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 8, 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 (Σ)

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** = {// , ; , { , }}

3 Production Rules

1. $program \rightarrow declaration\_list$
2. $declaration\_list \rightarrow declaration\_list\ \ declaration \,|\, declaration$
3. $declaration \rightarrow variable\_dec \,|\, function\_dec$

```
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. $\mathit{parameter\_type\_list} \rightarrow \mathit{type\_specifier}$ **IDENTIFIER**

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

---

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

15. $\mathit{print\_stmt} \rightarrow$ **print** $\mathit{expression}$;

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

17. $\mathit{local\_declaration} \rightarrow \mathit{local\_declaration}\; \mathit{scoped\_variable\_dec} \,|\, \epsilon$

18. $\mathit{stmt\_list} \rightarrow \mathit{stmt\_list}\; \mathit{statement} \,|\, \epsilon$

19. $\mathit{expression\_stmt} \rightarrow \mathit{expression}; \,|\, ;$

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

21. $\mathit{iteration\_stmt} \rightarrow$ **while** $(\mathit{simple\_expr})\; \mathit{statement}$

22. $\mathit{return\_stmt} \rightarrow$ **return** ; | **return** $\mathit{expression}$;

23. $\mathit{break\_stmt} \rightarrow$ **break**;

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

---

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

25. $\mathit{read\_expr} \rightarrow$ **read IDENTIFIER**;

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

27. $\mathit{and\_expr} \rightarrow \mathit{and\_expr}$ **&** $\mathit{unary\_rel\_expr} \,|\, \mathit{unary\_rel\_expr}$

28. $\mathit{unary\_rel\_expr} \rightarrow$ **!** $\mathit{unary\_rel\_expr} \,|\, \mathit{rel\_expr}$

29. $\mathit{rel\_expr} \rightarrow \mathit{sum\_expr}\; \mathit{relop}\; \mathit{sum\_expr} \,|\, \mathit{sum\_expr}$

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

31. $\mathit{sum\_expr} \rightarrow \mathit{sum\_expr}\; \mathit{sumop}\; \mathit{term} \,|\, \mathit{term}$

32. $\mathit{sumop} \rightarrow + \,|\, -$

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

34. $mulop \rightarrow * \mid / \mid \%$

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

36. $unaryop \rightarrow * \mid /$

37. $factor \rightarrow$ **IDENTIFIER** $\mid (expression) \mid call \mid constant$

38. $call \rightarrow$ **IDENTIFER** $(args)$

39. $args \rightarrow arg\_list \mid \epsilon$

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

41. $constant \rightarrow$ **NUMCONST** | **true** | **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.

```
static int fact = 1;

int factorial(int n)
{
    if (n == 1)
        fact = fact * 1;
    else
        fact = n * factorial(n-1);
    return;
}

bool checkPerfectSquare (int a)
{
    bool res = false;
    int i = 2;
    while (i < a)
    {
        if (i*i == a)
            res = true;
        i = i + 1;
    }
    return res;
}

int main()
{
    int n;
    read n;
    if (checkPerfectSquare(n) == true)
        print 1;
    else
    {
        factorial(n);
        print fact;
    }
    return;
}
```