**Recursive Decent Parser for**

**Y++**

The recursive decent parser is based upon the concept of lexical analyzer. The lexical analysis is the key to develop a compiler. In this project, I’ve developed a recursive decent parser for the C++ language. The source code for the parser is written in python. This report demonstrates the concepts used in the project and also enlightens about the code.

# Introduction

LEXICAL ANALYSIS is the very first phase in the compiler designing. A Lexer takes the modified source code which is written in the form of sentences. In other words, it helps you to convert a sequence of characters into a sequence of tokens. The lexical analyzer breaks this syntax into a series of tokens. It removes any extra space or comment written in the source code.

A lexer contains tokenizer or scanner. If the lexical analyzer detects that the token is invalid, it generates an error. It reads character streams from the source code, checks for legal tokens, and pass the data to the syntax analyzer when it demands.

# Methodology

Our Goal is to make programming language which can easily be understood by Secondary School Students and it can help them to improve their problem-solving skills. Most importantly students can experience what Programming is? and How ideas can be converted into a real product? That’s why name of language we call it **Y++** which means y = y + 1 i.e.

youToday = youYesterday + 1

## Regular Expression

Names of Variables and functions can start only from letters and end with any combination of letter or digits

Let

L = a, b, c, d, e, f, …z A, B, C, D, E, F, …Z

D= 0, 1, 2, 3, 4, 5 … 9

## DATA TYPES

|  |  |
| --- | --- |
| **C-Language** | **Y++** |
|  |  |
| Int | Numbers |
| Float | Decimals |
| string | Words |

## ESCAPE SEQUENCE

|  |  |
| --- | --- |
| **Escape Sequence** | **Represents** |
|  |  |
| \n | New Line |
| \t | Horizontal Tab |
| \’ | Single Quotation |
| \’’ | Double Quotation |

## INPUT/OUTPUT

|  |  |
| --- | --- |
| **C/C++** | **Y++** |
|  |  |
| cout<<”Programming is for Everyone”; | show(“Programming is for Everyone”); |
| cout<<”Num is ”<<num; | show(“Num is”@num); |
|  |  |
| cin<<num; | putin(num); |
| cin<<x1<<x2; | putin(x1@x2); |

## CONDITIONAL STATEMENTS

IF/Else Statements is same in Y++ as in C language

|  |  |
| --- | --- |
| **C-Language** | **Y++** |
| Int num=10;  Switch(num)  {  case num%2 == 0 :  cout<<”Num is Even”<<num;  break;  case num%2 == 1 :  cout<<”Num is ODD”<<num;  break;  } | Numbers num=10;  Check(num)  {  expr num%2 == 0 :  show(“Num is Even”@num);  stop;  expr num%2 == 1 :  show(“Num is ODD”@num);  finish;  } |

## LOOP STATEMENTS

While loop

|  |  |
| --- | --- |
| **C-Language** | **Y++** |
| While( counter<1000)  {  Cout<<”Sorry Teacher I will not do it again \n”;  counter++  } | RepeatTill(1000)  {  show(“Sorry Teacher I will not do it again \n”);  } |

# Here Simply Specify integer number for repetition

Repeat(Numbers num)

{ # Statements

}

For loop

|  |  |
| --- | --- |
| **C-Language** | **Y++** |
| For(int i=0; i<10; i++)  {  //Statements  } | Repeat(Numbers i=0; i<10; i++)  {  } |

## PREPROECESSOR DIRECTIVES

|  |  |
| --- | --- |
| **C-Language** | **Y++** |
| #include<stdio.h> | Addlib<stdio.h> |

## COMMENTS

|  |  |
| --- | --- |
| **Y++** | **Represents** |
| # | Single Line |
| ## | Multiple Lines |

## FUNCTIONS

**Main Function**

|  |  |
| --- | --- |
| **C-Language** | **Y++** |
| Void main()  {  // Statements  }  int main()  {  //Statements  return 0;  } | start()  {  #Statements  }  Numbers start()  {  #Statements  send 0;  } |

# Code

The input program is defined as:

include <iostream>   
using namespace std;   
include <iostream>

using namespace std;

start ()

{

numbers has = 3;

numbers bes = 8

loop (i=0, i,<8, i++)

{

aff(has<,bes)

{

send has;

}

nelse

{

bes = has + 2;

}

check(a)

{

casee 'A':

putin>>input;

show<<abcd;

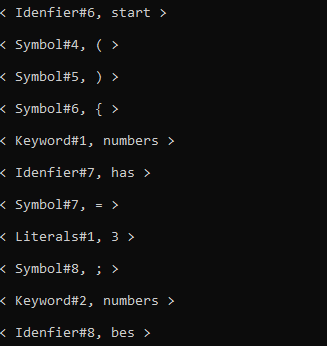
default: finish;

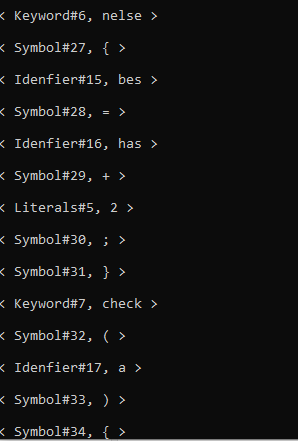
}

}

}  
 {case 'A':  
 cin>>input;  
 cout<<abcd;  
 default: break;  
 }  
}

And the output provided by the parser is:





# Conclusion

The parser can handle complex programs as well. I have kept the input simple just for the sake of understanding in the report. I have still used unique operators other than the loops. The parser can handle complex loops, conditional statements etc. as well.