IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 10, Issue 6 (May. - Jun. 2013), PP 30-34
www.iosrjournals.org

Lexical and Parser tool for CBOOP program

Tanuj Tyagi[1], Akhil Saxena[2], Sunil Nishad[3],Babita Tiwari[4]
Computer Science and Engineering Department,
Jaypee University Of Engineering And Technology (JUET)
A.B. Road,P.B No. 1, Raghogarh,Dist-Guna-473226(M.P)

Abstract: *This paper addresses an approach to build lexical and parser tool for Component Based Object Oriented Program (CBOOP). In this Paper,Lexical analysis tool use for the scanning of CBOOP program. Lexical tool reads the input characters of the source program and return the tokens. Parser will generate the syntax tree of CBOOP program and check the syntax of program. The driver program, i.e., main program will open the file containing CBOOP program according to input.*
Keywords:*CBOOP,Lexical Analyzer, Tokens, Parser, Driver program*

I. Introduction

The latest programming paradigm in software engineering field is Component Based Object Oriented Programming. This paradigm has advantage of reusability over previous paradigm like object oriented programming. It also has advantage of high granularity and lower modular interdependence. It also ensures that true inheritance does not violate encapsulation as it was in case of object oriented programming. The future of software engineering lies in this programming paradigm [5].

To use any software for any novice user there is need of executable form of that software. For this purpose there arises the need a compiler. Lexical analyzer and parser isfrontend of compiler. These stages of compiler build syntactic structure of any software. These stages play important role in building executable form. This paper aims at provide an approachto achieving these stages on Component Based Object Oriented Program.

It uses previous approaches of parsing and lexical analysis and applies and modifies it accordingly to parse the Component Based Object Oriented program. It provides with an approach that allow parsing and lexical analyzer of Component Based Object Oriented Programming.

Lexical Analyzer reads the source program character by character, breaking the source program into a sequence of tokens. The various tokens are keywords,identifiers, operators, constants and punctuation symbols such as comma and parenthesis. Each token is a collection of character with well-defined meaning of the source program that is to be treated as a single unit. The Lexical analyzer analyses successive character in the source program starting from the first character not yet grouped into a token. In order to determine the next token, many characters may be searched beyond the next token. These tokens are sent to the parser for syntax analysis [7].

Consider the following expression in C++:
result=(a+b)*2;
When this expression is input to the lexical analyzer, following tokens are generated:

Token	**Type of Token**
Result	Identifier
=	Assignment Operator
(	Punctuator
A	Identifier
+	Addition Operator
B	Identifier
)	Punctuator
C	Identifier
++	Increment Operator
5	Integer Literal
;	Punctuator

Table 1

The parser uses the tokens from lexical analyzer and generates the syntax tree.In this paper, bottom-up parser is used which works on context free grammar.Parser takes input line by line and checks syntax of that statement.It takes input and compares it with context free grammar and check whether input is correct or not. Top-down

parser is not used because it is less efficient than bottom-up parser. Bottom-up parser uses right-most derivation in reverse order [2][4].
For the above expression, i.e., result=(a+b)*2; parser generates the following parse tree:

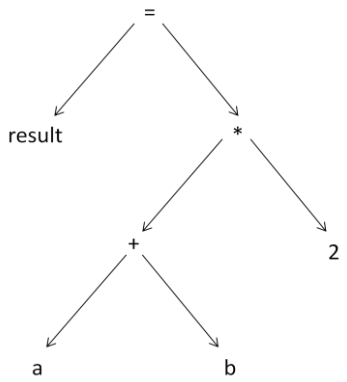
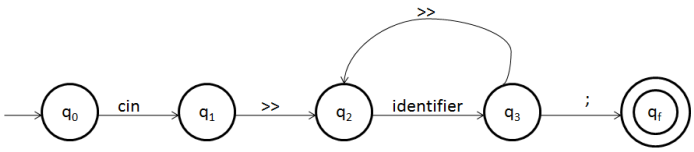


Fig. 1



cin>>a>>b;

Fig. 2 State diagram of input operation

Fig. 2 is the state diagram of input operation. It checks that input operation is syntactically correct or not. When cin is applied on the initial state q_0, state changes from $q_0$to q_1. Then, if extraction operator>>comes the state changes to q_2. Further, identifier changes the state to q_3. The extraction operator changes the state back to q_2, this terminal at q_3 state is used when more than one variable are to be declared. Finally, semi-colon (;) changes the state to the final state q_f.

II. Working OfCboop

First of all, the driver program opens component program on basis of input user enters. For example, if user entersinput as 5!.Then the driver program opens factorial component. If user enters a quadratic equation it opens quadratic component. The sole purpose of driver program is to open component program. The driver program is like "My Computer" icon in our desktop. From "My Computer" icon various drives can be opened that are present in the system. It doesn't solely perform any function or stores any data. Similarly is the driver program which just opens component program.Next, the driver program then performs lexical analysis on opened component and then parser analysis on the opened component.Now, since Component Based Object Oriented Program is collection of interdependent program. So, one component can depend on other component. So if driver program opens component that is dependent on other component then driver program first of all opens component that will be needed by input component. It then performs lexical analysis and parser on last in first out manner. Lexical analysis is performed on each component only once. On other hand, the parser performs syntax analyzer on component each time it is needed [6].

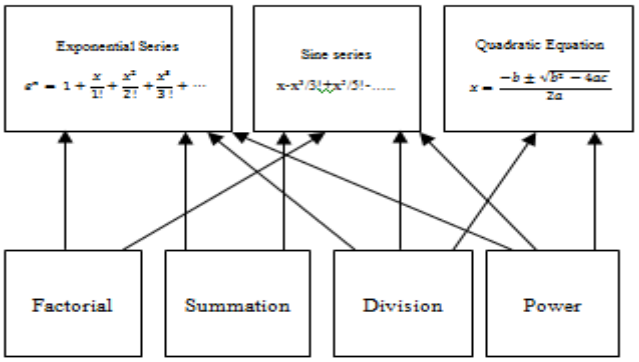


Fig. 3

Fig. 3 shows Component Based Object Oriented Program for mathematical calculator.If driver program opens exponential series component, then driver program checks which component will be needed by exponential series component. It finds out that it will need factorial, summation and division components. The driver program performs lexical analysis on each component, i.e., factorial, power, division,summation and exponential series component. Then parser analyzer performs the parsing on each component program opened. In Fig. 3, exponential series component is used to compute exponential series which is of the form:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots, -\infty < x < \infty$$

First of all driver program passes input 1 to factorial component, and then it passes 2 to factorial component and so on. So parsing is performed on factorial component with input as 1, 2 and so on. Then driver program opens power component and passes input as x and power to x. Then driver program opens division component passes output of power component and factorial component as input. Then driver program opens summation series component and passes output of division component. In this exponential series is evaluated [6].

III. Pseudo Code For Driver Program, Lexical And Parser

This section represents the pseudo code of driver program, lexical and parser.
In driver program, first of all it takes input from user. This input act as "key" which tells driver program which component to open. The driver program scans input from left to right character by character. It compares character read with cpgm which has list of all components. It then opens the desired component.
Next, the driver program performs lexical analyzer on this component by applying lexical analyzer pseudo code on it. Then it performs parser on component opened by applying parser pseudo code on it. It then applies this procedure in iterative manner.

```
Scan input Procedure driver_pgm(input,cpgm)
Begin
while(input)
{
from left to right character by character
Compare input with cpgm
then
Open component on input read
Perform lexical analyzer on component opened
Perform parsing on component open
Remove the character read from input
}
```

In lexical analyzer, component is read character by character. Next, lexical analyzer collects character into logical groupings called lexemes. First of all, lexical analyzer checks first character of lexeme. If it is character then it can be reserved word or identifiers. Then, lexical analyzer it with predefined reserved words then lexeme is reserved words otherwise it is identifier. If first character of lexeme is digit then it can be either integer literal or floating literal. If first character is neither digit nor character, it compares lexeme with predefined set of operators. This procedure is applied on each character of component.

```
Procedure lexical_analyzer
{
Read a character
switch(char)
{
case LETTER:
stoken[i]:=char
Read a character
while(char==LETTER|| char==DIGIT)
{
stoken[i]:=char
Read a character
}
ifstoken is reserved word
then
return reserved word
```

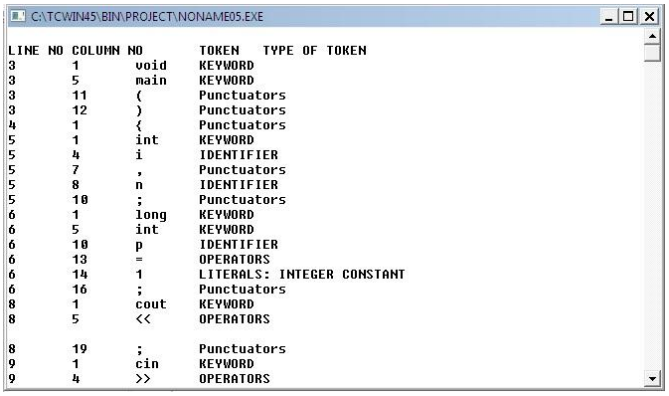
```
else
return identifier
break
case DIGIT:
stoken[i]:=char
Read a character
while(char==DIGIT)
{
stoken[i]:=char
Read a character
}
return literal
break
}
}
```

In parser, first of all define context free grammar that would be used for Component Based Object Oriented Program. Then take output of lexical analyzer as input for the parser. In this input program, each line of initial component is terminated by #. Now parser read input line by line and stores it in input array. Then apply bottom up parsing approach to generate table-driven parser. In this program take an empty array stack. Now the procedure compare this stack array with RHS of every production defined in context free grammar decided earlier. If no productions match then push a symbol of input array into stack and define this operation as "push". If it matches RHS of any production then it pops those symbols from stack that matches RHS of production and pushes LHS of that production. This procedure is iteratively applied until input array has "$". Now if input array and stack array has $ then that input is successfully parsed otherwise it is not successfully parsed.

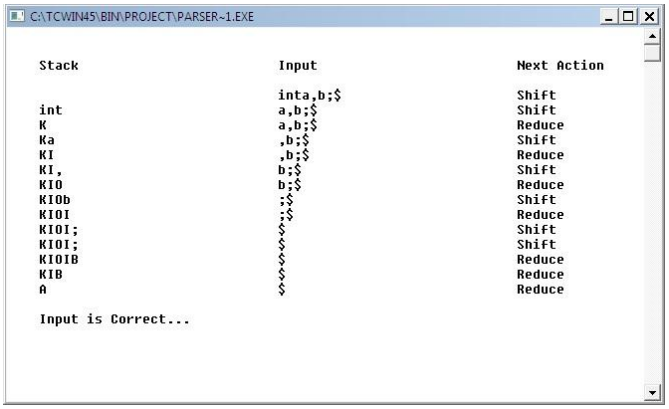
```
Procedure parser
{
do
{
Read a character
input[i]:=char
while(char!=#)
{
input[i]:=char
Read a character
}
stack[50]:=NULL
do
{
if stack not equal to RHS of productions of CFG
Then
push character of input into stack and print shift operation
else
pop all elements from stack and push LHS of that production into stack
print reduce operation
}
while(input!=NULL)
}
while(!eof())
}
```

IV. Result

First of all, on applying lexical analyzerpseudocode on component it will generate "symbol table". Symbol table is text file which list of all identifiers and integer literal and string literal used in component along with line number and column number. The symbol table for the factorial component is :



Then applying parser pseudocode on component, get table driven parser. This table has three columns: stack symbol, input and operation. The operation column can take two values: push operation and pop operation. Thistable shows how bottom up parser is applied on input and how it is reduced to start variable of context free grammar. The input and stack column shows value of input and stack array on each iteration of parser pseudo code.



V. Conclusion

This paper addresses the concept of compiler for component based object oriented program.It modifies the traditional approach of lexical and parser design to be able to be used for component based object oriented program. Italso discusses the approach to parse various components of component based object oriented program and tells how component is opened from component based object oriented program.It also tells how various interdependent components are successfully parsed. This paper designs front end of component based object oriented program compiler.

References

[1] ArvinderKaur, Kulvinder Singh, Component Selection for Component based Software engineering, International *Journal of Computer, Applications,2(1),2010,0975-8887*

[2] Miroslav D. C´ iric´ and Svetozar R. Ranc˘ic, Parsing in Different Languages,*FACTA UNIVERSITATIS,18(1),2005,299-307*

[3] João Costa Seco, Ricardo Silva and Margarida Piriquito, A Component-Based Programming Language with Dynamic Reconfiguration,*International Journal of Software and Information Systems Vol. 5,2008*

[4] Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D.Ullman, *Compilers*(Pearson,2007)

[5] AndyJuAn Wang, Kai Qian,*Component-Oriented Programming*(John Wiley & Sons, 2005)

[6] XIAOQING WU, BARRETT R. BRYANT, JEFF GRAY, MARJAN MERNIK, ALAN SPRAGUE, MURAT TANIK, *Component-Based Language Implementation with Object-Oriented Syntax and Aspect-Oriented Semantics,*The University of Alabama at Birmingham

[7] Luis Quesada, Fernando Berzal, and Francisco J.Cortijo,*A Lexical Analysis Tool with Ambiguity Support*,CITIC, University of Granada

[8] TIM A. WAGNER and SUSAN L. GRAHAM, *General Incremental Lexical Analysis*, University of California, Berkeley

[9] Oh-Cheon Kwon, Seok-Jin Yoon and Gyu-Sang Shin, Computer & Software Technology Laboratory, *ETRI(Electronics and Telecommunications Research Institute)Taejon, Korea*

[10] K. L. P. Mishra, N. CHANDRASEKARAN,*Theory of Computer Science: Automata, Languages and Computation*(Prentice-Hall,2007)

[11] O.G. Kakde,*Comiler Design*(Laxmi Publications,2005)

[12] G. SudhaSadasivam**,** *Component Based Technology(*Wiley India,2008)

[13] Laura Rimell and Stephen Clark**,** *Adapting a Lexicalized-Grammar Parser to Contrasting Domains***,**Oxford University Computing LaboratoryWolfson Building, Parks RoadOxford, OX1 3QD, UK