**COMPILER CONSTRUCTION LAB**



**PROJECT REPORT**

**CAPITA LANGUAGE**

**GROUP MEMBERS:**

|  |  |
| --- | --- |
| BASIL ASLAM | (02-134191-002) |
| ALI FAWAD | (02-134191-094) |
|  |  |

Table of Contents

[**Language Specification** 3](#_Toc75409614)

[Targeted audience 3](#_Toc75409615)

[Paradigm of language 3](#_Toc75409616)

**[Components of Language](#_Toc75409618)** [4](#_Toc75409618)

[Keywords: 4](#_Toc75409619)

[Data type 5](#_Toc75409620)

[1. Num: 5](#_Toc75409621)

[2. Alpha: 5](#_Toc75409622)

[3. Alphanumeric: 5](#_Toc75409623)

[4. Decimal: 5](#_Toc75409624)

[5. Void: 5](#_Toc75409625)

[6. Bool: 5](#_Toc75409626)

[**Operators** 6](#_Toc75409627)

[Arithmetic Operators 6](#_Toc75409628)

[Relational Operators 6](#_Toc75409629)

[Logical Operators 7](#_Toc75409630)

[Constants 8](#_Toc75409631)

[Num Data Type: 8](#_Toc75409633)

[Decimal Data Type: 8](#_Toc75409634)

[Alphanumeric Data Type: 8](#_Toc75409635)

[Alpha: 9](#_Toc75409636)

[**Phases of compiler implemented** 9](#_Toc75409637)

[Lexical 9](#_Toc75409638)

[Capita Code Example 9](#_Toc75409639)

[Implementation of Lexical Using C++ 10](#_Toc75409640)

[Syntax 20](#_Toc75409641)

[CFG 21](#_Toc75409642)

[Implementation of Syntax Using C++ 22](#_Toc75409643)

[Output 33](#_Toc75409644)

**INRODUCTION**

Capita is a user friendly language in which the programmer will be able to solve complex problems without any difficulty. In capita language there will be some keywords, identifiers, punctuators and operators which will help the user to solve as much complex algorithms in an easy way. This language of computer programming will transform the coding world with its promise of more sophisticated virtual data types and objects. In this language, programmers will typically saw a codebase as composed of individual command line instructions. The identification of objects with data and functions built in led to a new way of packaging and automating code work.

Capita will prove as a hybrid that contains the functionality of the C++ programming language. This means that you have all the features that are available in C++. The user can experience universally usable modular programs, efficient and close to the machine programming and portable programs for various platforms. The large quantities of existing Capita source code can also be used in C++ programs. This language will also support the concepts of object-oriented programming. Various language elements are added to Capita, such as references, templates, and exception handling. Even though these elements of the language are important for efficient program implementation.

Further the language specifications are given below:

# Language Specification

## **Targeted audience**

The targeted audience of the Language are all the learners and coders who want to study parser and for the individuals who want to see the backend working of a compiler. Through this individual can easily see how capita treats the input given by the user and how the tokens are identified by the lexical analyzer and then given to parser. This document will help individuals to know the Language that is valid for the Parser.

## **Paradigm of language**

Paradigm can also be termed as a method to solve some problems or do some tasks. A programming paradigm is an approach to solve the problem using some programming language or also we can say it is a method to solve a problem using tools and techniques that are available to us following some approach. There are lots of programming languages that are known but all of them need to follow some strategy when they are implemented and this methodology/strategy is paradigms. Apart from varieties of programming languages, there are lots of paradigms to fulfill each and every demand.

# Capita will support four main programming paradigms: imperative, functional, procedural, and object-oriented. Whether you agree that they are valid or even useful, it will strive to make all four available and working.

# Components of Language

The components of the Language Capita will include numbers from 0 to 9, alphabets from A to Z either capital or in lowercase. There will be some function definitions for print, some mathematical functions and some other conditional and loop functions that will help in the most efficient creation. Further the components of Capita Language are discussed below:

## **Keywords:**

Keywords are ideas and subjects that define the subject matter of your article. They're the words and phrases that people type into search engines, often known as "search queries" in SEO. If you boil everything on your page all of the graphics, video, writing, and so on you'll get a far better result. The Keywords of Capita Language along with their purpose are given below:

|  |  |
| --- | --- |
| Keywords | Purpose |
| However | Looping construct |
| Else | Alternate case for an if statement |
| Begin | To start a program |
| Common | declare public members of a class |
| Personal | declare private members of a class |
| Secured | declare protected members of a class |
| Output | To display on console |
| Input | To take input from user |
| Move | execute code based on different possible values for a variable |
| Sample | a block of code in a switch statement |
| If | execute code based on the result of a test |
| Loop | looping construct |
| Elif | Used in case of nested if conditions |
| Break | break out of a loop |
| Continue | bypass iterations of a loop |
| Do-loop | looping construct |
| Return | Returns the type of function used |
| Jump | jump to a different part of the program |
| Import | Imports the header files from the library |
| True | a constant representing the boolean true value |
| False | a constant representing the boolean false value |
| fixed | create permanent storage for a variable |

## **Data type**

Data type is used in the declaration of all variables to limit the type of data that can be stored. As a result, data types are used to inform variables about the types of data they can store. When a variable is declared in Capita Language, the compiler allocates memory for it based on the data-type that it is declared with. Each sort of data necessitates a particular quantity of memory. Data types names are also considered to be reserved keyword in Capita Language. Capita Language data types are given below:

### **Num:**

Integer data types are represented by the keyword num. The range of integers is -2147483648 to 2147483647.

### **Alpha:**

For storing characters, the character data type is utilized. The character data type's keyword is alpha. Characters range from -128 to 127 or 0 to 255.

### **Alphanumeric:**

An alphanumeric is a data type that stores a sequence of characters. Although this is not a built-in type, it functions similarly in its most basic application.

### **Decimal:**

The Decimal Point data type is used to store floating point or decimal numbers with single precision. The keyword decimal is used to describe the floating point data type.

### **Void:**

The term "void" refers to something that has no worth. A valueless entity is represented by the void datatype. For functions that do not return a value, the void data type is used.

### **Bool:**

For storing Boolean or logical values, the Boolean data type is employed. True or false can be stored in a Bool variable. The bool data type's keyword is bool.

## **Operators**

### **Arithmetic Operators**

There are following arithmetic operators supported by Capita language −

|  |  |
| --- | --- |
| **Operator** | **Description** |
| + | Adds two operands |
| - | Subtracts second operand from the first |
| \* | Multiplies both operands |
| / | Divides numerator by de-numerator |
| % | Modulus Operator and remainder of after an integer division |
| ++ | [Increment operator](https://www.tutorialspoint.com/cplusplus/cpp_increment_decrement_operators.htm), increases integer value by one |
| -- | [Decrement operator](https://www.tutorialspoint.com/cplusplus/cpp_increment_decrement_operators.htm), decreases integer value by one |

### **Relational Operators**

There are following relational operators supported by Capita language

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| == | Checks if the values of two operands are equal or not, if yes then condition becomes true. | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is true. |

### **Logical Operators**

There are following logical operators supported by C++ language.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| && | Called Logical AND operator. If both the operands are non-zero, then condition becomes true. | (A && B) is false. |
| || | Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true. | (A || B) is true. |
| ! | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false. | !(A && B) is true. |

## **Constants**

### The constants for initializing in any of the data type used in Capita are validated through DFA and RE which are given below:

### **Num Data Type:**

RE: (+, -, Ɛ) (d)+



### **Decimal Data Type:**

RE: (+, -, Ɛ) (d)\*. (d)+



### **Alphanumeric Data Type:**

RE: “{(any letter except special character) + \ (special char + escape character)}”



### **Alpha:**

RE: ‘{(any letter except special character) + \ (special char + escape character)}’



# Phases of compiler implemented

## **Lexical**

The initial phase of the compiler's scan of the source code is lexical. This process can be carried out from left to right, character by character, and these characters can be grouped into tokens.

By recognising the tokens, the character stream from the source programme is sorted into meaningful sequences. It enters the relevant tickets into the symbol table and moves on to the next phase with that token.

The following are the primary functions of this phase:

* In a source code, locate the lexical units.
* Sort lexical units into categories like constants and reserved words, then enter them into various tables. It will disregard any comments in the source code.
* Identify a token that isn't in the language.

The lexical phase creates tokens which are composed of class part, value part and line no. these three attributes are generated from the given source code.

### **Capita Code Example**

begin start(){

num a;

a = 10;

end;

}

### **Implementation of Lexical Using C++**

#include <iostream>

#include <string>

#include <fstream>

#include <conio.h>

#include <stdlib.h>

using namespace std;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

string line, without\_comment;

void remove\_comments()

{without\_comment = "";

ofstream wcmnt;

for (int i = 0; i < line.length(); i++)

{if (line[i] == '~')

{int j = i;

do

{i++;

j++;} while (line[j] != '~');

}else

without\_comment += line[i];}

line = without\_comment;

wcmnt.open("without\_cmnt.txt");

if (wcmnt.is\_open())

{wcmnt << without\_comment;}

else

{cout << "\n Error With Opening File \n";

exit(0);}

wcmnt.close();}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void print\_code()

{string a = ""; // used to read from file only

fstream program;

line = "";

program.open("program.txt");

if (program.is\_open())

{cout << "\n\n\*\*\*\*\* Your Entered Program \*\*\*\*\*\n";

int i = 1;

while (!program.eof())

{getline(program, a); // reading line by line code from txt file

line += a; // append code in line variable

line += '\n'; // add '/n' at the end of every line of code

cout << "Line " << i << " >> " << a << endl;

i += 1;}}

else

{cout << "Error In Opening The File!\n";}

program.close();}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void print\_code\_without\_cmnt()

{string a = ""; // used to read from file only

fstream program;

line = "";

program.open("without\_cmnt.txt");

if (program.is\_open())

{cout << "\n\n\*\*\*\*\* Program Without Comment \*\*\*\*\*\n";

int i = 1;

while (!program.eof())

{getline(program, a); // reading line by line code from txt file

line += a; // append code in line variable

line += '\n'; // add '/n' at the end of every line of code

cout << "Line " << i << " >> " << a << endl;

i += 1;}}

else

{cout << "Error In Opening The File!\n";}

program.close();}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool is\_punctuator(char word)

{bool flag = false;

char punctuators[13] = { '{','}','[',']','(',')',':',',','?','\\','#','.',';' };

for (int i = 0; i < 13; i++)

{if (word == punctuators[i])

{return true;

break;}}

return flag;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool is\_letter(char ch)

{if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))

return true;

else

return false;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool is\_digit(char ch)

{if ((ch >= '0') && (ch <= '9'))

return true;

else

return false;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool is\_delimeter(char ch)

{if ((ch == ' ') || (ch == '\t') || (ch == '\n') || (ch == '\r'))

return true;

else

return false;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int is\_keyword(string str)

{

string keyword[18] = {"however","else","begin","common","personal","secured","output","input","move","sample","if","loop","elif","break","continue","do-loop","return","jump" };

string data\_type[6] = { "num", "decimal", "alpha", "alphanumeric" ,"void", "bool" }; string boole[2] = { "false", "true" };

for (int i = 0; i < 18; i++)

{if (keyword[i] == str)

return 1;}

for (int i = 0; i < 6; i++)

{if (data\_type[i] == str)

return 2;}

for (int i = 0; i < 2; i++)

{if (boole[i] == str)

return 3;}

return 0;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool is\_divmul(char word)

{bool flag = false;

char punctuators[3] = { '\*','/','%' };

for (int i = 0; i < 3; i++)

{if (word == punctuators[i])

{return true;

break;}}

return flag;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

bool is\_add\_sub(char word)

{bool flag = false;

char punctuators[3] = { '+','-' };

for (int i = 0; i < 12; i++)

{if (word == punctuators[i])

{return true;

break;}}

return flag;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void lexical(string str, int line\_no)

{int state = 0;

int i = 0;

string lexeme = "";

int flag = 1;

char c = '\0';

string tok = ""; // used to read from file only

ofstream token;

token.open("token.txt");

while (str[i] != '\0')

{c = str[i];

flag = 1;

switch (state)

{case 0:

{go:

if ((is\_letter(c)) || (c == '\_'))

{state = 1;}

else if (is\_digit(c))

{state = 2;}

else if (is\_punctuator(c))

{state = 0;

lexeme = c;

cout << "Token ( " << lexeme << " , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token ( " << lexeme << " , " << lexeme << " , " << line\_no << " )" << endl;

lexeme = "";

flag = 0;}

else if (is\_divmul(c))

{state = 3;}

else if (is\_add\_sub(c))

{state = 4;}

else if (c == '=')

{state = 5;}

else if ((c == '>') || (c == '<')) //ascii for > and <

{state = 6;}

else if (is\_delimeter(c))

{flag = 0;}

else if (c == 39) //ascii for " ' "

{state = 8;}

else if (c == 34) //ascii for " " "

{state = 11;}

else

{cout << lexeme << " Invalid Tokken " << endl;

token << lexeme << " Invalid Tokken " << endl;

break;}

break; }

case 1:

{if (!(is\_letter(c) || is\_digit(c) || c == '\_'))

{if (is\_keyword(lexeme))

{if (is\_keyword(lexeme) == 1)

{cout << "Token (" << " KW , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token (" << " KW , " << lexeme << " , " << line\_no << " )" << endl;

}else if (is\_keyword(lexeme) == 2)

{cout << "Token (" << " DT , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token (" << " DT , " << lexeme << " , " << line\_no << " )" << endl;

}else if (is\_keyword(lexeme) == 3)

{cout << "Token (" << " Bool , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token (" << " Bool , " << lexeme << " , " << line\_no << " )" << endl;

}}else

{cout << "Token (" << " Iden , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token (" << " Iden , " << lexeme << " , " << line\_no << " )" << endl;

}state = 0;

flag = 0;

lexeme = c;

goto go;

}break;

}case 2:

{if (!(is\_digit(c)))

{state = 0;

lexeme += c;

lexeme = lexeme.erase(lexeme.size() - 1);

cout << "Token ( Int\_const , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token ( Int\_const , " << lexeme << " , " << line\_no << " )" << endl;

lexeme = c;

flag = 0;

goto go;

}else if (is\_digit(c))

{state = 2;

}else if (c == '.')

{state = 7; //move to state 7 because its float

}break;

}case 3:

{if (c == '=')

{state = 0;

lexeme += c;

cout << "Token ( Assign\_op , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token ( Assign\_op , " << lexeme << " , " << line\_no << " )" << endl;

lexeme = "";

flag = 0;

}else

{state = 0;

cout << "Token ( Divmul , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token ( Divmul , " << lexeme << " , " << line\_no << " )" << endl;

lexeme = c;

flag = 0;

goto go;

}break;

}case 4:

{if (c == '=')

{state = 0;

lexeme += c;

cout << "Token ( Assign\_op , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token ( Assign\_op , " << lexeme << " , " << line\_no << " )" << endl;

lexeme = c;

flag = 0;

goto go;

}else

{state = 0;

cout << "Token ( Add\_sub , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token ( Add\_sub , " << lexeme << " , " << line\_no << " )" << endl;

lexeme = c;

flag = 0;

goto go;

}break;

}case 5:

{if (c == '=')

{state = 0;

lexeme += c;

cout << "Token (" << " RO , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token (" << " RO , " << lexeme << " , " << line\_no << " )" << endl;

lexeme = "";

flag = 0;

}else if (str[i - 1] == '=')

{state = 0;

lexeme = str[i - 1];

cout << "Token (" << " Assign\_Op , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token (" << " Assign\_Op , " << lexeme << " , " << line\_no << " )" << endl;

lexeme = c;

flag = 0;

goto go;

}break;

}case 6:

{if (c == '=')

{state = 0;

lexeme += c;

cout << "Token (" << " RO , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token (" << " RO , " << lexeme << " , " << line\_no << " )" << endl;

lexeme = "";

flag = 0;

}else if (c != '=')

{state = 0;

lexeme = str[i - 1];

cout << "Token (" << " RO , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token (" << " RO , " << lexeme << " , " << line\_no << " )" << endl;

lexeme = c;

flag = 0;

goto go;

}break;

}case 7:

{if (is\_digit(c))

{state = 7;

}else if (!is\_digit(c))

{state = 0;

lexeme += c;

lexeme = lexeme.erase(lexeme.size() - 1);

cout << "Token (" << " Float\_Constant , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token (" << " Float\_constant , " << lexeme << " , " << line\_no << " )" << endl;

lexeme = c;

flag = 0;

goto go;

}break;}

case 8:

{if (is\_letter(c) || is\_digit(c))

{state = 9;

}else if (c == 92) // ascii for '\'

{state = 10;

}break;

}case 9:

{if (c == 39) //ascii for ' ' '

{state = 0;

lexeme += c;

cout << "Token (" << " char\_Constant , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token (" << " char\_constant , " << lexeme << " , " << line\_no << " )" << endl;

lexeme = c;

flag = 0;

goto go;

}break;

}case 10:

{if ((c <= 9 && c >= 13) || (c == 27) || (c == 32) || (c == 127)) //ascii for special char and escape

{state = 9;}break;}case 11:

{if (c != 34) //ascii for " ' "

{state = 11;}else if (c == 34) //ascii for " ' "

{state = 0;

lexeme += c;

cout << "Token (" << " string\_Constant , " << lexeme << " , " << line\_no << " )" << endl;

token << "Token (" << " string\_constant , " << lexeme << " , " << line\_no << " )" << endl;

lexeme = c;

flag = 0;

goto go;

}}}if (flag)

lexeme += c;

i++;}token.close();}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void token()

{string code = ""; // used to read from file only

fstream wcmnt;

wcmnt.open("without\_cmnt.txt");

if (wcmnt.is\_open())

{cout << "\n\n\*\*\*\*\* Tokens \*\*\*\*\*\n";

int line\_no = 1;

while (!wcmnt.eof())

{getline(wcmnt, code); // reading line by line code from txt file

lexical(code, line\_no); //method to generate lexime

line\_no += 1;}}else

{cout << "Error In Opening The File!\n";

}wcmnt.close();

}//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main()

{start:

system("cls");

int choice;

cout << "\n 1. Print Code (with comments) ";

cout << "\n 2. Print Code (with and Without Comments) ";

cout << "\n 3. Create Tokens ";

cout << "\n 4. Exit ";

cout << "\n Enter Your Choice: ";

cin >> choice;

switch (choice)

{case 1:

{print\_code();

getch();

goto start;

}case 2:

{print\_code();

remove\_comments(); //function to remove comment

print\_code\_without\_cmnt();

getch();

goto start;

}case 3:

{remove\_comments();

token();

getch();

goto start;

}case 4:

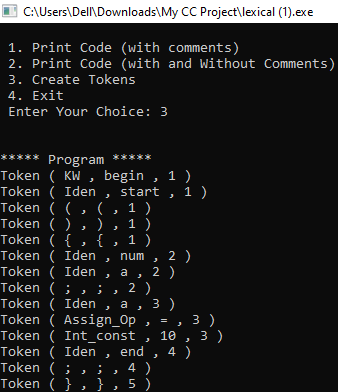
{exit(0); }

default:

{cout << "Please Select Valid Option";

goto start; }}}

**Output**



# Syntax

The goal of syntax analysis is to find structure in code. It assesses whether or not a text is formatted correctly. The primary goal of this step is to determine whether or not the programmer's source code is valid.

Syntax analysis is performed using tokens to generate a parse tree based on rules particular to the programming language. It also determines the source language's structure as well as the grammar and syntax of the language.

The following is a list of the tasks that were completed during this phase:

* Tokens should be obtained via the lexical analyzer.
* Checks whether or not the expression is syntactically proper.
* All syntax mistakes should be reported.
* Construct a parse tree, which is a hierarchical structure.

### **CFG**

The source code of Capita Language is validated through CFG’s which are stated below:

<begin> → DT start () { end } | DT start () { <m\_st> end } | DT start () { <s\_st> end }

<decl> → DT ID <init> <list>

<list> → ; | , ID <init> <list>

<init> → = <constants> | = ID <init> | ε

<constants> → num\_const | decimal\_const | alpha\_const | alphanumeric\_const

<however\_st> → however ( <Arithmeticdecl> <cond> ;) <body> <s\_st> <m\_st>

<if\_st> → if(<cond>) {<body>}

<elif> → elfi ( <cond> ){ }

<else\_st> → else { <body> }

<loop\_st> → loop (<cond>) {<body>}

<do-loop\_st> → do { <body> } loop(<cond>)

<move> → move ( <Idconst> ) { <sample> }

<sample> → sample <constants> : <body>

<Arith\_Incr\_Decr> → <INCDEC> | <Assignment> | ε

<Arithmeticdecl> → <decl> | <Assignment> | ;

<INCDEC> → ID incdec | incdec ID

<body> → ; | <decl> | <Assignment> | { <body> }

<cond> → c

<c> → <Idconst> RO

<idconst> → ID | <constants>

<assignment> → ID = ADDSUB <idconst> || ID = ADDSUB <constants> | ID <idconst>

<s\_st> → <assignment> | <decl> | <s\_st>

<m\_st> → <however> | <if\_st> <loop\_st> <do-loop\_st> <move> | <if\_st> <else\_st> | <if\_st> <elif>

## **Implementation of Syntax Using C++**

#include<iostream>

#include <string.h>

#include <fstream>

#include<limits>

#include<string>

using namespace std;

int i = 0;

string class\_part[500];

string lineNumber[500];

int decl();

int init();

int list();

int constants();

int Arithmaticdecl();

int cond();

int Arithm\_Incr\_Decr();

int body();

int INCDEC();

int Assignment();

int c();

int Idconst();

int begin();

int sample();

int move();

int m\_st();

int s\_st();

int else\_st();

int elif();

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int begin() {if (class\_part[i] == "DT") {

cout << "begin DT done\n";

i++;

if (class\_part[i] == "start") {

cout << "begin start done\n";

i++;

if (class\_part[i] == "(") {

cout << "begin ( done\n";

i++;

if (class\_part[i] == ")") {

cout << "begin ) done\n";

i++;

if (class\_part[i] == "{") {

cout << "begin{ done\n";

i++;

if (class\_part[i] == "end") {

i++;

if (class\_part[i] == ";") {

return true;

}}

else if (s\_st() || m\_st()) {

cout << "begin multi-statement done \n";

i++;

cout << "Now Here " << class\_part[i] << " " << class\_part[i + 1] << endl;

cout << "Now Here " << class\_part[i + 2] << " " << class\_part[i + 3] << endl;

cout << "Now Here " << class\_part[i + 4] << " " << class\_part[i + 5] << endl;

if (class\_part[i] == "end") {

cout << "end \n";

i++;

if (class\_part[i] == ";") {

return true;}}}}}}}}return false;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int decl()

{if (class\_part[i] == "DT")

{i++;

cout << "DT done (Declaration)\n";

if (class\_part[i] == "ID")

{i++;

cout << "ID done (Declaration)\n";

if (init())

{i++;

if (list())

{return true;}}}}return false;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int init()

{if (class\_part[i] == ";" || class\_part[i] == ",")

{cout << "in comma (initialization)";

i--;

return true;}

if (class\_part[i] == "=")

{cout << "AO done (initialization)\n";

i++;

if (constants())

{return true;}

else if (class\_part[i] == "ID")

{i++;

if (init())

{return true;}}}

else

{return false;}}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int list()

{if (class\_part[i] == ";")

{cout << "; done (list)\n";

return true;}

else if (class\_part[i] == ",")

{i++;

if (class\_part[i] == "ID")

{i++;

if (init())

{i++;

if (list())

{return true;}}}}

else

{return false;}}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int constants()

{if (class\_part[i] == "int\_const" || class\_part[i] == "float\_const" || class\_part[i] == "char\_const" || class\_part[i] == "string\_const")

{cout << "const done (constants)\n";

return true;}

else {return false;}}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int however()

{if (class\_part[i] == "however")

{cout << "however loop done (however) \n";

i++;

if (class\_part[i] == "(")

{cout << "( done. (however)\n";

i++;

if (Arithmaticdecl())

{i++;

if (cond())

{i++;

if (class\_part[i] == ";")

{cout << "; done (however)\n";

i++;

if (Arithm\_Incr\_Decr())

{i++;

if (class\_part[i] == ")")

{cout << ") done (however)\n";

i++;

if (body() || s\_st() || m\_st())

{cout << "however loop body, sst, mst done\n";

i++;

//cout << "KAHAN PHNSA? " << class\_part[i] << endl;

if (body()) {i++;

if (class\_part[i] == "}") {

return true;}}}}}}}}}}

else {return false;}}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int if\_st() {

if (class\_part[i] == "if") {

i++;

if (class\_part[i] == "(") {

cout << "if ( Done \n";

i++;

if (cond()) {

cout << "if cond() done \n";

i++;

if (class\_part[i] == ")") {

cout << "if ) done\n";

i++;

if (class\_part[i] == "{") {

cout << "if { done\n";

i++;

if (body()) {

i++;

cout << "if body() done\n";

cout << "(if) now here " << class\_part[i] << " " << class\_part[i + 1] << endl;

cout << "(if) now here " << class\_part[i] << " " << class\_part[i + 1] << endl;

return true;}}}}}}return false;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int elif() {if (class\_part[i] == "elif") {

i++;

if (class\_part[i] == "(") {

cout << "elif ( Done \n";

i++;

if (cond()) {

cout << "elif cond() done \n";

if (class\_part[i] == ")") {

cout << "elif ) done\n";

i++;

if (class\_part[i] == "{") {

cout << "elif { done\n";

if (body()) {

cout << "elif body() done\n";

i++;

if (class\_part[i] == "}") {

cout << "elif } done\n";

return true;}}}}}}}return false;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int else\_st() {

if (class\_part[i] == "else") {

i++;

cout << "else done \n";

if (class\_part[i] == "{") {

cout << "else { done \n";

if (body()) {

i++;

cout << "else body() done \n";

if (class\_part[i] == "}") {

cout << "else } done\n";

return true;}}}}return false;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int loop() {

if (class\_part[i] == "loop") {

i++;

cout << "loop done \n";

if (class\_part[i] == "(") {

i++;

cout << "loop ( done \n";

if (cond()) {

cout << "loop cond() done \n";

i++;

if (class\_part[i] == ")") {

i++;

cout << "loop ) done \n";

if (class\_part[i] == "{") {

cout << "loop { done \n";

if (body()) {

i++;

cout << "loop body done \n";

if (class\_part[i] == "}") {

cout << "loop } done \n";

return true;}}}}}}}return false;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int do\_loop() {

if (class\_part[i] == "do") {

cout << "do\_loop \n";

i++;

if (class\_part[i] == "{") {

cout << "do\_loop { done \n";

if (body()) {

cout << "do\_loop body done\n";

i++;

if (class\_part[i] == "}") {

cout << "do\_loop } done\n";

i++;

if (class\_part[i] == "do\_loop") {

cout << "do\_loop done\n";

i++;

if (class\_part[i] == "(") {

cout << "do\_loop ( done\n";

i++;

if (cond()) {

cout << "do\_loop cond done\n";

if (class\_part[i] == ")") {

cout << "do\_loop ) done\n";

return true;}

else if (class\_part[i + 1] == ")") {

cout << "do\_loop ) done\n";

return true;

}}}}}}}}return false;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int move() {

if (class\_part[i] == "move") {

cout << "move done\n";

i++;

if (class\_part[i] == "(") {

cout << "move ( done\n";

i++;

if (Idconst()) {

cout << "move idconst done\n";

i++;

if (class\_part[i] == ")") {

cout << "move ) done \n";

i++;

if (class\_part[i] == "{") {

cout << "move { done \n";

i++;

if (sample()) {

cout << "move sample done\n";

i++;

if (class\_part[i] == "}") {

cout << "move } done\n";

return true;}

else if (sample()) {

cout << "ANOTHER sample done\n";

return true;}}}}}}}return false;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int sample() {

if (class\_part[i] == "sample") {

cout << "sample done \n";

i++;

if (constants()) {

cout << "sample const done\n";

i++;

if (class\_part[i] == ":") {

cout << "sample : done \n";

i++;

if (body()) {

cout << "sample body done\n";

i++;

cout << class\_part[i];

if (class\_part[i] == "break") {

cout << "sample break done\n";

return true;}}}}}return false;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Arithm\_Incr\_Decr()

{if (INCDEC())

{return true;}

else if (Assignment())

{return true;}

else

{return false;}}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Arithmaticdecl()

{if (decl())

{return true;}

else if (Assignment())

{i++;

if (class\_part[i] == ";")

{cout << "; done (Arithmatic Declaration)\n";

return true;}}

else if (class\_part[i] == ";")

{cout << "; done (Arithmatic Declaration)\n";

return true;}

else

{return false;}}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int INCDEC()

{if (class\_part[i] == "ID")

{cout << "ID done (INCDEC)\n";

i++;

if (class\_part[i] == "incdec")

{cout << "incdec done (INCDEC)\n";

return true; }}

else if (class\_part[i] == "incdec")

{cout << "incdec done (INCDEC)\n";

if (class\_part[i] == "ID")

{cout << "ID done (INCDEC)\n";

return true; }}

else

{return false; }}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int body()

{if (class\_part[i] == ";")

{cout << "; done (body)\n";

return true; }

else if (decl()) {

cout << "BODY declaration done\n";

return true; }

else if (Assignment()) {

cout << "BODY assignment done.\n";

return true; }

else if (class\_part[i] == "{")

{cout << "{ done (body)\n";

i++;

if (class\_part[i] == "}")

{cout << "} done (body)\n";

return true; }}

else if (body()) {

return true;}

return false;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int cond()

{if (c())

{return true; }

else

{return false; }}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int c()

{if (Idconst())

{i++;

if (class\_part[i] == "RO")

{cout << "RO done (c)\n";

i++;

if (Idconst())

{return true; }}}

else {return false; }}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Idconst()

{if (class\_part[i] == "ID")

{cout << "ID done (Idconst)\n";

return true; }

else if (constants())

{return true; }

else

{return false; }}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int Assignment()

{if (class\_part[i] == "ID")

{cout << "ID done (Assignment)\n";

i++;

if (class\_part[i] == "=")

{cout << "AO done (Assignment)\n";

i++;

if (class\_part[i] == "ADDSUB") {

i++;

if (Idconst() || constants()) {

return true;}}

else if (Idconst())

{return true;}}}

else

{return false;}}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int s\_st() {if (Assignment()) {return true;}

else if (decl()) {return true;}

else if (s\_st()) {return true;}

return false;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int m\_st() {if (however()) {return true;}

else if (if\_st()) {

return true;}

else if (loop()) {

return true;}

else if (do\_loop()) {

return true;}

else if (move()) {

return true;}

else if (if\_st()) {

if (else\_st()) {

return true;}

else if (elif()) {

return true;}}

return false;}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main() {

ifstream File;

string str;

string delimeter = ",";

File.open("OUTPUT\_TOKEN.txt");

if (!File.is\_open()) {

cout << "Unable to open Token File";

exit(1);}

int count = 0;

int i = 0;

string line;

while (getline(File, line)) {

count++;}

cout << "Number of Tokens in File: " << count << endl << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

File.close();

File.open("OUTPUT\_TOKEN.txt");

string firstWord;

string lastword;

int word = 0;

while (File >> firstWord)

{class\_part[word] = firstWord;

//cout << firstWord << endl;

File.ignore(numeric\_limits<streamsize>::max(), '\n');

word++;}

File.close();

if (begin()) {cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << " SUCCESSFULLY PARSED \n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";}

else {cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\n NOT PARSED \n\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";}

return 0;}

### **Output**

