MiniC Language Manual

Name: Swetanjal Dutta
Roll Number: 20171077

31/08/2020

1 Macro Syntax Specification using Context Free Grammars

1.1 Meta Notation:

- <foo> means foo is a non terminal.
- **foo**(in bold font) means foo is a terminal i.e a token.
- $[x]$ means zero or one occurrence of x i.e x is optional. Note that ***brackets in quotes*** i.e '[' and ']' are terminals.
- x^* means zero or more occurrences of x.
- x^+ means one or more occurrences of x.
- x^+, means a comma separated list of one or more xs. Comma is a terminal.
- {} i.e large braces are used for grouping. Note that ***braces in quotes*** i.e '{' and '}' are terminals.
- | separates alternatives.
- Punctuation like round brackets, braces, semicolons and commas are terminals. Please note that they have not been written in bold in the CFG.

1.2 Production Rules:

1. <program> → <decl>$^+$
2. <decl> → <var_decl> | <method_decl>
3. <var_decl> → <type> <identifier>$^+$, ;
4. <method_decl> → {<type> | **VOID**} **ID** ([{<type> <identifier>}$^+$,]) <block>
5. <block> → '{' <var_decl>* <statement>* '}'
6. <type> → **INT** | **UINT** | **BOOL** | **CHAR** | **FILE**
7. <statement> → <assignment>$^+$, ;
   | <method_call>;
   | **IF** (<expr>) <block> [**ELSE** <block>]
   | **FOR** ([<assignment>$^+$,]; [<expr>$^+$,]; [<assignment>$^+$,]) <block>
   | **WHILE** (<expr>) <block>
   | **BREAK**;
   | **CONTINUE**;
   | <block>
   | **RETURN** [<expr>];
   | **PRINT** (<expr>);
   | **PRINTLN** (<expr>);

8. <assignment> → <identifier> **ASSIGN** <expr>
9. <method_call> → **ID** ([<expr>$^{+}$,])
10. <expr> → <expr8>
11. <expr8> → <expr8> **THEN** <expr8> **OTHERWISE** <expr8>
    | <expr7>
12. <expr7> → <expr7> **OR** <expr6> | <expr6>
13. <expr6> → <expr6> **AND** <expr5> | <expr5>
14. <expr5> → <expr5> **EQ** <expr4> | <expr5> **NE** <expr4> | <expr4>
15. <expr4> → <expr4> **GT** <expr3> | <expr4> **GE** <expr3> | <expr3>
16. <expr3> → <expr3> **LT** <expr2> | <expr3> **LE** <expr2> | <expr2>
17. <expr2> → <expr2> **ADD** <expr1> | <expr2> **SUB** <expr1> | <expr1>
18. <expr1> → <expr1> **MUL** <expr0>
    | <expr1> **DIV** <expr0>
    | <expr1> **MOD** <expr0>
    | <expr0>
19. <expr0> → <identifier>
    | <literal>
    | <method_call>
    | **NOT** <expr>
    | **SUB** <expr>
    | (<expr>)
    | **READ_INT**()
    | **READ_CHAR**()
    | **READ_BOOL**()
20. <identifier> → **ID** | **ID**{'[' <expr> ']'}*
21. <literal> → **INT_LIT** | **FLOAT_LIT** | **CHAR_LIT** | **STRING_LIT** | <bool_lit>
22. <bool_lit> → **TRUE** | **FALSE**
23. <arithmetic_op> → **ADD** | **SUB** | **MUL** | **DIV** | **MOD**
24. <relational_op> → **LT** | **GT** | **LE** | **GE**
25. <conditional_op> → **AND** | **OR**
26. <equality_op> → **EQ** | **NE**

1.3 Start Symbol:

- program

2 Micro Syntax Specification using Regular Expressions

2.1 Meta Notation:

- Token Type → Lexeme
- [x] matches exactly one occurrence of regular expression(x).

2.2 Rules:

1. FALSE → false
2. TRUE → true
3. NOT → !
4. NEGATE → ~
5. VOID → void
6. INT → int
7. FILE → FILE
8. UNINT → uint
9. CHAR → char
10. BOOL → bool
11. THEN → ?
12. OTHERWISE → :
13. FOR → for
14. WHILE → while
15. IF → if
16. ELSE → else
17. BREAK → break
18. CONTINUE → continue
19. RETURN → return
20. ADD → +
21. SUB → -
22. MUL → *
23. DIV → /
24. MOD → %
25. LT → <
26. GT → >
27. LE → <=
28. GE → >=
29. AND → &&
30. OR → ||
31. EQ → ==
32. NE → !=
33. ASSIGN → =
34. PRINT → print
35. PRINTLN → println

36. READ_INT → read_int
37. READ_CHAR → read_char
38. READ_BOOL → read_bool
39. , → ,
40. ; → ;
41. (→ (
42.) →)
43. { → {
44. } → }
45. INT_LIT → [0-9][0-9]*
46. FLOAT_LIT → [0-9][0-9]*(.[0-9][0-9]*)?
47. CHAR_LIT → '[a-zA-Z0-9 _.,;]' | '\[nt]'
48. ID → [a-zA-Z_][a-zA-Z0-9_]*
49. STRING_LIT → "[a-zA-Z0-9 _.,;\]*"

3 Lexical Considerations

1. All keywords and identifiers are case sensitive.
2. Keywords are reserved words. Identifiers cannot have the same name as any of the keywords.
3. White space may appear between lexical tokens.
4. Keywords and identifiers must be separated by white spaces.
5. The longest sequence of matching characters forms a token. For example, **intlr** is considered to be an identifier and not parsed as the keyword **int** followed by identifier **lr**.

4 Semantic Checks

1. Type Checking: The MiniC Language supports many data types that can appear within expressions. It is essential to check the compatibility of applying various operators on different types of data. For example, we cannot apply addition operator on bool data types. Like wise an expression returning a bool data type can be assigned to variables with type bool only. It cannot be assigned to say a variable with type int. Further more at certain places, an expression of a particular datatype is expected. For example, the <expr> within **if** expects a boolean. Another example being that of array indexes, which expect expressions to have a value of int datatype which is positive or zero. All these checks are essential during compilation so that program can be executed properly.
2. No identifier can be used before it is declared.
3. The program must contain a **main()** method from where execution of the program begins. This function should have no parameters.
4. The expression in a return statement must have the same type as the declared result type of the enclosing method definition.
5. All **break** and **continue** statements must be contained within the body of for/while loops.
6. The number and types of arguments in a method call must be the same as the number and types of the formals in the function signature.
7. If a method call is used as an expression, the method must return a result.