Team Wibbly Wobbles

CS153 Compiler Project

Compiler Project Write Up

Our team decided to implement a spinoff of the Go Language. Go is a C-style language with many similar aspects and also includes many extra features such as automatic memory management, variable-length arrays and a robust standard library. However, our compiler only implements a subset of its features with a few twists.

We opted for a Pascal-like implementation of passing by reference because C's pointer usage is too complicated. Whether or not a function call is passing by reference will be determined by the type signature of the function. We also required that each statement have a semi-colon at the end.

Our compiler implements the following features of the Go Language:

- Variable declarations and assignments
- arithmetic and relational operations on integers and floats (includes +, */, %, ==, etc.)
- concatenation of strings using the + operator
- string type inferencing when adding numbers and strings
- nested function declarations
- C-style for-loops
- While loops (This is a for loop with a condition in Go)
- If statements, switch statements
- Functions, including pass by value and pass by reference
- basic error handling using synchronization sets

We chose not to implement the following:

- library functions for input/output: fmt.println is simply println in our language
- the additional features of go: slices, channels, etc.

- various forms of the for-loop (for example, Go allows initialization, test condition, etc to all be omitted)

BNF Grammar for Go

TOKENS

```
<DEFAULT> SKIP : {
<IGNORE: [" ","\r","\n","\t"]>
| <"//" (~["\n","\r"])*>
| "/*" : MULTI_LINE_COMMENT
}

 <MULTI_LINE_COMMENT> SKIP : {
"*/" : DEFAULT
}

 <MULTI_LINE_COMMENT> MORE : {
<~[]>
}

 /********************
  -- Tokens --
 ********************/

// Reserved words token

 <DEFAULT> TOKEN : {
<BREAK: "break"> : {
| <DEFAULT_TOKEN: "default"> : {
| <FUNC: "func"> : {
| <CASE: "case"> : {
| <STRUCT: "struct"> : {
| <ELSE: "else"> : {
| <PACKAGE: "package"> : {
| <SWITCH: "switch"> : {
| <IF: "if"> : {
| <CONTINUE: "continue"> : {
| <FOR: "for"> : {
| <RETURN: "return"> : {
| <INT: "int"> : {
| <FLOAT: "float"> : {
| <STRING: "string"> : {
| <BOOL: "bool"> : {
```

```
| <VAR: "var"> : {
| <VOID: "void"> : {
}

 // Misc tokens

 <DEFAULT> TOKEN : {
<#NEWLINE: ["\r","\n"]>
| <#WHITE_SPACE: <NEWLINE> | "\t" | " ">
| <SEMICOLON: ";">
| <COMMA: ",">
| <DOT: ".">
| <OPEN_PAREN: "(" > : {
| <CLOSE_PAREN: ")" > : {
| <OPEN_BRACE: "{" > : {
| <CLOSE_BRACE: "}" > : {
| <OPEN_BRACKET: "[" > : {
| <CLOSE_BRACKET: "]" > : {
}

 // Special character tokens

 <DEFAULT> TOKEN : {
<PLUS: "+" > : {
| <INCREMENT: "++" > : {
| <DECREMENT: "--" > : {
| <MINUS: "-" > : {
| <MULTIPLY: "*" > : {
| <DIVIDE: "/" > : {
| <MODULO: "%" > : {
| <STAR_EQUAL: "*=" > : {
| <DIVIDE_EQUAL: "/=" > : {
| <MODULO_EQUAL: "%=" > : {
| <EQUALS: "=" > : {
| <EQUAL_EQUAL: "==" > : {
| <PLUS_EQUAL: "+=" > : {
| <MINUS_EQUAL: "-=" > : {
| <LESS_THAN: "<" > : {
| <GREATER_THAN: ">" > : {
| <LESS_EQUAL: "<=" > : {
| <GREATER_EQUAL: ">=" > : {
| <PIPE: "|" > : {
| <OR: "||" > : {
| <AMPERSAND: "&" > : {
| <AMPERSAND_EQUAL: "&=" > : {
| <AND: "&&" > : {
| <NOT_EQUAL: "!=" > : {
| <NOT: "!" > : {
| <CARET: "^" > : {
```

```
}

 // Types

 <DEFAULT> TOKEN : {
<INTEGER_NUMBER: ("-")? (<DIGIT>)+> : {
| <REAL_NUMBER: <INTEGER_NUMBER> "." (<INTEGER_NUMBER>)?> : {
| <BOOL_CONSTANT: "true" | "false"> : {
| <INTERPRETED_STRING: "\"" (<ALPHANUMERIC> | <ESCAPED_CHAR> |
<INTERPRETED_LITERAL_SYMBOL> | <WHITE_SPACE>)* "\""> : {
| <IDENTIFIER: <LETTER> (<LETTER> | <DIGIT> | "_")*> : {
| <#ALPHANUMERIC: (<LETTER> | <DIGIT>)+>
| <#ESCAPED_CHAR: "\\" ("a" | "b" | "f" | "n" | "r" | "t" | "v" | "\\" | "\'" | "\"")>
| <#LETTER: ["a"-"z","A"-"Z"]>
| <#DIGIT: ["0"-"9"]>
| <#INTERPRETED_LITERAL_SYMBOL: "\'" | "~" | "!" | "@" | "#" | "$" | "%" | "^" | "&" | "*" | "_" |
"-" | "+" | "=" | "<" | ">" | "." | "/" | "?" | ";" | "(" | ")" | "[" | "]" | "{" | "}" | "|" | "`" | ":">
}

 <DEFAULT> TOKEN : {
<ERROR: ~["\r","\n"]>
}


/*************************

  -- Production Rules --

 *************************/

start ::= <PACKAGE> <IDENTIFIER> statementList <EOF>

statement ::= ( declarationStatement | arrayDeclaration <SEMICOLON> | assignmentStatement

<SEMICOLON> | arrayAssignmentStatement <SEMICOLON> | increment <SEMICOLON> |

decrement <SEMICOLON> | switchStatement | ifStatement | forStatement | printStatement

<SEMICOLON> | functionCall <SEMICOLON> )

statementList ::= ( statement )*

declarationStatement ::= ( variableDeclaration <SEMICOLON> | functionDeclaration )

variableDeclaration ::= identifier ( type )

arrayDeclaration ::= <VAR> identifier <OPEN_BRACKET> <INTEGER_NUMBER>

<CLOSE_BRACKET> type
```

arrayAssignmentStatement ::= identifier <OPEN_BRACKET> <INTEGER_NUMBER>

<CLOSE_BRACKET> <EQUALS> operand

functionDeclaration ::= <FUNC> identifier <OPEN_PAREN> parameterList <CLOSE_PAREN>

returnType block

parameterList ::= (parameter (<COMMA> parameter)*)?

parameter ::= identifier (<MULTIPLY>)? (type)

functionCall ::= identifier <OPEN_PAREN> (expression (<COMMA> expression)*)?

<CLOSE_PAREN>

block ::= <OPEN_BRACE> statementList <CLOSE_BRACE>

ifStatement ::= <IF> expression block (elseStatement)?

elseStatement ::= <ELSE> (ifStatement | block)

switchStatement ::= <SWITCH> (expression)? switchBlock

switchBlock ::= <OPEN_BRACE> caseGroup <CLOSE_BRACE>

caseGroup ::= (<CASE> expressionList ":" statementList)* defaultCase

defaultCase ::= (<DEFAULT_TOKEN> ":" statementList)?

assignmentStatement ::= identifier <EQUALS> expression

increment ::= (<INCREMENT> identifier | identifier <INCREMENT>)

decrement ::= (<DECREMENT> identifier | identifier <DECREMENT>)

forStatement ::= <FOR> forClause block

forClause ::= ((assignmentStatement <SEMICOLON> expression <SEMICOLON>

(assignmentStatement | increment | decrement)) | expression)

operand ::= ((identifier | integerConstant | realConstant | booleanConstant | interpretedString) |

<OPEN_PAREN> expression <CLOSE_PAREN>)

relationalOperators ::= <EQUAL_EQUAL>

| <NOT_EQUAL>

| <LESS_THAN>

| <LESS_EQUAL>

| <GREATER_THAN>

| <GREATER_EQUAL>

expression ::= term (<EQUAL_EQUAL> term | <NOT_EQUAL> term | <LESS_THAN> term | <LESS_EQUAL> term | <GREATER_THAN> term | <GREATER_EQUAL> term | <PLUS> term | <MINUS> term)*

term ::= operand (<MULTIPLY> operand | <DIVIDE> operand | <MODULO> operand | <AMPERSAND> operand | <PIPE> operand | <CARET> operand)*

expressionList ::= expression (<COMMA> expression)*

printStatement ::= "Println" <OPEN_PAREN> expression <CLOSE_PAREN>

identifier ::= <IDENTIFIER>

integerConstant ::= <INTEGER_NUMBER>

voidConstant ::= <VOID>

booleanConstant ::= <BOOL_CONSTANT>

realConstant ::= <REAL_NUMBER>

interpretedString ::= <INTERPRETED_STRING>

type ::= <INT>

| <FLOAT>

| <STRING>

| <BOOL>

returnType ::= (<INT> | <FLOAT> | <STRING> | <BOOL> | <VOID>)

processArrayElementType ::= java code

handleError ::= java code

Team Wibbly Wobbles

CS153 Compiler Project

Jasmin Code for Various Constructs

1. Print statement

```
package main

Println("Hello World");
```

```
.class public Input
.super java/lang/Object

.field private static _runTimer LRunTimer;


.method public <init>()V

        aload_0
        invokenonvirtual        java/lang/Object/
<init>()V
        return

.limit locals 1
.limit stack 1
.end method

.method public static main([Ljava/lang/String;)V

    new  RunTimer
    dup
    invokenonvirtual     RunTimer/<init>()V
    putstatic      Input/_runTimer LRunTimer;

    getstatic java/lang/System/out Ljava/io/
PrintStream;
    ldc "Hello World"
    invokevirtual java/io/PrintStream/println(Ljava/
lang/String;)V

    getstatic      Input/_runTimer LRunTimer;
    invokevirtual
RunTimer.printElapsedTime()V

    return

.limit locals 1
.limit stack  16
.end method
```

2. IF and FOR statement

(Fizzbuzz)

```
package main

i int;

for i = 1; i <= 100; i++ {
    Println(i);

    if i % 3 == 0 {
        Println("fizz");
    }

    if i % 5 == 0 {
        Println("buzz");
    }
}
```

```
 .....
    ldc 1
    putstatic Input/i I
label1:
    getstatic Input/i I
    i2f
    ldc 100
    i2f
    fcmpg
    ifgt label2
    getstatic java/lang/System/out Ljava/io/PrintStream;
    getstatic Input/i I
    invokevirtual java/io/PrintStream/println(I)V
    getstatic Input/i I
    ldc 3
    irem
    i2f
    ldc 0
    i2f
    fcmpg
    ifne label3
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc "fizz"
    invokevirtual java/io/PrintStream/println(Ljava/lang/
String;)V
    goto label4
label3:
label4:
    getstatic Input/i I
    ldc 5
    irem
    i2f
    ldc 0
    i2f
    fcmpg
    ifne label5
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc "buzz"
    invokevirtual java/io/PrintStream/println(Ljava/lang/
String;)V
    goto label6
label5:
label6:
    ldc 1
    getstatic Input/i I
    iadd
    putstatic Input/i I
    goto label1
label2:

    getstatic      Input/_runTimer LRunTimer;
    invokevirtual          RunTimer.printElapsedTime()V

    return
```

3. Passing by Reference

```
package main



x int;
x = 10;









func passByReference(a *int) void {

    Println(a);
    a = 5;

    Println(a);

    Println(x); // This would be 5 in Java.
However, due to the limitations of the teacher's
implementation
                // It x doesn't become 5 until it leaves
the function and unwraps the Wrap class

}









passByReference(x);
Println(x);
```

```
.class public Input
.super java/lang/Object

.field private static _runTimer LRunTimer;

.field private static x I

.method public <init>()V

            aload_0
            invokenonvirtual            java/lang/Object/
<init>()V
            return

.limit locals 1
.limit stack 1
.end method

.method private static passByReference(LIWrap;)V

    .var 0 is a LIWrap;

    getstatic java/lang/System/out Ljava/io/PrintStream;
    aload 0
    getfield IWrap/value I
    invokevirtual java/io/PrintStream/println(I)V
    aload 0
    ldc 5
    putfield IWrap/value I
    getstatic java/lang/System/out Ljava/io/PrintStream;
    aload 0
    getfield IWrap/value I
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    getstatic Input/x I
    invokevirtual java/io/PrintStream/println(I)V

    return

.limit locals 1
.limit stack  16
.end method

.method public static main([Ljava/lang/String;)V

    new    RunTimer
    dup
    invokenonvirtual        RunTimer/<init>()V
    putstatic        Input/_runTimer LRunTimer;

    ldc 10
    putstatic Input/x I
    new IWrap
    dup
    getstatic Input/x I
```

```
dup
    astore 1
    invokestatic  Input/passByReference(LIWrap;)V
    aload 1
    getfield IWrap/value I
    putstatic Input/x I
    getstatic java/lang/System/out Ljava/io/
PrintStream;
    getstatic Input/x I
    invokevirtual java/io/PrintStream/println(I)V

    getstatic        Input/_runTimer LRunTimer;
    invokevirtual  RunTimer.printElapsedTime()V

    return

.limit locals 2
.limit stack  16
.end method
```