TKOM – Specyfikacja

Zadanie 6: Interpreter własnego języka z wbudowanym typem macierzy dwuwymiarowej
Paweł Ostaszewski
Nr albumu: 273888

1. Struktura kompilatora

Kompilator będzie się składał z modułu obsługującego plik źródłowy, z modułu skanującego (leksera), z modułu analizującego składnię (parsera), z modułu analizującego semantykę kodu, z modułu obsługującego błędy kompilacji oraz z modułu głównego koordynującego pracę wszystkich pozostałych modułów. Moduł obsługujący plik źródłowy będzie zawierał wskaźniki na obecną linię oraz znak w kodzie źródłowym oraz będzie przechowywał całą obecnie analizowaną linię kodu źródłowego w buforze. Moduł parsujący będzie uruchamiał moduł skanujący do pobrania kolejnego atomu leksykalnego (tokenu) z pliku źródłowego za każdym razem gdy token jest potrzebny. Wszystkie atomy leksykalne opisane są w punkcie 3. Atomy leksykalne. Moduł analizujący semantykę będą sprawdzać czy strumień atomów leksykalnych z kodu źródłowego jest zgodny ze składnią języka opisaną w punkcie 2. Składnia języka. Moduł obsługujący błędy będzie zawierał całą listę błędów kompilacji ze wszystkich etapów kompilacji oraz wskaźniki na miejsca w kodzie źródłowym wywołujące błędy kompilacji.

2. Składnia języka (EBNF)

```
Program = StatementList ;
BlockStatement = '{' StatementList '}' ;
StatementList = { Statement } ;
Statement = SimpleStatement | StructuredStatement |
DeclarationStatement ;
DeclarationStatement = Type Identifier ( FunctionDeclaration |
VariableDeclarations )
VariableDeclarations = ( AssignmentWithoutVariable | ',' )
{ ',' Identifier [ AssignmentWithoutVariable ] } ';' ;
FunctionDeclaration = '(' Arguments ')' BlockStatement ;
Arguments = [ Argument { ',' Argument } ] ;
Argument = Type Variable ;
SimpleStatement = [ ReadStatement | WriteStatement |
ReturnStatement | IdentifierStatement ] ';' ;
IdentifierStatement = Identifier ( AssignmentWithoutVariable |
FunctionCallStatementWithoutIdentifier ) ;
Assignment = Variable '=' Expression ;
AssignmentWithoutVariable = '=' Expression ;
```

```
FunctionCallStatement = FunctionIdentifier '(' Parameters
')' ;
FunctionCallStatementWithoutIdentifier = '(' Parameters ')' ;
Parameters = [ Expression { ',' Expression } ] ;
ReturnStatement = "return" Expression ;
FunctionIdentifier = Identifier ;
ReadStatement = "read" '(' InputVariables ')' ;
InputVariables = [ Variable { ',' Variable } ] ;
WriteStatement = "write" '(' OutValues ')' ;
OutValues = Expression { ',' Expression } ;
StructuredStatement = BlockStatement | IfStatement |
WhileStatement ;
IfStatement = "if" '(' ConditionExpression ')' "then"
Instructions [ "else" Instructions ] ;
WhileStatement = "while" '(' ConditionExpression ')'
Instructions ;
Instructions = ( BlockStatement | Statement )
ConditionExpression = Expression ;
Expression = SimpleExpression | ( SimpleExpression
RelativeOperator SimpleExpression ) ;
SimpleExpression = [ Sign ] Term { AdditiveOperator Term } ;
Term = Factor { MultiplicativeOperator Factor } ;
Factor = Variable | Constant | '(' Expression ')' | "not" Factor |
FunctionCallStatement ;
Type = SimpleType | MatrixType ;
SimpleType = "integer" | "boolean" ;
MatrixType = "matrix" '[' NaturalNumber ']' '[' NaturalNumber
']' "of" SimpleType ;
Variable = Identifier ;
Identifier = Letter { Letter | Digit | '_' } ;
RelativeOperator = '==' | '<' | '>' | '<=' | '!=' | '>=' ;
Sign = '+' | '-' ;
AdditiveOperator = '+' | '-' | "or" ;
MultiplicativeOperator = '*' | '/' | '^' | '%' | "div" |
"mod" | "and" ;
IntegerConstant = IntegerNumber ;
LiteralConstant = '"' { Letter | Digit | SpecialCharacter |
WhiteSpace | "\""} '"' ;
MatrixConstant = '[' Row ']' ;
Row = '[' IntegerConstant { IntegerConstant } ']' ;
OneLineComment = "//" { Letter | Digit | Space | Tabulation |
SpecialCharacter } NewLine;
```

```
MultiLinesComment = "/*" { Letter | Digit | WhiteSpace |
SpecialCharacter } "*/" ;
Constant = IntegerConstant | LiteralConstant |
ConstantIdentifier | MatrixConstant;
ConstantIdentifier = Identifier ;
Letter = 'a' | 'A' | 'b' | 'B' | 'c' | 'C' | 'd' | 'D' | 'e' |
'E' | 'f' | 'F' | 'g' | 'G' | 'h' | 'H' | 'i' | 'I' | 'j' |
'J' | 'k' | 'K' | 'l' | 'L' | 'm' | 'M' | 'n' | 'N' | 'o' |
'O' | 'p' | 'P' | 'q' | 'Q' | 'r' | 'R' | 's' | 'S' | 't' |
'T' | 'u' | 'U' | 'v' | 'V' | 'w' | 'W' | 'x' | 'X' | 'y' |
'Y' | 'z' | 'Z' ;
NonZeroDigit = '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' |
'9' ;
Digit = '0' | NonZeroDigit ;
NaturalNumber = NonZeroDigit | NonZeroDigit { Digit } ;
IntegerNumber = [ Sign ] ( Digit | NonZeroDigit { Digit } ) ;
SpecialCharacter = '"' | ''' | '<' | '>' | '=' | ':' | ';' |
'{' | '}' | '[' | ']' | '(' | ')' | '+' | '-' | '*' | '\' | '/' |
',' | '.' | '!' | '%' | '^' | '$' | '@' ;
WhiteSpace = Space | Tabulation | NewLine ;
Space = ' ' ;
Tabulation = '\t' ;
NewLine = '\n' ;
```

3. Atomy leksykalne

1. LeftParenthesis
2. RightParenthesis
3. LeftBracket
4. RightBracket
5. LeftCurlyBrace
6. RightCurlyBrace
7. Read
8. Write
9. While
10. If
11. Then
12. Else
13. NotOperator
14. AndOperator
15. OrOperator
16. Plus
17. Minus
18. Multiplication

19. Division
20. Times
21. DivOperator
22. ModOperator
23. Matrix
24. Of
25. Identifier
26. IntegerConstant
27. LiteralConstant
28. MatrixConstant
29. LesserThanOperator
30. LesserOrEqualOperator
31. GreaterThanOperator
32. GreaterOrEqualOperator
33. NotEqualOperator
34. EquationOperator
35. Assignment
36. Semicolon
37. Colon
38. Period
39. Comma
40. At
41. Dollar
42. Percent
43. ExclamationMark
44. BackSlash
45. QuotationMark
46. Tabulation
47. NewLine
48. Space
49. OneLineComment
50. MultiLinesCommentStart
51. MultiLinesCommentEnd
52. BooleanKeyWord
53. IntegerKeyWord
54. Return
55. EOT (End Of Text)
56. Others

15 atomów o numerach 7, 8, 9, 10, 11, 12, 13, 14, 15, 21, 22, 23, 24, 52, 53 są atomami tekstowymi uzyskiwanymi w ten sam sposób co atom 25 – Identifier. Są one odróżniane od Identifier poprzez poprzez wywołanie funkcji haszującej, która daje odpowiednia wartość klucza do mapy dla jednego z 15 atomów tekstowych lub wartość nie będąca kluczem dla identyfikatora.

4. Wbudowany typ macierz dwuwymiarowej

Język będzie zawierać wbudowany typ macierzy dwuwymiarowej, która może zawierać w sobie elementy innych wbudowanych typów liczbowych takich jak integer, float. Natomiast użycie nie liczbowych typów języka takich jak string, character będzie powodowało błąd kompilacji. Na macierzach dwuwymiarowych można wykonywać operacje dodawania, odejmowania, mnożenia, odwracania macierzy, dzielenia macierzy poprzez mnożenie przez macierz odwrotną. Wykonywanie innych operacji na macierzach będzie wywoływało błąd kompilacji. Ponadto wykonywanie operacji na macierzach zawierających elementy różnych typów również będzie wywoływało błąd kompilacji.

5. Testowanie

Bedą testowane przede wszystkim kompilacje programów zawierające deklaracje macierzy dwuwymiarowych oraz operacje na macierzach dwuwymiarowych, między innymi mnożenie i dodawanie macierzy dwuwymiarowych tak jak jest to opisane w punkcie 4. Wbudowany typ macierzy dwuwymiarowej. Będzie testowana również kompilacja programów zawierające pozostałe standardowe typy i wyrażenia. Będą wykonane testy sprawdzające poprawność składni języka z punktu 2. Składnia języka. Będzie przetestowana poprawność wczytania wszystkich atomów leksykalnych z punktu 3. Atomy leksykalne. Będą przeprowadzone testy wyrażeń arytmetycznych i algebraicznych takich jak w punkcie 6. Przykłady.

6. Przykład programu

```
// Poczatek programu - komentarz jednolinijkowy

integer globalnaZmienna3 = 2384;

integer funkcja1()
{
	return 0;
}

integer zmienna1 = 0, zmienna2 = 15, zmienna3 = 3, zmienna4 = -5;

integer potega(integer x)
{
	return x*x;
```

```
}

integer main()
{
     matrix[1][5] of integer tablica = [ [ 23 12 56 -3 27 ] ] ;
     matrix[3][6] of integer macierz = [ [ 43 0 13 -51 85 3 ] [ 71
-232 6 9 -1 5 ] [ 2 -12 35 -31 7 8 ] ] ;
     matrix[2][2] of integer macierz1, macierz2, macierz3, macierz4;

     integer i = 0;

     integer liczba = 62473, liczba1 = 2, liczba2, liczba3 = -4,
liczba4 = -7; // pierwszy komentarz

     liczba2 = 0;

     // drugi komentarz

     liczba1 = (liczba2 + liczba3) * liczba4;
     liczba1 = liczba2 / liczba3;
     liczba1 = liczba2 ^ 5;
     liczba1 = potega(liczba3 * 12);

     macierz2 = [ [ 24 12 ] [ -3 7 ] ] + [ [ 4 -2 ] [ 25  7 ] ];
     macierz3 = [ [ 24  5 ] [ 3 27 ] ] * [ [ 4  5 ] [ -1 73 ] ];
     macierz1 = (macierz2 + macierz3) * macierz2;
     macierz1 = macierz2 * macierz3;

     while (i < 10)
          i = i + 1;

     if((zmienna1 == zmienna2) and (zmienna3 != zmienna4)) then
          write (globalnaZmienna3);
     else
     {
          write (i);
     }


     read ( zmienna2, zmienna3 );

     write ( zmienna1, zmienna2, zmienna3, zmienna4 );

     write (liczba, liczba1, liczba2, liczba3, liczba4);
```

```
	write ((-zmienna2 + 6) * 3);

	write (tablica);

	write (macierz);

	write (macierz2);

	write (macierz3);

	/* komentarz wielolinijkowy
	integer liczba5 = 4;
	*/

	return 0;
}
```