Oreofe Solarin
Email: ons8@case.edu
Course: CSDS 337 - Compiler Design
Instructor: Dr. Vipin Chaudhary

Programming Homework - 1
ID: 3637932
Term: Spring 2024
Due Date: 7^{th} February, 2024

Number of hours delay for this Problem Set: Put hours here
Cumulative number of hours delay so far: Put hours here

I discussed this homework with: Put names here

Problem 1: 70%
Implement a **for** loop in the predictive parser given to you. The loop should follow the syntax of

```
for ( statement ; boolean expression ; statement ) { Statements }
```

Implement the for loop to be robust but with proper functionality. For example, the second expression should always be type Boolean. If the syntax is violated, an appropriate error should be thrown.

An example is shown below:

```
for (i = 0; i < 10; i = i + 1) {
// do something
}
```

Include two sample codes, one showing the correctness of the grammar and the second showing an error with incorrect use of the grammar.

1. The correct code would be the implementation of **bubblesort** in your grammar, named, PG1-1-correct.t and the corresponding intermediate code output would be PG1-1-correct.i. Show the working steps of the output program with an example that uses four distinct unsorted integers as input. Clearly state all assumptions you make.
2. The incorrect code would be named: PG1-1-incorrect.t

Solution: **PG1-1-correct.t**

```
    {
   int[4] a; int i; int j; int temp;
   a[0] = 4;
   a[1] = 0;
   a[2] = 2;
   a[4] = 3;
   for (i = 0; i < 5; i = i + 1) {
       for (j = 0; j < 5  - 1 ; j = j + 1) {
           if (a[j] > a[j + 1]) {
               temp = a[j];
               a[j] = a[j + 1];
               a[j + 1] = temp;
           }
       }
   }
}
```

PG1-1-correct.i

```
L1: t1 = 0 * 4
a [ t1 ] = 4
L3: t2 = 1 * 4
a [ t2 ] = 0
L4: t3 = 2 * 4
a [ t3 ] = 2
L5: t4 = 4 * 4
a [ t4 ] = 3
L6: i = 0
L7: iffalse i < 5 goto L2
L8: j = 0
L10: t5 = 5 - 1
iffalse j < t5 goto L9
L11: t6 = j * 4
t7 = a [ t6 ]
t8 = j + 1
t9 = t8 * 4
t10 = a [ t9 ]
iffalse t7 > t10 goto L12
L13: t11 = j * 4
temp = a [ t11 ]
L14: t12 = j * 4
t13 = j + 1
t14 = t13 * 4
t15 = a [ t14 ]
a [ t12 ] = t15
L15: t16 = j + 1
t17 = t16 * 4
a [ t17 ] = temp
L12: j = j + 1
goto L10
L9: i = i + 1
goto L7
L2:
```

PG1-1-incorrect.t

```
{
   int[4] a; int i; int j; int temp;
    a[0] = 4;
    a[1] = 0;
    a[2] = 2;
    a[3] = 6;
    for i = 0; i < 4; i = i + 1 {
        for (j = 0; j < 4; j = j + 1) {
            if (a[j] > a[j + 1]) {
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
            }
        }
    }
```

```
}
```

PG1-1-incorrect.t Error Log

```
Exception in thread "main" java.lang.Error: near line 7: syntax error
    at parser.Parser.error(Parser.java:15)
    at parser.Parser.match(Parser.java:19)
    at parser.Parser.stmt(Parser.java:110)
    at parser.Parser.stmts(Parser.java:62)
    at parser.Parser.stmts(Parser.java:62)
    at parser.Parser.stmts(Parser.java:62)
    at parser.Parser.stmts(Parser.java:62)
    at parser.Parser.stmts(Parser.java:62)
    at parser.Parser.block(Parser.java:30)
    at parser.Parser.program(Parser.java:23)
    at main.Main.main(Main.java:9)
```

Working Steps:

1. Initialization:
   (a) Initialize variables `a`, `i`, `j`, `temp` of type integer array, and integers respectively.
   (b) Assign values to array `a`: `a[0] = 4`, `a[1] = 0`, `a[2] = 2`, `a[3] = 6`.
   (c) Begin the outer loop: `for (i = 0; i < 4; i = i + 1)`.
2. Outer Loop Execution:
   (a) Start with `i = 0`.
   (b) Check if `i < 4` is true.
   (c) Begin the inner loop: `for (j = 0; j < 4; j = j + 1)`.
3. Inner Loop Execution:
   (a) Start with `j = 0`.
   (b) Check if `j < 4` is true.
   (c) Perform comparison: `if (a[j] > a[j + 1])`.
   (d) Swap elements if the condition is true.
   (e) Increment `j`.
   (f) Repeat until `j < 4` is false.
4. Increment `i`.
5. Repeat steps 2-4 until `i < 4` is false.
6. End of program.

Assumptions Made:

1. The array `a` is intended to be of size 4, but the incorrect code accesses `a[3]` which is out of bounds, resulting in undefined behavior.
2. The inner loop condition should be `j < 4 - 1` to prevent accessing `a[j + 1]` out of bounds.

Problem 2: 30%
Add a binary operator '**div**' to the list of operators. This operator has higher precedence than multiplication and division but lower than parentheses. This operator is the integer division operator, i.e., the output is the integer quotient.

Include two sample codes, one showing the correctness of the grammar and the second showing an error with incorrect use of the grammar due to this operator.

1. The correct code would be named PG1-2-correct.t and the corresponding intermediate code output would be PG1-2-correct.i. The sample code should illustrate the working precedence order. Show the working steps of the output program and clearly state all assumptions you make.
2. The incorrect code would be named: PG1-2-incorrect.t

Solution: **PG1-2-correct.t**

```
{
  int a; int b; int ans;
  a = 12;
  b = 4;

  ans = a div b;
}
```

PG1-2-correct.i

```
L1: a = 12
L3: b = 4
L4: ans = a / b
L2:
```

PG1-2-incorrect.t

```
{
  int a; float b; int ans;
  a = 12;
  b = 4;

  ans = a div b;
}
```

PG1-2-incorrect.t Error Log

```
Exception in thread "main" java.lang.Error: near line 6: Type error: float is not compatible wit
        at parser.Parser.error(Parser.java:15)
        at parser.Parser.term(Parser.java:203)
        at parser.Parser.expr(Parser.java:184)
        at parser.Parser.rel(Parser.java:174)
        at parser.Parser.equality(Parser.java:166)
        at parser.Parser.join(Parser.java:158)
        at parser.Parser.bool(Parser.java:150)
        at parser.Parser.assign(Parser.java:139)
        at parser.Parser.stmt(Parser.java:128)
        at parser.Parser.stmts(Parser.java:62)
```

```
        at parser.Parser.stmts(Parser.java:62)
        at parser.Parser.stmts(Parser.java:62)
        at parser.Parser.block(Parser.java:30)
        at parser.Parser.program(Parser.java:23)
        at main.Main.main(Main.java:9)
make: *** [run] Error 1
```

Working Steps (for Correct Code):

1. Tokenization: The lexer identifies tokens in the input code, including identifiers (`a`, `b`, `ans`), literals (`12`, `4`), and the `div` operator.
2. Parsing: The parser processes the tokens according to grammar rules, recognizing expressions and statements, and constructs the abstract syntax tree (AST).
3. Type Checking: During parsing, the type compatibility of operands for the `div` operator is verified. In this case, since both `a` and `b` are of type `int`, no type errors occur.
4. Code Generation: Intermediate code is generated based on the parsed AST, where the `div` operator is translated into appropriate instructions for integer division.

Assumptions Made:

1. The `div` operator is only compatible with operands of type `int`.
2. The grammar rules for expressions have been appropriately modified to handle the `div` operator and enforce type compatibility.

Problem 3: % included in previous problems
Include the complete grammar that includes **for** loop and operator '**div**' added to the original grammar provided in the sample.

Solution:

1.
```
        case Tag.FOR:
             For fornode = new For();
             savedStmt = Stmt.Enclosing; Stmt.Enclosing = fornode;
             match(Tag.FOR);
             match('(');
             s1 = stmt();
             x = bool();
             match(';');
             s2 = assign(false);
             match(')');
             s = stmt();
             fornode.init(s1, x, s2, s);
             Stmt.Enclosing = savedStmt;  // reset Stmt.Enclosing
             return fornode;
```

```
    public void gen(int b, int a) {
            after = a;
            int label1 = newlabel();
            stmt1.gen(b, label1);
            emitlabel(label1);
            expr.jumping(0, a);
```

```
        int label2 = newlabel();
        emitlabel(label2);
        int label3 = newlabel();
        stmt3.gen(label2, label3);
        emitlabel(label3);
        stmt2.gen(label3, a);
        emit("goto L" + label1);
      }
```

2.
```
Expr term() throws IOException {
Expr x = unary();
while(look.tag == '*' || look.tag == '/' || look.tag == Tag.DIV) {
  Token tok;
  if (look.tag == Tag.DIV) {
     if (x.type != Type.Int) {
        error("Type error: " + x.type + " is not compatible with div operator");
     }
      tok = new Token('/');
     move();
        Expr x2 = unary();
        if (x2.type != Type.Int) {
           error("Type error: " + x2.type + " is not compatible with div operator");
        }
     x = new Arith(tok, x, x2);
  } else {
      tok = look;
     move();
     x = new Arith(tok, x, unary());
  }

}
return x;
}
```

Deliverables: A zip file containing

- **All folders from the dragon-front-source file you were given, plus your changes**
- **A text file containing a simple list of all changes you made to the source files, clearly labeled**
- **A pdf file with relevant answers to questions in this document.**
- **A text file marked readme that contains:**
  - **Full name and Case ID**
  - **(not required) any special notes about your implementation the grader should be aware of**

Tip: An app such as Notepad++ can be used to create .t files which you can use to test your program in the test folder.