Hai Nguyen

CPSC 323

Professor Anthony Le

December 10, 2019

**Assignment 3 (Intermediate Code Generator) Documentation**

Operating System: Windows 10

Language: C++ (Visual Studio 2017)

Description: This program reads a source code file and generate assembly code (using predictive RDP and Syntax Directed Translation). The result is printed to the screen and "assembly\_code.txt" file.

**1. Problem Statement:**

Build an intermediate code generator using predictive RDP and Syntax Directed Translation.

**2. How to use the program:**

- User enters a source code file (option 2 in main menu).

- The program generates a stream of tokens and analyze the syntax.

- If the code is correct:

- The program will print “Finished” to the screen, then print the assembly code and the symbol table to the screen and “assembly\_code.txt” file.

- Otherwise, it will print an error message.

Note:

- The program also prints the token stream from Lexical Analysis phase to “analysis.txt” file, and the parsing process from Syntax Analysis phase to “parse\_result.txt” file.

**3. Design of the program**:

- This program use a predictive Recursive Descent Parser and Syntax Directed Translation to compile the code.

- For this assignment, I assume that PUSHI also works with real numbers, “true” is 1, and “false” is 0.

**Production rules:**

<Statement> -> <Expression statement> | <Assign> | <Declare> | <If> | <While>

Assign statement:

<Assign> -> id = <Expression>;

<Expression> -> <Term> <ExpressionPrime>

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Term> -> <Factor> <TermPrime>

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Factor> -> - <Primary> | <Primary>

<Primary> -> id | int\_num | ( <Expression> ) | real\_num | true | false

<Empty> -> epsilon

Expression statement:

<Expression Statement> -> <Expression>;

<Expression> -> <Term> <ExpressionPrime>

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Term> -> <Factor> <TermPrime>

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Factor> -> - <Primary> | <Primary>

<Primary> -> id | int\_num | ( <Expression> ) | real\_num | true | false

<Empty> -> epsilon

Declare statement:

<Declare> -> <Type> id <MoreIDs>;

<Type> -> int | float | bool

<MoreIDs> -> , id <MoreIDs> | <Empty>

If statement:

<If> -> if <Conditional> then <StatementBlock> <ElseBlock> endif

<Conditional> -> <Expression> <ConditionalPrime>

<ConditionalPrime> -> <Relop> <Expression> | <Empty>

<Relop> -> < | <= | == | <> | >= | >

<StatementBlock> -> { <Statement> <moreStatement>

<moreStatement> -> <Statement> <moreStatement> | }

<ElseBlock> -> else <StatementBlock> | <Empty>

While statement:

<While> -> while <Conditional> do <StatementBlock> whileend

<Conditional> -> <Expression> <ConditionalPrime>

<ConditionalPrime> -> <Relop> <Expression> | <Empty>

<Relop> -> < | <= | == | <> | >= | >

<StatementBlock> -> { <Statement> <moreStatement>

<moreStatement> -> <Statement> <moreStatement> | }

**4. Any Limitation:**

- The program exits if it finds an error in the source code.

- The program does not check value types (“true + 1” is valid).

**5. Any Shortcomings:**

None

**6. Test cases:**

Case 1:

int a, b;

float num1;

b = 2;

a = b + 2 \* 3;

1 PUSHI 2

2 POPM 5001

3 PUSHM 5001

4 PUSHI 2

5 PUSHI 3

6 MUL nil

7 ADD nil

8 POPM 5000

IDENTIFIER ADDRESS TYPE

a 5000 int

b 5001 int

num1 5002 float

Case 2:

int a, max;

while a >= max do {

a = a / 2;

} whileend

1 Label nil

2 PUSHM 5000

3 PUSHM 5001

4 GEQ nil

5 JUMPZ 11

6 PUSHM 5000

7 PUSHI 2

8 DIV nil

9 POPM 5000

10 JUMP 1

IDENTIFIER ADDRESS TYPE

a 5000 int

max 5001 int

Case 3:

int max;

int a;

a = 0;

max = 15;

if a > max then {

a = max;

} endif

1 PUSHI 0

2 POPM 5001

3 PUSHI 15

4 POPM 5000

5 PUSHM 5001

6 PUSHM 5000

7 GRT nil

8 JUMPZ 11

9 PUSHM 5000

10 POPM 5001

IDENTIFIER ADDRESS TYPE

max 5000 int

a 5001 int

Case 4:

int max;

int a;

a = 0;

max = 15;

if a > max then {

a = max;

}

else{

a = max - 1;

} endif

1 PUSHI 0

2 POPM 5001

3 PUSHI 15

4 POPM 5000

5 PUSHM 5001

6 PUSHM 5000

7 GRT nil

8 JUMPZ 12

9 PUSHM 5000

10 POPM 5001

11 JUMP 16

12 PUSHM 5000

13 PUSHI 1

14 SUB nil

15 POPM 5001

IDENTIFIER ADDRESS TYPE

max 5000 int

a 5001 int