     3
sys_write       equ     4
stdin           equ     0 ; default keyboard
stdout          equ     1 ; default terminal screen
stderr          equ     3

section .data               ;used to declare constants
        userMsg                 db      'Enter an integer(less than 32,765): '
        lenUserMsg      equ     $-userMsg
        displayMsg      db      'You entered: '
        lenDisplayMsg   equ     $-displayMsg
        newline         db      0xA     ; 0xA 0xD is ASCII <LF><CR>

        Ten         DW      10  ;Used in converting to base ten.

        printTempchar   db      'Tempchar = : '
        lenprintTempchar        equ     $-printTempchar


        Result      db      'Ans = '
        ResultValue     db      'aaaaa'
                        db      0xA
        ResultEnd       equ     $-Result    ; $=> here, subtract address Result

        num             times 6 db 'ABCDEF'
        numEnd                  equ     $-num
; Start of user variable area    ----------------------------------------------

section  .bss               ;used to declare uninitialized variables

        TempChar        RESB    1           ;temp space for use by GetNextChar
        testchar        RESB    1
        ReadInt         RESW    1           ;Temporary storage GetAnInteger.
        tempint         RESW    1           ;Used in converting to base ten.
        negflag         RESB    1           ;P=positive, N=negative
```

```
A RESW 1
B RESW 1
C RESW 1
T1 RESW 1
T2 RESW 1
T3 RESW 1
        global _start
section .text

_start:  nop
        ; prompt user for positive number

Again:
call PrintString
call GetAnInteger
mov ax, [ReadInt]
mov [A], ax
call PrintString
call GetAnInteger
mov ax, [ReadInt]
mov [B], ax
call PrintString
call GetAnInteger
mov ax, [ReadInt]
mov [C], ax
mov ax,[A]
cmp ax,[B]
JLE L1
mov ax,[A]
cmp ax,[C]
JLE L2
mov ax, [A]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
JMP L3
L2 NOP
mov ax, [C]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
L3 NOP
JMP L4
```

```
L1 NOP
mov ax,[B]
cmp ax,[C]
JL L5
mov ax, [B]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
JMP L6
L5 NOP
mov ax, [C]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
L6 NOP
L4 NOP
; exit code
fini:
        mov eax,sys_exit ;terminate, sys_exit = 1
        xor ebx,ebx      ;successfully, zero in ebx indicates success
        int 80h

;       END PgmIO1.asm

;
;     Subroutine to print a string on the display
;
; Input:
;     DS:BX = pointer to the string to print
;
; Output: None
;
; Registers destroyed: none
;
;PrintString    PROC
PrintString:
        push   ax           ;Save registers;
        push   dx
; subpgm:
        ; prompt user
        mov eax, 4              ;Linux print device register conventions
        mov ebx, 1              ; print default output device
        mov ecx, userMsg        ; pointer to string
```

```
        mov edx, lenUserMsg    ; arg1, where to write, screen
        int     80h            ; interrupt 80 hex, call kernel
        pop     dx             ;Restore registers.
        pop     ax
        ret
;PrintString    ENDP

;%NEWPAGE

;

; Subroutine to get an integer (character string) from the keyboard buffer
;    and convert it to a 16 bit binary number.
;
; Input: none
;
; Output: The integer is returned in the AX register as well as the global
;         variable ReadInt .
;
; Registers Destroyed: AX, BX, CX, DX, SI
;
; Globals Destroyed: ReadInt, TempChar, tempint, negflag
;
;GetAnInteger    PROC

GetAnInteger:   ;Get an integer as a string
        ;get response
        mov eax,3       ;read
        mov ebx,2       ;device
        mov ecx,num     ;buffer address
        mov edx,6       ;max characters
        int 0x80

        ;print number    ;works
        mov edx,eax     ; eax contains the number of character read including <lf>
        mov eax, 4
        mov ebx, 1
        mov ecx, num
        int 80h

ConvertStringToInteger:
        mov ax,0        ;hold integer
        mov [ReadInt],ax ;initialize 16 bit number to zero
        mov ecx,num     ;pt - 1st or next digit of number as a string
                        ;terminated by <lf>.
        mov bx,0
        mov bl, byte [ecx] ;contains first or next digit
Next:   sub bl,'0'      ;convert character to number
        mov ax,[ReadInt]
```

```
        mov dx,10
        mul dx          ;eax = eax * 10
        add ax,bx
        mov [ReadInt], ax

        mov bx,0
        add ecx,1       ;pt = pt + 1
        mov bl, byte[ecx]

        cmp bl,0xA      ;is it a <lf>
        jne Next        ; get next digit   &&&&&&&&&&&&&&&&&&&&&&&&7
        ret
;       ENDP GetAnInteger



;%NEWPAGE
;
; Subroutine to convert a 16 bit integer to a text string
;
; input:
;       AX = number to convert
;       DS:BX = pointer to end of string to store text
;       CX = number of digits to convert
;
; output: none
;
; Registers destroyed: AX, BX, CX, DX, SI
; Globals destroyed negflag
;
;ConvertIntegerToString PROC

ConvertIntegerToString:
        mov ebx, ResultValue + 4   ;Store the integer as a five
                        ;digit char string at Result for printing

ConvertLoop:
        sub dx,dx  ; repeatedly divide dx:ax by 10 to obtain last digit of number
        mov cx,10  ; as the remainder in the DX register.  Quotient in AX.
        div cx
        add dl,'0' ; Add '0' to dl to convert from binary to character.
        mov [ebx], dl
        dec ebx
        cmp ebx,ResultValue
        jge ConvertLoop

        ret

;ConvertIntegerToString  ENDP
```

Linux VPS output:

```
developer@vultr:~/compilers$ nasm -f elf64 finalProgram.asm -o finalProgram.o
ld -o finalProgram finalProgram.o
./finalProgram
Enter an integer(less than 32,765): 10
10
Enter an integer(less than 32,765): 5
5
Enter an integer(less than 32,765): 2
2
Ans = 00010
developer@vultr:~/compilers$ ./finalProgram
Enter an integer(less than 32,765): 3
3
Enter an integer(less than 32,765): 11
11
Enter an integer(less than 32,765): 6
6
Ans = 00011
developer@vultr:~/compilers$ ./finalProgram
Enter an integer(less than 32,765): 1
1
Enter an integer(less than 32,765): 7
7
Enter an integer(less than 32,765): 34
34
Ans = 00034
```

Question Three/Five

Java0 Source Code:

```
CLASS PGM1{
PROCEDURE NFactorial(){
VAR N, I, ANSWER;
READ N;
I = N - 1;
WHILE I > 1 DO {
N = N * I;
I = I - 1;
}
PRINTNUMBER N;
}
}
```

Pass One Token List:

```
CLASS CLASS
PGM1 <var>
{ ${
PROCEDURE PROCEDURE
NFactorial <var>
( $(
) $)
{ ${
VAR VAR
N <var>
, $,
I <var>
, $,
ANSWER <var>
; $;
READ READ
N <var>
; $;
I <var>
= $=
N <var>
- $-
1 <int>
; $;
WHILE WHILE
I <var>
> $>
1 <int>
DO DO
```

```
{ ${
N <var>
= $=
N <var>
* $*
I <var>
; $;
I <var>
= $=
I <var>
- $-
1 <int>
; $;
} $}
PRINTNUMBER PRINTNUMBER
N <var>
; $;
} $}
} $}
EOF $EOF
```

Pass Two Symbol Table

```
0 PGM1 <PROGRAM NAME> Null 0 CS
1 NFactorial <PROCEDURE NAME> Null 0 CS
2 N Var Null 0 DS
3 I Var Null 2 DS
4 ANSWER Var Null 4 DS
5 1 NumericLiteral 1 6 DS
6 T1 Var Null 8 DS
6 T2 Var Null 10 DS
6 T3 Var Null 12 DS
```

Syntax Analyzer Quad Output:

```
READ N - -
- N lit1 T1
= I T1 -
WHILE W1 - -
> I lit1 -
DO L1 JLE -
* N I T1
= N T1 -
- I lit1 T1
= I T1 -
W1 JMP - -
L1 NOP - -
PRINTNUMBER N - -
```

Semantics Assembly Output:

```
call PrintString
call GetAnInteger
mov ax, [ReadInt]
mov [N], ax
mov ax,[N]
sub ax,[lit1]
mov [T1], ax
mov ax, [T1]
mov [I], ax
W1 NOP
mov ax,[I]
cmp ax,[lit1]
JLE L1
mov ax,[N]
mul word [I]
mov [T1], ax
mov ax, [T1]
mov [N], ax
mov ax,[I]
sub ax,[lit1]
mov [T1], ax
mov ax, [T1]
mov [I], ax
JMP W1
L1 NOP
mov ax, [N]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
```

Final .ASM file:

```
sys_exit        equ     1
sys_read        equ     3
sys_write       equ     4
stdin           equ     0 ; default keyboard
stdout          equ     1 ; default terminal screen
stderr          equ     3

section .data           ;used to declare constants
        userMsg                 db      'Enter an integer(less than 32,765): '
        lenUserMsg      equ     $-userMsg
        displayMsg      db      'You entered: '
```

```
        lenDisplayMsg  equ     $-displayMsg
        newline        db      0xA     ; 0xA 0xD is ASCII <LF><CR>

        Ten         DW     10  ;Used in converting to base ten.

        printTempchar  db      'Tempchar = : '
        lenprintTempchar       equ     $-printTempchar


        Result      db     'Ans = '
        ResultValue    db      'aaaaa'
                       db      0xA
        ResultEnd     equ      $-Result    ; $=> here, subtract address Result

        num            times 6 db 'ABCDEF'
        numEnd                 equ     $-num
lit1 DW 1
; Start of user variable area    ---------------------------------------------

section  .bss              ;used to declare uninitialized variables

        TempChar      RESB    1          ;temp space for use by GetNextChar
        testchar      RESB    1
        ReadInt       RESW    1          ;Temporary storage GetAnInteger.
        tempint       RESW     1          ;Used in converting to base ten.
        negflag       RESB    1         ;P=positive, N=negative
N RESW 1
I RESW 1
ANSWER RESW 1
T1 RESW 1
T2 RESW 1
T3 RESW 1
        global _start
section .text

_start:  nop
        ; prompt user for positive number

Again:
call PrintString
call GetAnInteger
mov ax, [ReadInt]
mov [N], ax
mov ax,[N]
sub ax,[lit1]
mov [T1], ax
mov ax, [T1]
mov [I], ax
W1 NOP
```

```
mov ax,[I]
cmp ax,[lit1]
JLE L1
mov ax,[N]
mul word [I]
mov [T1], ax
mov ax, [T1]
mov [N], ax
mov ax,[I]
sub ax,[lit1]
mov [T1], ax
mov ax, [T1]
mov [I], ax
JMP W1
L1 NOP
mov ax, [N]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
; exit code
fini:
        mov eax,sys_exit ;terminate, sys_exit = 1
        xor ebx,ebx      ;successfully, zero in ebx indicates success
        int 80h

;       END PgmIO1.asm

;
;     Subroutine to print a string on the display
;
; Input:
;     DS:BX = pointer to the string to print
;
; Output: None
;
; Registers destroyed: none
;
;PrintString    PROC
PrintString:
        push   ax           ;Save registers;
        push   dx
; subpgm:
        ; prompt user
        mov eax, 4              ;Linux print device register conventions
        mov ebx, 1              ; print default output device
        mov ecx, userMsg        ; pointer to string
```

```
        mov edx, lenUserMsg    ; arg1, where to write, screen
        int     80h            ; interrupt 80 hex, call kernel
        pop     dx             ;Restore registers.
        pop     ax
        ret
;PrintString    ENDP

;%NEWPAGE

;

; Subroutine to get an integer (character string) from the keyboard buffer
;    and convert it to a 16 bit binary number.
;
; Input: none
;
; Output: The integer is returned in the AX register as well as the global
;         variable ReadInt .
;
; Registers Destroyed: AX, BX, CX, DX, SI
;
; Globals Destroyed: ReadInt, TempChar, tempint, negflag
;
;GetAnInteger    PROC

GetAnInteger:   ;Get an integer as a string
        ;get response
        mov eax,3       ;read
        mov ebx,2       ;device
        mov ecx,num     ;buffer address
        mov edx,6       ;max characters
        int 0x80

        ;print number    ;works
        mov edx,eax     ; eax contains the number of character read including <lf>
        mov eax, 4
        mov ebx, 1
        mov ecx, num
        int 80h

ConvertStringToInteger:
        mov ax,0        ;hold integer
        mov [ReadInt],ax ;initialize 16 bit number to zero
        mov ecx,num     ;pt - 1st or next digit of number as a string
                        ;terminated by <lf>.
        mov bx,0
        mov bl, byte [ecx] ;contains first or next digit
Next:   sub bl,'0'      ;convert character to number
        mov ax,[ReadInt]
```

```
        mov dx,10
        mul dx          ;eax = eax * 10
        add ax,bx
        mov [ReadInt], ax

        mov bx,0
        add ecx,1       ;pt = pt + 1
        mov bl, byte[ecx]

        cmp bl,0xA      ;is it a <lf>
        jne Next        ; get next digit   &&&&&&&&&&&&&&&&&&&&&&&&7
        ret
;       ENDP GetAnInteger



;%NEWPAGE
;
; Subroutine to convert a 16 bit integer to a text string
;
; input:
;       AX = number to convert
;       DS:BX = pointer to end of string to store text
;       CX = number of digits to convert
;
; output: none
;
; Registers destroyed: AX, BX, CX, DX, SI
; Globals destroyed negflag
;
;ConvertIntegerToString PROC

ConvertIntegerToString:
        mov ebx, ResultValue + 4   ;Store the integer as a five
                        ;digit char string at Result for printing

ConvertLoop:
        sub dx,dx  ; repeatedly divide dx:ax by 10 to obtain last digit of number
        mov cx,10  ; as the remainder in the DX register.  Quotient in AX.
        div cx
        add dl,'0' ; Add '0' to dl to convert from binary to character.
        mov [ebx], dl
        dec ebx
        cmp ebx,ResultValue
        jge ConvertLoop

        ret

;ConvertIntegerToString  ENDP
```

Linux VPS output:

```
developer@vultr:~/compilers$ nasm -f elf64 finalProgram.asm -o finalProgram.o
ld -o finalProgram finalProgram.o
./finalProgram
Enter an integer(less than 32,765): 5
5
Ans = 00120
developer@vultr:~/compilers$
```

Question Four

Java0 Source Code:

```
CLASS PGM1{
PROCEDURE NestedWhiles(){
VAR J, K;
WHILE J < 10 DO {
K = 3;
J = J + 1;
PRINTNUMBER J;
WHILE K > 0 DO {
K = K - 1;
PRINTNUMBER K;
}
}
}
}

/* nested whiles, counts down from three ten times. */
```

Pass One Token List:

```
CLASS CLASS
PGM1 <var>
{ ${
PROCEDURE PROCEDURE
NestedWhiles <var>
( $(
) $)
{ ${
VAR VAR
J <var>
, $,
K <var>
; $;
WHILE WHILE
J <var>
< $<
10 <int>
DO DO
{ ${
K <var>
= $=
3 <int>
; $;
J <var>
= $=
```

```
J <var>
+ $+
1 <int>
; $;
PRINTNUMBER PRINTNUMBER
J <var>
; $;
WHILE WHILE
K <var>
> $>
0 <int>
DO DO
{ ${
K <var>
= $=
K <var>
- $-
1 <int>
; $;
PRINTNUMBER PRINTNUMBER
K <var>
; $;
} $}
} $}
} $}
} $}
EOF $EOF
```

Pass Two Symbol Table

```
0 PGM1 <PROGRAM NAME> Null 0 CS
1 NestedWhiles <PROCEDURE NAME> Null 0 CS
2 J Var Null 0 DS
3 K Var Null 2 DS
4 10 NumericLiteral 10 4 DS
5 3 NumericLiteral 3 6 DS
6 1 NumericLiteral 1 8 DS
7 0 NumericLiteral 0 10 DS
8 T1 Var Null 12 DS
8 T2 Var Null 14 DS
8 T3 Var Null 16 DS
```

Syntax Analyzer Quad Output:

```
WHILE W1 - -
< J lit10 -
DO L1 JGE -
= K lit3 -
```

```
+ J lit1 T1
= J T1 -
PRINTNUMBER J - -
WHILE W2 - -
> K lit0 -
DO L2 JLE -
- K lit1 T1
= K T1 -
PRINTNUMBER K - -
W2 JMP - -
L2 NOP - -
W1 JMP - -
L1 NOP - -
```

Semantics Assembly Output:

```
W1 NOP
mov ax,[J]
cmp ax,[lit10]
JGE L1
mov ax, [lit3]
mov [K], ax
mov ax,[J]
add ax,[lit1]
mov [T1], ax
mov ax, [T1]
mov [J], ax
mov ax, [J]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
W2 NOP
mov ax,[K]
cmp ax,[lit0]
JLE L2
mov ax,[K]
sub ax,[lit1]
mov [T1], ax
mov ax, [T1]
mov [K], ax
mov ax, [K]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
```

```
mov ecx, Result
mov edx, ResultEnd
int 80h
JMP W2
L2 NOP
JMP W1
L1 NOP
```

Final .ASM file:

```
sys_exit        equ     1
sys_read        equ     3
sys_write       equ     4
stdin           equ     0 ; default keyboard
stdout          equ     1 ; default terminal screen
stderr          equ     3

section .data               ;used to declare constants
        userMsg                 db      'Enter an integer(less than 32,765): '
        lenUserMsg      equ     $-userMsg
        displayMsg      db      'You entered: '
        lenDisplayMsg   equ     $-displayMsg
        newline         db      0xA     ; 0xA 0xD is ASCII <LF><CR>

        Ten         DW      10  ;Used in converting to base ten.

        printTempchar   db      'Tempchar = : '
        lenprintTempchar        equ     $-printTempchar


        Result      db      'Ans = '
        ResultValue     db      'aaaaa'
                        db      0xA
        ResultEnd       equ     $-Result    ; $=> here, subtract address Result

        num             times 6 db 'ABCDEF'
        numEnd                  equ     $-num
lit10 DW 10
lit3 DW 3
lit1 DW 1
lit0 DW 0
; Start of user variable area    ----------------------------------------------

section  .bss               ;used to declare uninitialized variables

        TempChar        RESB    1           ;temp space for use by GetNextChar
        testchar        RESB    1
```

```
        ReadInt     RESW    1           ;Temporary storage GetAnInteger.
        tempint     RESW    1           ;Used in converting to base ten.
        negflag     RESB    1           ;P=positive, N=negative
J RESW 1
K RESW 1
T1 RESW 1
T2 RESW 1
T3 RESW 1
        global _start
section .text

_start: nop
        ; prompt user for positive number

Again:
W1 NOP
mov ax,[J]
cmp ax,[lit10]
JGE L1
mov ax, [lit3]
mov [K], ax
mov ax,[J]
add ax,[lit1]
mov [T1], ax
mov ax, [T1]
mov [J], ax
mov ax, [J]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
W2 NOP
mov ax,[K]
cmp ax,[lit0]
JLE L2
mov ax,[K]
sub ax,[lit1]
mov [T1], ax
mov ax, [T1]
mov [K], ax
mov ax, [K]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
```

```
JMP W2
L2 NOP
JMP W1
L1 NOP
; exit code
fini:
        mov eax,sys_exit ;terminate, sys_exit = 1
        xor ebx,ebx      ;successfully, zero in ebx indicates success
        int 80h

;       END PgmIO1.asm

;
;     Subroutine to print a string on the display
;
; Input:
;     DS:BX = pointer to the string to print
;
; Output: None
;
; Registers destroyed: none
;
;PrintString    PROC
PrintString:
        push   ax           ;Save registers;
        push   dx
; subpgm:
        ; prompt user
        mov eax, 4                  ;Linux print device register conventions
        mov ebx, 1                  ; print default output device
        mov ecx, userMsg            ; pointer to string
        mov edx, lenUserMsg         ; arg1, where to write, screen
        int       80h               ; interrupt 80 hex, call kernel
        pop    dx           ;Restore registers.
        pop    ax
        ret
;PrintString    ENDP

;%NEWPAGE

;

; Subroutine to get an integer (character string) from the keyboard buffer
;   and convert it to a 16 bit binary number.
;
; Input: none
;
; Output: The integer is returned in the AX register as well as the global
;       variable ReadInt .
```

```
;
; Registers Destroyed: AX, BX, CX, DX, SI
;
; Globals Destroyed: ReadInt, TempChar, tempint, negflag
;
;GetAnInteger    PROC

GetAnInteger:   ;Get an integer as a string
        ;get response
        mov eax,3       ;read
        mov ebx,2       ;device
        mov ecx,num     ;buffer address
        mov edx,6       ;max characters
        int 0x80

        ;print number    ;works
        mov edx,eax     ; eax contains the number of character read including <lf>
        mov eax, 4
        mov ebx, 1
        mov ecx, num
        int 80h

ConvertStringToInteger:
        mov ax,0        ;hold integer
        mov [ReadInt],ax ;initialize 16 bit number to zero
        mov ecx,num     ;pt - 1st or next digit of number as a string
                        ;terminated by <lf>.
        mov bx,0
        mov bl, byte [ecx] ;contains first or next digit
Next:   sub bl,'0'      ;convert character to number
        mov ax,[ReadInt]
        mov dx,10
        mul dx          ;eax = eax * 10
        add ax,bx
        mov [ReadInt], ax

        mov bx,0
        add ecx,1       ;pt = pt + 1
        mov bl, byte[ecx]

        cmp bl,0xA      ;is it a <lf>
        jne Next        ; get next digit   &&&&&&&&&&&&&&&&&&&&&&&&7
        ret
;       ENDP GetAnInteger



;%NEWPAGE
;
```

```
; Subroutine to convert a 16 bit integer to a text string
;
; input:
;       AX = number to convert
;       DS:BX = pointer to end of string to store text
;       CX = number of digits to convert
;
; output: none
;
; Registers destroyed: AX, BX, CX, DX, SI
; Globals destroyed negflag
;
;ConvertIntegerToString PROC

ConvertIntegerToString:
        mov ebx, ResultValue + 4   ;Store the integer as a five
                        ;digit char string at Result for printing

ConvertLoop:
        sub dx,dx  ; repeatedly divide dx:ax by 10 to obtain last digit of number
        mov cx,10  ; as the remainder in the DX register.  Quotient in AX.
        div cx
        add dl,'0' ; Add '0' to dl to convert from binary to character.
        mov [ebx], dl
        dec ebx
        cmp ebx,ResultValue
        jge ConvertLoop

        ret

;ConvertIntegerToString  ENDP
```

Linux VPS output:

```
developer@vultr:~/compilers$ nasm -f elf64 finalProgram.asm -o finalProgram.o
ld -o finalProgram finalProgram.o
./finalProgram
Ans = 00001
Ans = 00002
Ans = 00001
Ans = 00000
Ans = 00002
Ans = 00002
Ans = 00001
Ans = 00000
Ans = 00003
Ans = 00002
```

```
Ans = 00001
Ans = 00000
Ans = 00004
Ans = 00002
Ans = 00001
Ans = 00000
Ans = 00005
Ans = 00002
Ans = 00001
Ans = 00000
Ans = 00006
Ans = 00002
Ans = 00001
Ans = 00000
Ans = 00007
Ans = 00002
Ans = 00001
Ans = 00000
Ans = 00008
Ans = 00002
Ans = 00001
Ans = 00000
Ans = 00009
Ans = 00002
Ans = 00001
Ans = 00000
Ans = 00010
Ans = 00002
Ans = 00001
Ans = 00000
developer@vultr:~/compilers$
```

Bonus Question (5 nested IFS)

Java0 Source Code:

```
CLASS PGM1{
PROCEDURE fiveNestedIfs(){
VAR J;
READ J;
IF J < 50 THEN{
PRINTNUMBER J;
IF J < 40 THEN{
PRINTNUMBER J;
IF J < 30 THEN{
PRINTNUMBER J;
IF J < 20 THEN{
PRINTNUMBER J;
IF J < 10 THEN{
PRINTNUMBER J;
}
}
}
}
}
}
}
```

/* Five If statements to prove nesting capabilities. It will check if the number is less than 60, then 50, and so on until 10. */

Pass One Token List:

```
CLASS CLASS
PGM1 <var>
{ ${
PROCEDURE PROCEDURE
fiveNestedIfs <var>
( $(
) $)
{ ${
VAR VAR
J <var>
; $;
READ READ
J <var>
; $;
IF IF
J <var>
< $<
```

50 <int>
THEN THEN
{ ${
PRINTNUMBER PRINTNUMBER
J <var>
; $;
IF IF
J <var>
< $<
40 <int>
THEN THEN
{ ${
PRINTNUMBER PRINTNUMBER
J <var>
; $;
IF IF
J <var>
< $<
30 <int>
THEN THEN
{ ${
PRINTNUMBER PRINTNUMBER
J <var>
; $;
IF IF
J <var>
< $<
20 <int>
THEN THEN
{ ${
PRINTNUMBER PRINTNUMBER
J <var>
; $;
IF IF
J <var>
< $<
10 <int>
THEN THEN
{ ${
PRINTNUMBER PRINTNUMBER
J <var>
; $;
} $}
} $}
} $}
} $}
} $}
} $}
} $}

EOF $EOF

Pass Two Symbol Table

0 PGM1 <PROGRAM NAME> Null 0 CS
1 fiveNestedIfs <PROCEDURE NAME> Null 0 CS
2 J Var Null 0 DS
3 50 NumericLiteral 50 2 DS
4 40 NumericLiteral 40 4 DS
5 30 NumericLiteral 30 6 DS
6 20 NumericLiteral 20 8 DS
7 10 NumericLiteral 10 10 DS
8 T1 Var Null 12 DS
8 T2 Var Null 14 DS
8 T3 Var Null 16 DS

Syntax Analyzer Quad Output:

READ J - -
IF - - -
< J lit50 -
THEN L1 JGE -
PRINTNUMBER J - -
IF - - -
< J lit40 -
THEN L2 JGE -
PRINTNUMBER J - -
IF - - -
< J lit30 -
THEN L3 JGE -
PRINTNUMBER J - -
IF - - -
< J lit20 -
THEN L4 JGE -
PRINTNUMBER J - -
IF - - -
< J lit10 -
THEN L5 JGE -
PRINTNUMBER J - -
L5 NOP - -
L4 NOP - -
L3 NOP - -
L2 NOP - -
L1 NOP - -

Semantics Assembly Output:

```
call PrintString
call GetAnInteger
mov ax, [ReadInt]
mov [J], ax
mov ax,[J]
cmp ax,[lit50]
JGE L1
mov ax, [J]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
mov ax,[J]
cmp ax,[lit40]
JGE L2
mov ax, [J]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
mov ax,[J]
cmp ax,[lit30]
JGE L3
mov ax, [J]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
mov ax,[J]
cmp ax,[lit20]
JGE L4
mov ax, [J]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
mov ax,[J]
cmp ax,[lit10]
JGE L5
mov ax, [J]
```

```
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
L5 NOP
L4 NOP
L3 NOP
L2 NOP
L1 NOP
```

Final .ASM file:

```
sys_exit        equ     1
sys_read        equ     3
sys_write       equ     4
stdin           equ     0 ; default keyboard
stdout          equ     1 ; default terminal screen
stderr          equ     3

section .data               ;used to declare constants
        userMsg                 db      'Enter an integer(less than 32,765): '
        lenUserMsg      equ     $-userMsg
        displayMsg      db      'You entered: '
        lenDisplayMsg   equ     $-displayMsg
        newline         db      0xA     ; 0xA 0xD is ASCII <LF><CR>

        Ten         DW      10  ;Used in converting to base ten.

        printTempchar   db      'Tempchar = : '
        lenprintTempchar        equ     $-printTempchar


        Result      db      'Ans = '
        ResultValue     db      'aaaaa'
                        db      0xA
        ResultEnd       equ     $-Result    ; $=> here, subtract address Result

        num             times 6 db 'ABCDEF'
        numEnd                  equ     $-num
lit50 DW 50
lit40 DW 40
lit30 DW 30
lit20 DW 20
lit10 DW 10
; Start of user variable area    ---------------------------------------------
```

```
section  .bss                ;used to declare uninitialized variables

         TempChar      RESB    1          ;temp space for use by GetNextChar
         testchar      RESB    1
         ReadInt       RESW    1          ;Temporary storage GetAnInteger.
         tempint       RESW      1          ;Used in converting to base ten.
         negflag       RESB    1          ;P=positive, N=negative
J RESW 1
T1 RESW 1
T2 RESW 1
T3 RESW 1
         global _start
section .text

_start:  nop
         ; prompt user for positive number

Again:
call PrintString
call GetAnInteger
mov ax, [ReadInt]
mov [J], ax
mov ax,[J]
cmp ax,[lit50]
JGE L1
mov ax, [J]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
mov ax,[J]
cmp ax,[lit40]
JGE L2
mov ax, [J]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
mov ax,[J]
cmp ax,[lit30]
JGE L3
mov ax, [J]
call ConvertIntegerToString
mov eax, 4
```

```
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
mov ax,[J]
cmp ax,[lit20]
JGE L4
mov ax, [J]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
mov ax,[J]
cmp ax,[lit10]
JGE L5
mov ax, [J]
call ConvertIntegerToString
mov eax, 4
mov ebx, 1
mov ecx, Result
mov edx, ResultEnd
int 80h
L5 NOP
L4 NOP
L3 NOP
L2 NOP
L1 NOP
; exit code
fini:
        mov eax,sys_exit ;terminate, sys_exit = 1
        xor ebx,ebx      ;successfully, zero in ebx indicates success
        int 80h

;       END PgmIO1.asm

;
;     Subroutine to print a string on the display
;
; Input:
;     DS:BX = pointer to the string to print
;
; Output: None
;
; Registers destroyed: none
;
;PrintString    PROC
PrintString:
```

```
        push    ax          ;Save registers;
        push    dx
; subpgm:
        ; prompt user
        mov eax, 4              ;Linux print device register conventions
        mov ebx, 1              ; print default output device
        mov ecx, userMsg        ; pointer to string
        mov edx, lenUserMsg     ; arg1, where to write, screen
        int     80h             ; interrupt 80 hex, call kernel
        pop     dx          ;Restore registers.
        pop     ax
        ret
;PrintString    ENDP

;%NEWPAGE

;

; Subroutine to get an integer (character string) from the keyboard buffer
;    and convert it to a 16 bit binary number.
;
; Input: none
;
; Output: The integer is returned in the AX register as well as the global
;         variable ReadInt .
;
; Registers Destroyed: AX, BX, CX, DX, SI
;
; Globals Destroyed: ReadInt, TempChar, tempint, negflag
;
;GetAnInteger    PROC

GetAnInteger:   ;Get an integer as a string
        ;get response
        mov eax,3       ;read
        mov ebx,2       ;device
        mov ecx,num     ;buffer address
        mov edx,6       ;max characters
        int 0x80

        ;print number    ;works
        mov edx,eax     ; eax contains the number of character read including <lf>
        mov eax, 4
        mov ebx, 1
        mov ecx, num
        int 80h

ConvertStringToInteger:
        mov ax,0        ;hold integer
```

```
        mov [ReadInt],ax ;initialize 16 bit number to zero
        mov ecx,num     ;pt - 1st or next digit of number as a string
                        ;terminated by <lf>.
        mov bx,0
        mov bl, byte [ecx] ;contains first or next digit
Next:   sub bl,'0'      ;convert character to number
        mov ax,[ReadInt]
        mov dx,10
        mul dx          ;eax = eax * 10
        add ax,bx
        mov [ReadInt], ax

        mov bx,0
        add ecx,1       ;pt = pt + 1
        mov bl, byte[ecx]

        cmp bl,0xA      ;is it a <lf>
        jne Next        ; get next digit   &&&&&&&&&&&&&&&&&&&&&&&&7
        ret
;       ENDP GetAnInteger



;%NEWPAGE
;
; Subroutine to convert a 16 bit integer to a text string
;
; input:
;       AX = number to convert
;       DS:BX = pointer to end of string to store text
;       CX = number of digits to convert
;
; output: none
;
; Registers destroyed: AX, BX, CX, DX, SI
; Globals destroyed negflag
;
;ConvertIntegerToString PROC

ConvertIntegerToString:
        mov ebx, ResultValue + 4   ;Store the integer as a five
                        ;digit char string at Result for printing

ConvertLoop:
        sub dx,dx  ; repeatedly divide dx:ax by 10 to obtain last digit of number
        mov cx,10  ; as the remainder in the DX register.  Quotient in AX.
        div cx
        add dl,'0' ; Add '0' to dl to convert from binary to character.
        mov [ebx], dl
```

```
        dec ebx
        cmp ebx,ResultValue
        jge ConvertLoop

        ret

;ConvertIntegerToString  ENDP
```

Linux VPS output:

```
developer@vultr:~/compilers$ nasm -f elf64 finalProgram.asm -o finalProgram.o
ld -o finalProgram finalProgram.o
./finalProgram
Enter an integer(less than 32,765): 31
31
Ans = 00031
Ans = 00031
developer@vultr:~/compilers$ ./finalProgram
Enter an integer(less than 32,765): 60
60
developer@vultr:~/compilers$ ./finalProgram
Enter an integer(less than 32,765): 9
9
Ans = 00009
Ans = 00009
Ans = 00009
Ans = 00009
Ans = 00009
developer@vultr:~/compilers$
```