Complier Specifications

Devansh Gautam
Shovan Swain

August 23, 2019

Contents

1 REFERENCE GRAMMAR

1.1 Macro Syntax

⟨Program⟩ →VariableList FunctionList

⟨VariableList⟩ →Variable; | VariableList | ε

⟨FunctionList⟩ →Function | FunctionList | ε

⟨Variable⟩ →Datatype Identifier = Expression | Datatype Identifier | Datatype Identifier[IntegerLiteral] | Datatype Identifier[IntegerLiteral][IntegerLiteral]

⟨Expression⟩ →Literal | Identifier | Expression operator Expression | FunctionCall | (Expression) | Expression ? Expression : Expression | Identifier[Expression] | Identifier[Expression][Expression]

⟨FunctionCall⟩ →Identifier(ParameterList)

⟨ParameterList⟩ →ε | Expression | Expression, ParameterList

⟨Function⟩ →Datatype identifier (argument list) Block

⟨Block⟩ →{VariableList StatementList}

⟨ArgumentList⟩ →Variable, ArgumentList | Variable | ε

⟨StatementList⟩ →Statement StatementList | ε

⟨Statement⟩ →AssignmentStatement | FunctionCall; | ReturnStatement | forStatement | whileStatement | ifStatement | Block

⟨AssignmentStatement⟩ →Identifier = Expression;

⟨ReturnStatement⟩ →return expression; | return;

⟨forStatement⟩ →for(variable; expression; statement) Block

⟨while statement⟩ →while(expression) Block

⟨elifStatementList⟩ →elif(expression) Block elifStatementList | else Block | ε

⟨if statement⟩ →if(expression) Block elifStatementList

1.2 Micro Syntax

⟨numerals⟩ →[0-9]

⟨alphabets⟩ →[a-zA-Z]

⟨CharLiteral⟩ →‘Σ’

⟨StringLiteral⟩ →"Σ*"

⟨BoolLiteral⟩ →true | false

⟨IntegerLiteral⟩ →[+−]?⟨numerals⟩⟨numerals⟩*

⟨Literal⟩ →CharLiteral | StringLiteral | BoolLiteral | IntegerLiteral

⟨Identifier⟩ →⟨alphabets⟩ (⟨alphabets⟩ | ⟨numerals⟩)*

⟨Operator⟩ →+ | - | / | * | % | and | or | xor | not | == | != | > | <

⟨Datatype⟩ →int | unsigned | char | bool | void

2 SEMANTIC CHECKS

- No identifier is to be used before it is declared.
- The value assigned to an identifier must be of the same type as the identifier.
- The number and types of arguments in a method call must be the same as the number and types of the formals, i.e., the signatures must be identical.
- The integer literal while declaring an array must be positive.
- The expression used as an array index must be of the type integer.
- If a method call is used as an expression, the method must return a result.
- A return statement must not have a return value unless it appears in the body of a method that is declared to return a value.
- The `expression` in an `if`/`elif`/`while` statement should evaluate to a proper boolean

3 TYPES

The basic types in *Complier* are - `int(signed by default), unsigned, char, bool and 1D , 2D arrays.`

Arrays are zero indexed i.e. they are indexed from 0 to $N-1$ where N is the number of elements in the array. Arrays are accessed using the usual C bracketing convention.

4 VARIABLE SCOPES

In *Complier*, all variables must be declared at the start of a block of the code, and they can only be used in that block of code.

5 CONTROL STATEMENTS

Complier has the 3 basic Control Statements : `for, while` and `if` statements. All 3 control statements follow C-like syntax to avoid any post processing overheads.

5.1 if/elif/else

The `if` statement works as follows :
First, the `expression` is evaluated. If it is true, the true part of the code is executed, otherwise the else/elif branches (if any) are executed.

5.2 for

The `for` statement works as follows :
The initial parentheses encloses 3 sections. The `Variable` part initializes an integer variable whose scope is limited to the loop. This variable is used to keep track of the number of iterations the loop goes into. The `Expression` part is a terminating expression which may or may not include the indexing variable. The last part, the `statement` part specifies a certain action to be performed after each time the loop is executed. The loop is executed until the terminating expression is satisfied.

5.3 while

The `while` statement works as follows :
This, in contrast to the `while` has only one expression. The `statement list` enclosed in the brackets are executed until the expression evaluates to a false value

6 Functions

The language has support only for Call by value style parameter passing. There is also support for recursion. All functions have a fixed signature.

The program must have a function with the signature 'int main()'. This function would be called when the execution of the program starts.

7 Expression

Expressions are evaluated in the order of precedence that is followed by C in order to reduce inconsistencies and avoid mistakes. The operators also have their usual meanings to maintain consistency.

8 Input/Output

8.1 Input

read(x) can be used to read the value for the variable x from the input.

8.2 Output

print(x) can be used to print the value of the variable or the expression x to the output.