Parser Pseudocode

Data Structure:

inputStack: stack of characters to be scanned.

parseTokens: stack of token object returned by scanner

token object: an object that has attributes- type and value.

Global Variables:

String buffer: stbuff

Indent: counter that tracks the indentation for XML like output

Parser parses the entire file to a string buffer. There are functions below that are component of grammar that uses recursive descent parse approach. They rely on one another to completely.

```
//input: text file
//output: XML like tree formed by parsing the text file
//perform the scan on the text file and call parse on the output of scan file and output XML tree
Main:
        ask user for the input file name
        read the whole text file and put it in a single string
        break the string into stack of characters: inputStack
        string buffer: stbuff
        IF inputStack not empty
                parseTokens = Scan(inputStack)
                Parse(parseTokens, stbuff)
                IF stbuff doesn't contain "error":
                        Print stbuff
                ELSE:
                        Print "error"

//input: parseTokens, stbuff
//output: none
//data: integer indent
//parses through the stack of token object and updates the string buffer.
Parse:
        Indent += 0
        create_xml("<program>", stbuff, indent)
        Program(parseTokens, stbuff, indent)
        Indent -= 1
        create_xml("<\program>", stbuff, indent)

//input: stubuff, indent, parseTokens
//output: none
```

```
//data: tok_top = top of the parseTokens
Program:
        Tok_top := top(parseTokens)
        Case tok_top of type
                Id, read, write, $$:
                        Stmt_list(parseTokens, stbuff, indent)
                        Match("$$", parseTokens, stbuff, indent)
        ELSE
                Add "error" to stbuff

//input: stbuff, indent, parseTokens,
//output: none
//data: tok_top = top of the parseTokens
Stmt_list:
        Tok_top := top(parseTokens)
        create_xml("<stmt_list>", stbuff, indent)
        Indent += 1
        Case tok_top of type
                Id, read, write:
                        Stmt(parseTokens, stbuff, indent)
                        Stmt_list(parseTokens, stbuff, indent)
        Indent -= 1
        create_xml("</stmt_list>", stbuff, indent)

//input: stbuff, indent, parseTokens,
//output: none
//data: tok_top = top of the parseTokens
Stmt:
        create_xml("<stmt>", stbuff, indent)
        Indent += 1
        Case tok_top of type
                Id:
                        match("id", parseTokens, stbuff, indent)
                        match("assign", parseTokens, stbuff, indent)
                        expr(parseTokens, stbuff, indent)
                 Read:
                        match("read", parseTokens, stbuff, indent)
                        match("id", parseTokens, stbuff, indent)
                Write:
                        Match("write", parseTokens, stbuff, indent)
                         expr(parseTokens, stbuff, indent)
        Indent -= 1
        create_xml("<\stmt>", stbuff, indent)

//input: stbuff, indent, parseTokens
//output: none
//data: tok_top = top of the parseTokens
```

```
expr:
        create_xml(“<expr>”, stbuff, indent)
        Indent += 1
        Case tok_top of type
                Id, number, lparen:
                        Term(parseTokens, stbuff, indent)
                        Term_tail(parseTokens, stbuff, indent)
        Indent -= 1
        create_xml(“<\expr>”, stbuff, indent)

//input: stbuff, indent, parseTokens
//output: none
//data: tok_top = top of the parseTokens
Procedure Term_tail:
        create_xml(“<term_tail>”, stbuff, indent)
        Indent += 1
        Case tok_top of type
                Plus, minus:
                        Add_op(parseTokens, stbuff, indent)
                        Term(parseTokens, stbuff, indent)
                        expr(parseTokens, stbuff, indent)
                rparen, id, read, write, $$:
                        do nothing
        Indent -= 1
        create_xml(“<\term_tail>”, stbuff, indent)

//input: stbuff, indent, parseTokens
//output: none
//data: tok_top = top of the parseTokens
Procedure term:
        create_xml(“<term>”, stbuff, indent)
        Indent += 1
        Case tok_top of type
                Id, number, lparen:
                        Factor(parseTokens, stbuff, indent)
                        Factor_tail(parseTokens, stbuff, indent)
        Indent -= 1
        create_xml(“<\term”, stbuff, indent)

//input: stbuff, indent, parseTokens
//output: none
//data: tok_top = top of the parseTokens
Procedure Factor_tail:
        create_xml(“<factor_tail>”, stbuff, indent)
        Indent += 1
        Case tok_top of type
                Mult, div:
```

```
                        Mult_op(parseTokens, stbuff, indent)
                        Factor(parseTokens, stbuff, indent)
                        Factor_tail(parseTokens, stbuff, indent)
                Plus, minus, rparen, id, read, write, $$:
                        do nothing
        Indent -= 1
        create_xml("<\factor_tail>", stbuff, indent)

//input: stbuff, indent, parseTokens
//output: none
//data: tok_top = top of the parseTokens
Procedure factor:
        create_xml("<factor>", stbuff, indent)
        Indent += 1
        Case tok_top of type
                Id:
                        Match("id", parseTokens, stbuff, indent)
                number:
                        Match("number", parseTokens, stbuff, indent)
                lparen:
                        Match("lparen", parseTokens, stbuff, indent)
                        expr(parseTokens, stbuff, indent)
                        match("rparen", parseTokens, stbuff, indent)
        Indent -= 1
        create_xml("<\factor>", stbuff, indent)

//input: stbuff, indent, parseTokens
//output: none
//data: tok_top = top of the parseTokens
Procedure add_op:
        create_xml("<add_op>", stbuff, indent)
        Indent += 1
        Case tok_top of type
                plus:
                        Match("plus", parseTokens, stbuff, indent)
                minus:
                        Match("minus", parseTokens, stbuff, indent)
        Indent -= 1
        create_xml("<\add_op>", stbuff, indent)

//input: stbuff, indent, parseTokens
//output: none
//data: tok_top = top of the parseTokens
Procedure Mult_op:
        create_xml("<mult_op>", stbuff, indent)
        Indent += 1
        Case tok_top of type
```

```
                mult:
                        Match(“mult”, parseTokens, stbuff, indent)
                div:
                        Match(“div”, parseTokens, stbuff, indent)
        Indent -= 1
        create_xml(“<\mult_op>”, stbuff, indent)


//input: expected token type: string expected_tok, parseTokens, stbuff, indent
//output: none
//Data: tok_top = top of parseTokens, expected_tok = string expected token
Procedure Match:
        IF exp_tok does not equal tok_top:
                Add “error” in stbuff
        Case Tok_top is of type:
                Id, write, read, plus, minus, times, div, number, assign:
                        Indent += 1
                        Add ”<” tok_top.type “>”to stbuff
                                Indent += 1
                                Add tok_top.value to stbuff
                                Indent -= 1
                        Add “</” tok_top type “>” to stbuff
                        Indent -= 1

        Pop(parseTokens)
        Tok_top = top(parseTokens)

//input: string text, stbuff, indent
//output: none
//data: string text = the string that needs to be added to stbuff
        Integer count  = to count number of tabs to add
Procedure create_xml:
        Count := 0
        WHILE count < indent:
                Add “\t” to stbuff
                Count++
        Add text to stbuff
        Add “\n” to stbuff
```