A Grammar for the Compiler course project

Fall 2018

1 Introduction

This is the grammar for the Fall 2018 semester's Compiler course project. This language has a lot of features in common with a real-world structured programming language .And keep in mind that in this language variable names can not start with numbers , as an example “4young “ can not be used,but we can use“4young “ as a name for a function..

For the grammar that follows here are the types of the various elements by type font or symbol:

- **Keywords are in this type font.(They are bold and underlined)**
- **TOKEN CLASSES ARE IN THIS TYPE FONT.**
- *Nonterminals are in this type font.(They are italic)*
- The symbol ∑ means the empty string.

1.1 Some Token Definitions

- **White space** (a sequence of blanks and tabs) is ignored. White-space may be required to separate some tokens in order to get the scanner not to collapse them into one token. For example: in order not to consider “int x” as “intx” we have to use spaces.
- **Comments** are ignored by the scanner. Comments begin with // and run to the end of the line.
- All **keywords** are in lowercase. You need not worry about being case independent since not all lex/flex programs make that easy.To explain more about these words we have to say that they are word which are used to make the structures of different parts of the language.For instance,”**if** ”is one of these words.

2 The Grammar

1. *num*→ **0** | ... | **9**|*number*
2. *letter*→ **a** | ... | **z** | **A** | ... | **Z**| *letter*
3. *numOrletter*→*num* | *letter* | ε |*numOrletter*
4. *program* → *list*
5. *list* → *list declaration* | *declaration*
6. *declaration* →*function* | *varDeclaration*
4. *varDeclaration* → *type variableList***;**
5. *ScopedVariableDec* → *scopedSpecifier variableList***;**
6. *variableList*→ *variableList***,** *varInitialization* | *varInitialization*
7. *varInitialization* → *varForm* | *varForm***:** *(eachExpression)*
8. *varForm*→*letter numOrletter* **[***num***]** | *letter numOrletter*
9. *scopedSpecifier*→ **static** *type* | *type*
10. *type* → **Boolean** | **character** | **integer** | **char** | **bool** |***int***

11. *function* → **void** *numOrletter* **(** *parameter* **)** **{***statement***}**| *type letter numOrletter* **(** *parameter* **)** *statement*

12. *parameter* → *listOfParameters* | ε

13. *listOfParameters* → *listOfParameters ; paramTypeList* | *paramTypeList*

14. *paramTypeList* → *type paramList*

15. *paramList* → *paramList* **,** *paramId* | *paramId*

16. *localDeclarations* → *localDeclarations ScopedVariableDec* | ε

17. *paramId* →*letter numOrletter* | *letter numOrletter* **[ ]**

18. *statement* → *phrase* | *compoundPhrase* | *selectPhrase* | *iterationPhrase* | *returnPhrase* | *continue*

19.*compoundPhrase* → **{** *localDeclarations statementList* **}**

20. *statementList* → *statementList statement* | ε

21.*phrase* → *allExpression* **;** | **;**

22. *selectPhrase* → **if (** *eachExpression* **)** *ifBody* | **if (** *eachExpression* **)** **{***ifBody ifBody* **}**

23.*ifBody*→ *statement* | *statement* **other** *statement* | **;**

24. *iterationPhrase* → **till (** *eachExpression* **)** *statement*

25. returnPhrase → **comeback;** | **giveback** *allExpression* **;** | **giveBack** *numOrlette* **;**

26. continue → **continue ;**

27. allExpression → *alterable mathOp allExpression*| *alterable* **++** | *alterable***--** | *eachExpression* | *alterable mathOp alterable*

28.*mathOp* → **=** | **+=** | **-=** | ***=** | **/=**

29. *eachExpression* → *eachExpression logicOp eachExpression* | *eachExpression logicOp eachExpression* | *eachExpression logicOp* **then** *eachExpression* | *logicOp eachExpression* | *relExpression* | *eachExpression logicOp* **else** *eachExpression*

30. *relExpression* → *mathEXP compareType mathEXP* | *mathEXP*

31. *compareType* →*equal* | *nonEqual*

32. equal →**<=** | **>=** | **==**

33. nonEqual→ **<** | **>** | **! =**

34. *mathEXP* → *mathEXP op mathEXP* | *unaryExpression*

35. *op* → **+** | **-** | ***** | **/** | **%**

36. *unaryExpression* → *unaryop unaryExpression* | *factor*

37. *unaryop* → **-** | ***** | **?**

38. *factor* → *inalterable* | *alterable*

39. *alterable* → *letter* numOrletter | *alterable* **[** *allExpression* **]** | *alterable* **.** *letter* numOrletter

41. inalterable → **(** *allExpression* **)** | *constant* | *letter numOrletter* **(** *args* **)**

42. args → *arguments* | ε

43. arguments → *arguments* **,** *allExpression* | all*Expression*

44. constant → **CONST | true | false**

45.logicOp → ***&&*** **| || | ~| and| or**

3 Semantic Notes

- The only numbers can only be defined as ints.
- There can only be one function with a given name. There is no function overloading.
- In if statements the **other** is associated with the most recent **if**.
- *If there are any ++ present in an assignment the variable would first increased and then the assignment would occur*.
- Expressions are evaluated in order consistent with operator associativity and precedence found in mathematics.

- Function return type is specified in the function declaration, however **void** in the declaration clears that the function does not return a value.

- Code that exits a procedure without a **comeBack** or a **giveback** returns a 0 for an function returning **int** and **false** for a function returning **bool** and a blank for a function returning **char**.

- **comeBack** and **giveBack** are the same when there is nothing after **giveback**.

- All variables, functions must be declared before use.

- *?n* generates a uniform random integer in the range 0 to |n|-1 with the sign of n attached to the result. ?5 is a random number in the range 0 to 4. ?-5 is a random number in the range 0 to 4. ?0 is undefined. ?x for array x gives a random element from the array x.

- Note that this language supports = for all types of arrays. It does not support relative testing: >=, <=, >, < for any arrays.

4 An Example

```
int 1firstFunc()
{
  int firstNum ;
  int secondNum ;
  firstNum= 5 * 10 ;
  secondNum= firstNum ++ ;
  //secondNum will be 51
Giveback secondNum
        }

void secondFunc(bool B ; int A ) {
int firstArray [5] ;
bool A1 ;
bool A2 ;
A1=  firstNum <= secondNum
A2 = B ;
if ( A1 and Then A2 )
continue ;
Other
{
int B1 ;

if (B1<5)
till(B1  != 5)
B1++;
If (B1== A)
A2  = false;
}
comeBack;
}
```