Benjamin Scholar
Email: scholar@case.edu
Course: CSDS 337 - Compiler Design
Instructor: Dr. Vipin Chaudhary

Programming Homework - 1
ID: ??
Term: Spring 2023
Due Date: 8^{th} February, 2023

Number of hours delay for this Problem Set: 0
Cumulative number of hours delay so far: 0

I discussed this homework with: Zachary Baldwin, Matthias Portzel

---

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

```
for ( stmt; boolean expression ; stmt) { 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:

a Correct input

```
{
    int i; int n; int temp;
    int[4] data;

    n = 4;
    data[0] = 44;
    data[1] = 3;
    data[2] = 16;
    data[3] = 4;

    for(;;) {
        bool swapped;
        swapped = false;

        for(i = 0 ; i < n - 1; i = i + 1) {
            if(data[i] > data[i + 1]) {
                swapped = true;
                temp = data[i];
                data[i] = data[i + 1];
                data[i + 1] = temp;
            }
        }

        if(!swapped) break;
    }

}
```

Correct output:

```
L1:       n = 4
L3:       t1 = 0 * 4
          data [ t1 ] = 44
L4:       t2 = 1 * 4
          data [ t2 ] = 3
L5:       t3 = 2 * 4
          data [ t3 ] = 16
L6:       t4 = 3 * 4
          data [ t4 ] = 4
L7:L8:L9:         swapped = false
L11:      i = 0
L13:      t5 = n - 1
          iffalse i < t5 goto L12
L14:      t6 = i * 4
          t7 = data [ t6 ]
          t8 = i + 1
          t9 = t8 * 4
          t10 = data [ t9 ]
          iffalse t7 > t10 goto L15
L16:      swapped = true
L17:      t11 = i * 4
```

```
        temp = data [ t11 ]
L18:    t12 = i * 4
        t13 = i + 1
        t14 = t13 * 4
        t15 = data [ t14 ]
        data [ t12 ] = t15
L19:    t16 = i + 1
        t17 = t16 * 4
        data [ t17 ] = temp
L15:    i = i + 1
L20:    goto L13
L12:    if swapped goto L10
L21:    goto L2
L10:    goto L8
L2:
```

Walkthrough: I will walk through each step of the output assembly program here. The input program is a bubble-sort implementation. This implementation sorts 4 integers utilizing 2 for loops (to show that the for loop statement is functional after modifications).

Here we can see the data being setup. We can see the offsets being calculated to write data to the data array.

```
L1:     n = 4
L3:     t1 = 0 * 4
        data [ t1 ] = 44
L4:     t2 = 1 * 4
        data [ t2 ] = 3
L5:     t3 = 2 * 4
        data [ t3 ] = 16
L6:     t4 = 3 * 4
        data [ t4 ] = 4
```

Here we can see the outer for loop getting set up. Because there is no initial statement specified, there is no operations occuring here. It can also be seen that the local variable swapped is initialized to false.

```
L7:L8:L9:       swapped = false
```

Here we can see the initial statement for the inner loop. We are setting the iterating variable i to 0.

```
L11:    i = 0
```

Here are are calculating the loop exit condition for the inner loop. First 1 is subtracted from the value of n to determine the correct value for the exit condition comparison. If the exit condition was false, we will jump to the end of the loop.

```
L13:    t5 = n - 1
        iffalse i < t5 goto L12
```

This represents the calculation of the comparison for the if statement. This is the beginning of the statement specified for the for loop to execute.

```
L14:    t6 = i * 4
        t7 = data [ t6 ]
        t8 = i + 1
        t9 = t8 * 4
        t10 = data [ t9 ]
        iffalse t7 > t10 goto L15
```

This represents the various calculations occuring within the if statement

```
L16:    swapped = true
L17:    t11 = i * 4
        temp = data [ t11 ]
L18:    t12 = i * 4
        t13 = i + 1
        t14 = t13 * 4
        t15 = data [ t14 ]
        data [ t12 ] = t15
L19:    t16 = i + 1
        t17 = t16 * 4
        data [ t17 ] = temp
```

The loop variable i for the inner loop is incremented and then we jump back to the top of the inner loop.

```
L15:    i = i + 1
L20:    goto L13
```

Finally, if the swapped variable is false, we jump out of the outer loop and exit the program. Otherwise, we jump back to the top of the outer loop.

```
L12:    if swapped goto L10
L21:    goto L2
L10:    goto L8
L2:
```

Explanation:

It is evident that my implementation of the for loop generates code that meets the specifications of the assignment. Some design decisions made about the for loop implementation are as follows. I decided to constrain the first and third arguments to assignment statements only to decrease complexity of the grammar, and because having statements other than assignments doesn't make much sense in the declaration of a forloop. All 3 positions are optional (can be left empty, as can be seen in the correct program above.

b Incorrect input

```
{
    int i;
    int x;

    x = 0;

    for(i = 0; i + 1; i = i + 1) {
        x = x + 1;
    }

}
```

Incorrect error message:

```
PG1-1-incorrect.t
Exception in thread "main" java.lang.Error: near line 7: boolean expression required in second
 position of for loop
        at inter.Node.error(Node.java:10)
        at inter.For.init(For.java:26)
        at parser.Parser.stmt(Parser.java:95)
        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)
PG1-2-correct.t
```

Explanation:

This code does not finish parsing due to the fact that the expression in the second position of the for loop statement is not of type bool (Type.Bool). This means that the for loop can not evaluate the exit condition. The way I implemented this check mirrors the behavior of the pre-existing While and If statement types.

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:

a Correct input

```
{
    int a;
    int b;
    float c;
    float d;

    a = 5;
    b = 3;
    d = 3.5;
    c = a div -b * d;
}
```

Correct output:

```
L1:     a = 5
L3:     b = 3
L4:     d = 3.5
```

```
L5:     t1 = minus b
        t2 = a div t1
        c = t2 * d
L2:
```

Explanation:

It can be seen that previous code compiles within the modified front end. Additionally, all the operations are being executed with the correct priority. First the unary minus is run on b, then a is divided by the temporary value using the *div* operator, then the multiplication with float d occurs. This is the desired order of operations, thus the implementation of the *div* operator appears to be correct.

All numeric types are able to be operated on by the *div* operator (including floats). Differing types would need to be cast to other types in many cases. This, however, would be handled by another part of the compiler most likely, thus it is not considered in this implementation.

b Incorrect input

```
{

    bool a;
    float b;
    int c;

    a = false;
    b = 23.1;

    c = b div a;
}
```

Incorrect error message:

```
PG1-2-correct.t
PG1-2-incorrect.t
Exception in thread "main" java.lang.Error: near line 10: type error
        at inter.Node.error(Node.java:10)
        at inter.Arith.<init>(Arith.java:11)
        at parser.Parser.div(Parser.java:214)
        at parser.Parser.term(Parser.java:204)
        at parser.Parser.expr(Parser.java:196)
        at parser.Parser.rel(Parser.java:186)
        at parser.Parser.equality(Parser.java:178)
        at parser.Parser.join(Parser.java:170)
        at parser.Parser.bool(Parser.java:162)
        at parser.Parser.assign(Parser.java:149)
        at parser.Parser.assign(Parser.java:139)
        at parser.Parser.stmt(Parser.java:133)
        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: *** [Makefile:12: run] Error 1
```

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: The resulting grammar can be described as follows:

```
program  -> block
block -> { decls stmts }
decls -> decls decl | ϵ
decl -> type id;
type -> type [ num ] | basic
stmts -> stmts stmt | ϵ


stmt -> loc = bool
| if(bool) stmt
| if(bool) stmt else stmt
| while(bool) stmt
| do stmt while (bool);
| for( stmt ; bool ; stmt) stmt
| break ;
| block

loc -> loc [bool] | id
bool -> bool || join | join
join -> join && equality | equality
equality -> equality == rel | equality ≠ rel | rel
rel -> expr < expr | expr ≤ expr | expr > expr | expr ≥ expr | expr
expr -> expr + term | expr − term | term
term -> term * divise | term / divise | divise
divise -> divise div unary | unary
unary -> ! unary | − unary | factor
factor -> ( bool ) | loc | num | real | true | false
```

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.