CS347 Compilers Lab Assignment 2 (Part II)

Desh Raj (130101018)
Aneesh Dash (130101006)
Anirudh Agnihotry (130101007)

Department of Computer Science & Engineering
IIT Guwahati
India

March 10, 2016

1 Introduction

We will create a grammar for a simple language which satisfies the following requirements:

- Global declarations for both functions and variables
- Arithmetic expressions (brackets, +, -, *, /, %)
- Variable types - `int` and `bool`
- Loop construct - `while`
- Conditional statement - `if else`
- Nesting of loops and conditionals
- Type checking
- Input/Output commands - `print, read`
- Recursion

A context-free grammar G can be mathematically defined as:

$$G = (V, \sum, R, S)$$

where V = set of non-terminals,
$\sum$ = set of terminals,
R = production rules, and
S = start symbol.

We will describe each of these components to precisely describe our grammar.

2 Terminals ($\sum$)

The set of terminals consists of six types of elements:

1. alphabet = a | ... | z | A | ... | Z

2. number = 0 | ... | 9

3. arithmetic_op = (|) | + | - | * | /

4. relational_op = < | > | = | !

5. logical_op = & | |

6. whitespace (tabs or line breaks)

All lexemes of the grammar consist of combinations of these terminals. Hence, tokens of the grammar can be defined using these 4 sets.

Tokens

The token classes are defined as follows:

1. **KEYWORD = static | int | bool | break | return | print | read | if | else | while**

2. **IDENTIFIER** = alphabet (alphabet | number)*

3. **NUMCONST** = (number)$^{+}$

4. **BOOLCONST = true | false**

5. **OPERATOR** = arithmetic_op | relational_op | logical_op | $\{<=,>=,!=,==\}$

6. **DELIMITER** = $\{ // \, , \, ; \, , \, \{ \, , \, \} \}$

7. **FUNCTION = IDENTIFIER** ((**IDENTIFIER**)*)

3 Production Rules

1. $program \rightarrow declaration\_list$

2. $declaration\_list \rightarrow declaration\_list\ declaration \,|\, declaration$

3. $declaration \rightarrow variable\_dec \,|\, function\_dec \,|\, scoped\_variable\_dec$

   Productions 1,2,3 define the global declaration for variables and functions.

---

4. $variable\_dec \rightarrow type\_specifier\ variable\_dec\_list$

5. $scoped\_variable\_dec \rightarrow scoped\_type\_specifier\ variable\_dec\_list;$

6. $variable\_dec\_list \rightarrow variable\_dec\_list; var\_dec\_id \,|\, var\_dec\_id$

7. $var\_dec\_id \rightarrow$ **IDENTIFIER**

8. $scoped\_type\_specifier \rightarrow$ **static** $type\_specifier \,|\, type\_specifier$

9. $type\_specifier \rightarrow$ **int** | **bool**

   Productions 4 to 9 define the variable declaration as static or simple int or bool type.

---

10. $function\_dec \rightarrow type\_specifier$ **IDENTIFIER** $(parameters)\ statement$

11. $parameters \rightarrow parameter\_list \,|\, \epsilon$

12. $parameter\_list \rightarrow parameter\_list\,,\,parameter\_type\_list\,|\,parameter\_type\_list$

13. $parameter\_type\_list \rightarrow type\_specifier\ \mathbf{IDENTIFIER}$

    ```
    Productions 10 to 13 define function declaration with parameters.
    ```

    ---

14. $statement \rightarrow print\_stmt\,|\,expression\_stmt\,|\,compound\_stmt\,|\,conditional\_stmt\,|\,iteration\_stmt\,|\,retu$
    $\mathbf{break};$

15. $print\_stmt \rightarrow \mathbf{print}\ expression;$

16. $compound\_stmt \rightarrow \{\,local\_declaration\ stmt\_list\}$

17. $local\_declaration \rightarrow local\_declaration\ scoped\_variable\_dec\,|\,\epsilon$

18. $stmt\_list \rightarrow stmt\_list\ statement\,|\,\epsilon$

19. $expression\_stmt \rightarrow expression;\,|\,;$

20. $conditional\_stmt \rightarrow \mathbf{if}\ (simple\_expr)\ statement\,|\,\mathbf{if}\ (simple\_expr)\ statement\ \mathbf{else}\ statement$

21. $iteration\_stmt \rightarrow \mathbf{while}\ (simple\_expr)\ statement$

22. $return\_stmt \rightarrow \mathbf{return}\ ;\,|\,\mathbf{return}\ expression\ ;$

23. $break\_stmt \rightarrow \mathbf{break};$

    ```
    Productions 14 to 23 define print statement, loop construct (while), conditional
    statement (if-else) and all types of nesting and recursion.
    ```

    ---

24. $expression \rightarrow read\_expr\,|\,\mathbf{IDENTIFIER}\ =\ simple\_expr\,|\,\mathbf{IDENTIFIER}\ +\ =$
    $simple\_expr\,|\,\mathbf{IDENTIFIER}\ -\ =\ simple\_expr\,|\,\mathbf{IDENTIFIER}\ *\ =\ simple\_expr\,|$
    $\mathbf{IDENTIFIER}\ /\ =\ simple\_expr\,|\,simple\_expr$

25. $read\_expr \rightarrow \mathbf{read}\ \mathbf{IDENTIFIER};$

26. $simple\_expr \rightarrow (simple\_expr|and\_expr)\,|\,and\_expr$

27. $and\_expr \rightarrow and\_expr\ \&\ unary\_rel\_expr\,|\,unary\_rel\_expr$

28. $unary\_rel\_expr \rightarrow !\ unary\_rel\_expr\,|\,rel\_expr$

29. $rel\_expr \rightarrow sum\_expr\ relop\ sum\_expr\,|\,sum\_expr$

30. $relop \rightarrow relational\_op\,|\,\{<=,>=,==,!=\}$

31. $sum\_expr \rightarrow sum\_expr\ sumop\ term\,|\,term$

32. $sumop \rightarrow + \,|\, -$

33. $term \rightarrow term\,mulop\,unary\_expr \,|\, unary\_expr$

34. $mulop \rightarrow * \,|\, / \,|\, \%$

35. $unary\_expr \rightarrow unaryop\,unary\_expr \,|\, factor$

36. $unaryop \rightarrow * \,|\, /$

37. $factor \rightarrow \textbf{IDENTIFIER} \,|\, (expression) \,|\, call \,|\, constant$

38. $call \rightarrow \textbf{IDENTIFER}\,(args)$

39. $args \rightarrow arg\_list \,|\, \epsilon$

40. $arg\_list \rightarrow arg\_list\,expression \,|\, expression$

41. $constant \rightarrow \textbf{NUMCONST} \,|\, \textbf{true} \,|\, \textbf{false}$

```
Productions 24 to 41 define read statement and arithmetic, logical and relational
operators in expressions.
```

4 Start Symbol

As is obvious from the production rules, the start symbol is *program*.

5 Language Specifications

- There are only two data types - `bool` and `int`. Correspondingly, constants can either take integer values or be **true** or **false**.

- Pre/post increment/decrement operators are not allowed.

- AND and OR are represented by & and | respectively.

- Function overloading is not allowed. Functions must have a return type, i.e., they cannot be void. However, they may/may not return a value.

- There is only `while` statement for loop construct.

- The `print` command prints the output of the expression which follows it.

- The `read` command gets input from the command line and assigns the value to the identifier which follows it.

6 Example Code

The following program takes an integer from user input. It checks if it is a perfect square. If yes, prints 1. Otherwise, it prints the factorial of the number. The program contains all the features listed in Section 1.

```
1   static int fact = 1;
2
3   int factorial(int n)
4   {
5       if (n == 1)
6           fact = fact * 1;
7       else
8           fact = n * factorial(n-1);
9       return;
10  }
11
12  bool checkPerfectSquare (int a)
13  {
14      bool res = false;
15      int i = 2;
16      while (i < a)
17      {
18          if (i*i == a)
19              res = true;
20          i = i + 1;
21      }
22      return res;
23  }
24
25  int main()
26  {
27      int n;
28      read n;
29      if (checkPerfectSquare(n) == true)
30          print 1;
31      else
32      {
33          factorial(n);
34          print fact;
35      }
36      return;
37  }
```