SPL-1 Project Report, 2020

Mini C Compiler

Syntax Error Checking of C file

SE 305: Software Project Lab I

Submitted by

Md. Inzamam-Ul Haque Sobuz

BSSE Roll No. : 1113

Exam Roll No. : 1104

BSSE Session: 2018-19

Registration Number: 2018-2255-96

Supervised by

Nadia Nahar

Designation: Lecturer

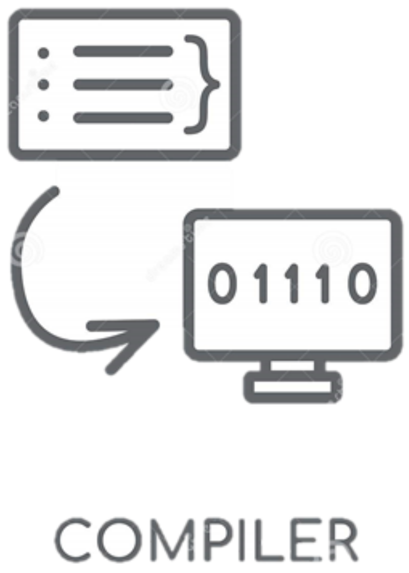
Institute of Information Technology

Institute of Information Technology

University of Dhaka

26-08-2021

Mini C Compiler



Project:	Mini C Compiler
Author:	Md. Inzamam-Ul Haque Sobuz
Submitted:	26.08.21
Supervised By:	Nadia Nahar
	Lecturer,
	Institute of Information Technology
	University of Dhaka

Supervisor's Approval: Nadia

Table of Contents

1. Introduction

A compiler is a software program that transforms high-level source code that is written by a developer in a high-level programming language into a low-level object code(binary code) in machine language, which can be understood by the processor. The process of converting high-level programming into machine language is known as compilation.

A compiler executes four major steps:

- Scanning
- Lexical Analysis
- Syntactic Analysis
- Semantic Analysis

A full Compiler is a huge project. In my SPL1, I do the basics of the first 3 steps of a compiler. So, my project **Mini C Compiler** is a simple C compiler that will be used to compile a C program. It will be able to handle basic syntactic structures. It can handle typical variables, loops, conditionals, etc. Specifically, It can handle both simple and any nested level conditional statements(if, else-if, else), loop statements(for, while, do-while), function statements, normal statements, and show output as syntax errors.

2. Objectives

An error-free program is runnable and gives output as expected in every language. A compiler helps to detect errors in a program file and then run it for getting the desired output. Those errors help a programmer to write an error-free program in the C language.

This **Mini C Compiler** is a small part of a compiler. The objective of this project is to help to detect errors in a C file(basic structure). It takes a C file as input and shows errors of taken input C file as output.

3. Scope

The scope of the **Mini C Compiler** is as follows:

- Scanning a C file as input.
- Lexical analysis of the inputted file.
- Detects all possible errors of the inputted file.
- Syntactic analysis of the inputted file.
- Show line numbers that are not syntactically correct in the C language perspective.
- Show some possible error tips for every incorrect line as a suggestion.

4. Background Study

To implement this project, some prior study was necessary:

4.1. Lexical Analysis

The first phase of the compiler is the lexical analysis. In this phase, the compiler breaks the submitted source code into meaningful elements called lexemes and generates a sequence of tokens from the lexemes. A token is an object describing a lexeme. Along with the value of the lexeme (the actual string of characters of the lexeme), it contains information such as its type (is it a keyword? or an identifier? or an operator? ...) and the position (line and/or column number) in the source code where it appears.

4.2. Syntax Analysis

Syntax analysis is performed, which involves preprocessing to determine whether the tokens created during lexical analysis are in proper order as per their usage. The correct order of a set of keywords, which can yield the desired result, is called syntax. The compiler has to check the source code to ensure syntactic accuracy.

4.3. Abstract Syntax Tree

During syntax analysis, the compiler uses the sequence of tokens generated during the lexical analysis to generate a tree-like data structure called Abstract Syntax Tree, AST for short. AST is tree representations of code. They are a fundamental part of the way a compiler works. Syntax analysis is also the phase where eventual syntax errors are detected and reported to the user in the form of informative messages.

4.4. Automata Theory

Finite automata are the mathematical model of how languages are parsed. Simple languages like regular expressions can be parsed by a state machine without memory, whereas a language with a “structured” syntax where parenthesis and blocks need matching of e.g. left parenthesis with right parenthesis requires memory in the form of a stack. Finite Automata are used in two of the three front-end phases of the compiler.

5. Challenges

There are several challenges I had to face while implementing the project. A lot of terms and the majority of the tasks were completely new to me that led me towards much confusion and complexity in implementation. Some of the challenges I faced during this implementation is enlisted below:

- Working with header files for the first time.
- Working with multiple source files.
- It was a challenge to process the input file because the written format has no specific structure.
- Tokenization was so challenging as the input file had a lot of unnecessary spaces, newlines.
- Handling a lot of error types and any level nested statement.
- First-time handling such a large codebase led to a difficult time finding errors and debugging.

6. Project Description

There are basically two parts in my project:

- Lexical Analysis / Tokenization
- Syntax Analysis

6.1. Lexical Analysis / Tokenization

In this part, the main goal is to divide into valid tokens of the full inputted file's text. It is the process of taking a file C file(such as the input code by the user given) and producing a sequence of symbols called lexical tokens, or just tokens. Also tracking some information(token type, line number, column number) of every token for further analysis.

6.2. Syntax Analysis

This part represents the full project using previously produced tokens. This part has some sub-part.

6.2.1. Grouping by keyword scope

Grouping based on some tram(Function, Loop, Conditional statement) and storing some needed information(starting line, end line, condition, type of token, etc).

6.2.2. Variable scope handle

After declaration, every variable has a scope limited. In scope, the variable is used without any violation and can't redeclare the same-named variable. But out of scope, it is not allowed to be used.

6.2.3. Error Analysis

Analysis token and token type with comparing valid formatting and finding errors. This is the most important and also challenging task in this project. A lot of error types handling is included. As a trivial programming language follows some conventions that allow it to have some distinct characteristics, my project identifies discrepancies in these conventions.

6.2.4. Error Showing with Suggestion

Error suggestion helps to find out why the error happened and also helps to fix that error. As mentioned before my project can analyze an error and then sends an error message to the user much like a conventional compiler.
For example: in C, a comparator operator must-have elements on both sides. and not following the proper syntax will produce an error.

```
Line No - 90 : "	else if( > 0 )"
 Tips : Not valid expression.
```

7. User Manual

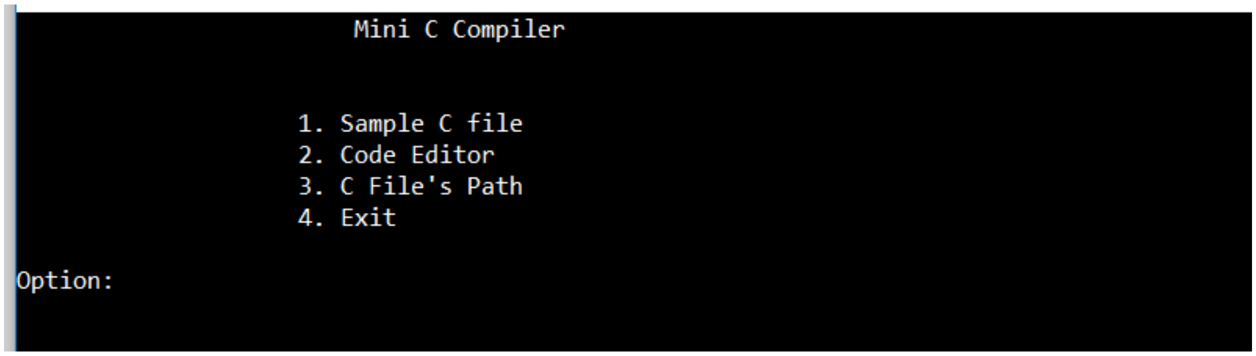
This user manual is designed to get the person using this program familiarized with the type of outputs this program gives. On the other hand, the input procedure is fairly straightforward.

7.1. For the Input

As shown in the image below the main menu will present the user with three choices for the input (Fig no: F1).

7.1.1. Sample C File

By inputting '1' the program will run on the sample C file given in the program. This is mainly an example to show the user the output format.

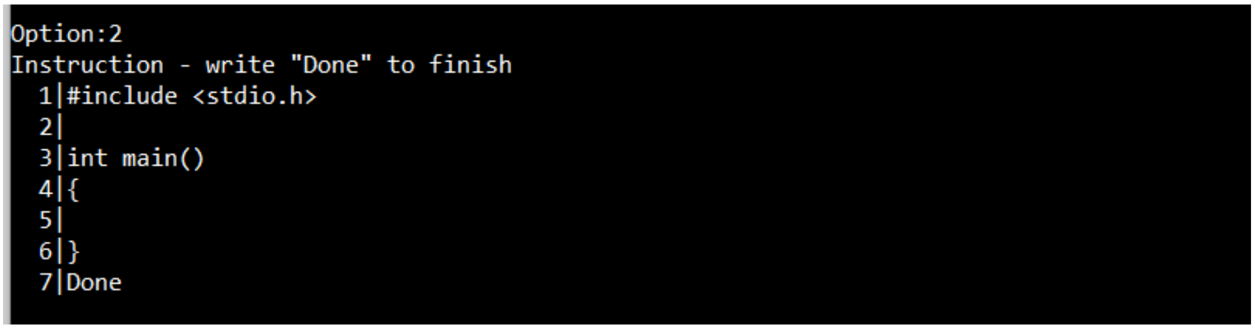


```
                        Mini C Compiler


                    1. Sample C file
                    2. Code Editor
                    3. C File's Path
                    4. Exit

Option:
```

Fig no: F1

7.1.2. Code Editor

This option(fig no: F2) presents the user with a conventional text editor where the user can write code that will be the input for the program. This is activated by inputting '2' into the terminal menu.

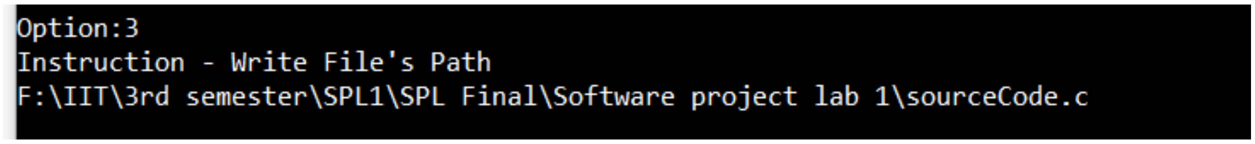


```
Option:2
Instruction - write "Done" to finish
  1|#include <stdio.h>
  2|
  3|int main()
  4|{
  5|
  6|}
  7|Done
```

Fig no: F2

7.1.3. C File Path

In this option(fig no: F3) the user has to give a file path to an already written C code. Then the program will take the file found at the given address path. This is activated by inputting '3' into the menu.



```
Option:3
Instruction - Write File's Path
F:\IIT\3rd semester\SPL1\SPL Final\Software project lab 1\sourceCode.c
```

Fig no: F3

7.2. For the Output

This part of the manual guides the user on reading the outputs given by the program. I have programmed the application to give out errors on a line-by-line basis. So when the user wants to know which type of error he/she is looking at, he/she has to look it up in this manual. I have divided the errors shown in the program into three types.

- Missing Syntax Errors
- Validation Errors
- Multiple Instances Errors

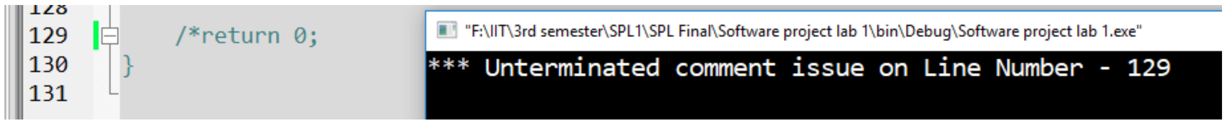
Some examples of errors are given below:

7.2.1. Missing Syntax

If the code contains an expression previously undefined or if it has missing statements that are supposed to be in the code, these parts of the error message domain are in this section.

7.2.1.1. Unterminated Comment Issue

In Fig no: F1, an unterminated comment issue happened.



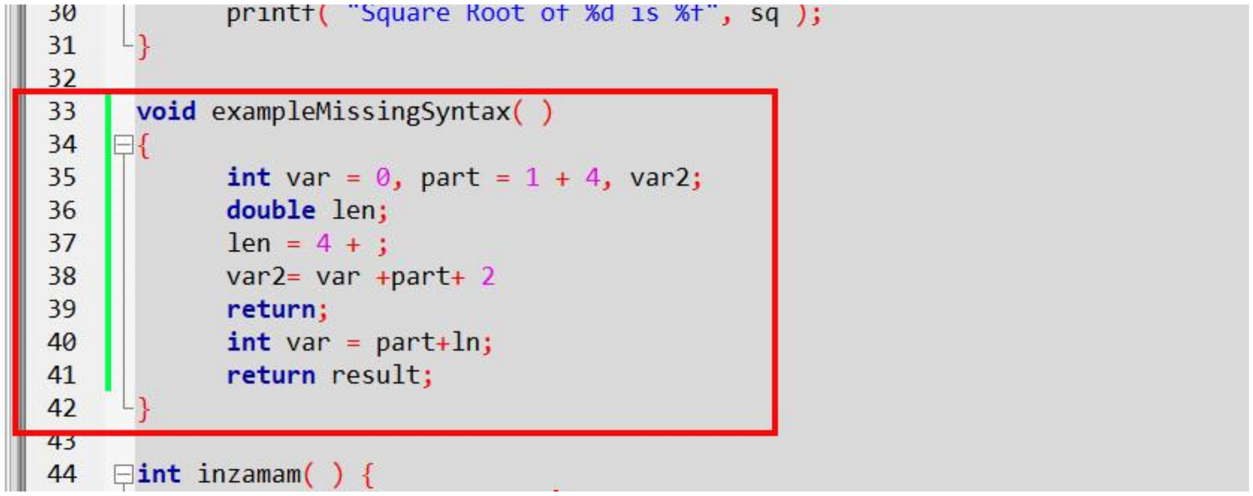
```
        /*return 0;
}
```

```
"F:\IIT\3rd semester\SPL1\SPL Final\Software project lab 1\bin\Debug\Software project lab 1.exe"
*** Unterminated comment issue on Line Number - 129
```

Fig no: F4

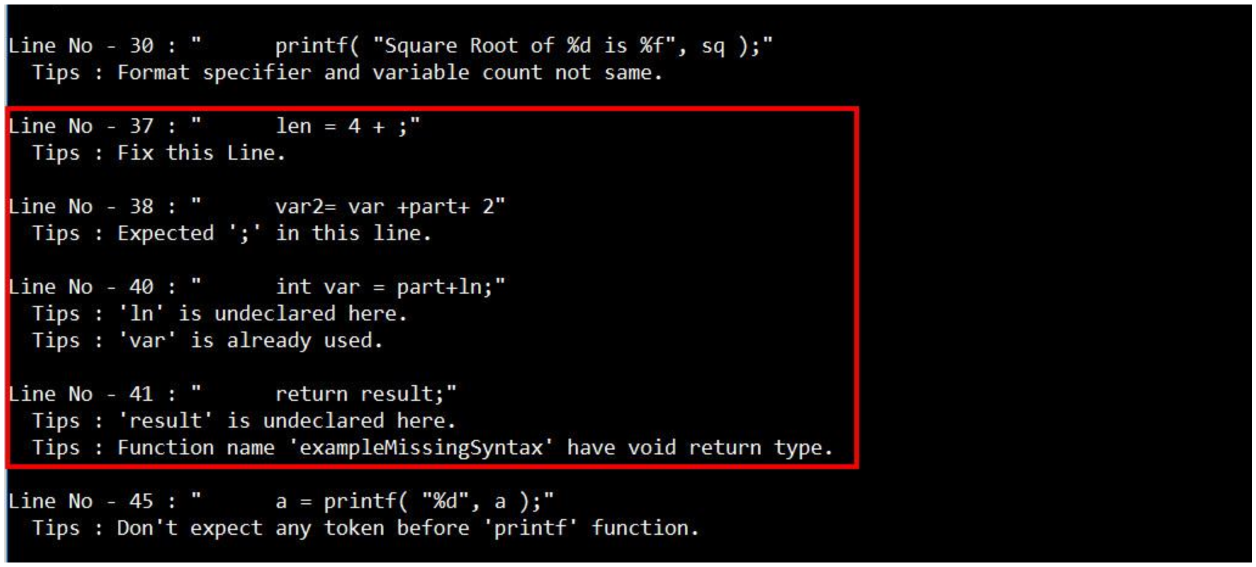
7.2.1.2. Missing Return types and Variable types

Some screenshots of various types of missing errors are shown. In (Fig no: F2), source code and (Fig no: F3), output for source code.



```
        printf( "Square Root of %d is %f", sq );
}

void exampleMissingSyntax( )
{
        int var = 0, part = 1 + 4, var2;
        double len;
        len = 4 + ;
        var2= var +part+ 2
        return;
        int var = part+ln;
        return result;
}

int inzamam( ) {
```

Fig no: F5(input part)

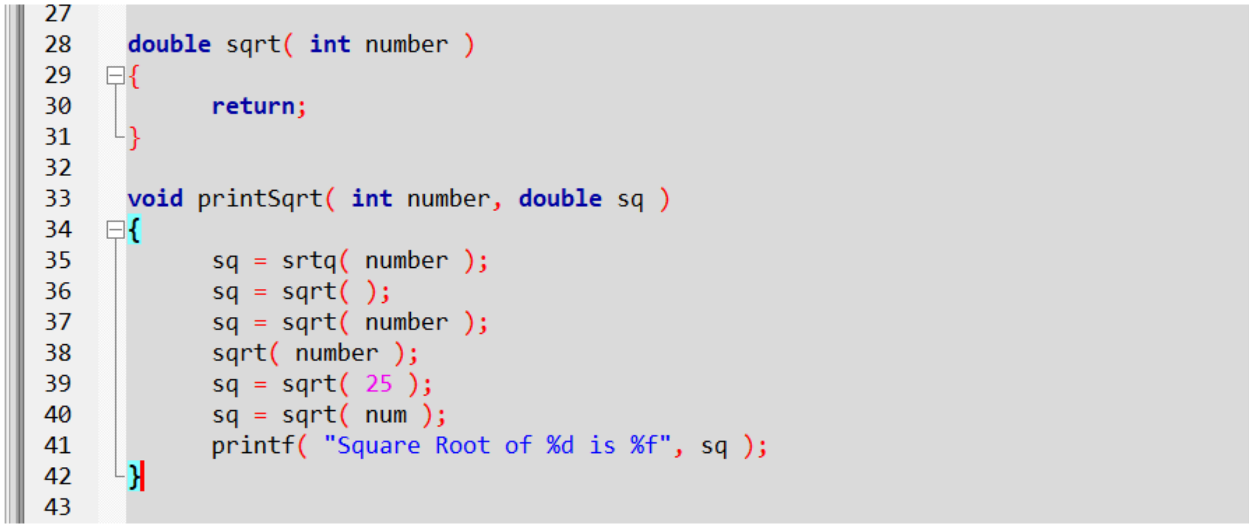


```
Line No - 30 : "       printf( "Square Root of %d is %f", sq );"
  Tips : Format specifier and variable count not same.

Line No - 37 : "       len = 4 + ;"
  Tips : Fix this Line.

Line No - 38 : "       var2= var +part+ 2"
  Tips : Expected ';' in this line.

Line No - 40 : "       int var = part+ln;"
  Tips : 'ln' is undeclared here.
  Tips : 'var' is already used.

Line No - 41 : "       return result;"
  Tips : 'result' is undeclared here.
  Tips : Function name 'exampleMissingSyntax' have void return type.

Line No - 45 : "       a = printf( "%d", a );"
  Tips : Don't expect any token before 'printf' function.
```

Fig no: F6(output part)

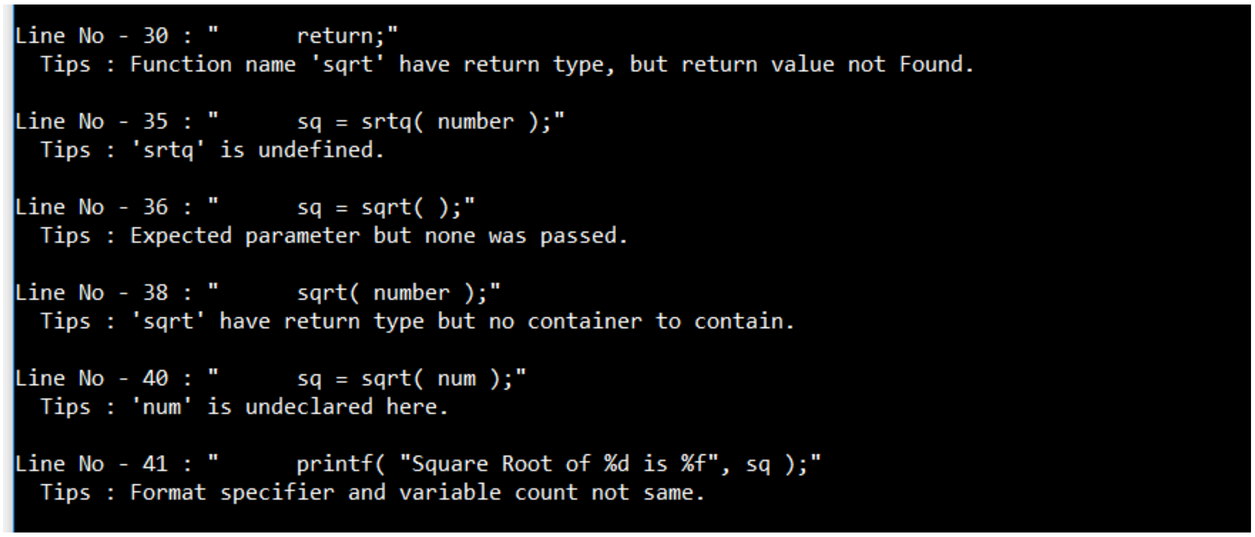
7.2.1.3. Missing Function Calling

In some cases, a user might write a method where there are arguments to be passed but when the method is called the user might forget to pass the argument. This is shown in (Fig no: F7) where the sqrt() function needs a parameter but on line 36 the parameter is missing when the function is called.



```
27
28  double sqrt( int number )
29  {
30          return;
31  }
32
33  void printSqrt( int number, double sq )
34  {
35          sq = srtq( number );
36          sq = sqrt( );
37          sq = sqrt( number );
38          sqrt( number );
39          sq = sqrt( 25 );
40          sq = sqrt( num );
41          printf( "Square Root of %d is %f", sq );
42  }
43
```

Fig no: F7(input part)



```
Line No - 30 : "      return;"
  Tips : Function name 'sqrt' have return type, but return value not Found.

Line No - 35 : "      sq = srtq( number );"
  Tips : 'srtq' is undefined.

Line No - 36 : "      sq = sqrt( );"
  Tips : Expected parameter but none was passed.

Line No - 38 : "      sqrt( number );"
  Tips : 'sqrt' have return type but no container to contain.

Line No - 40 : "      sq = sqrt( num );"
  Tips : 'num' is undeclared here.

Line No - 41 : "      printf( "Square Root of %d is %f", sq );"
  Tips : Format specifier and variable count not same.
```

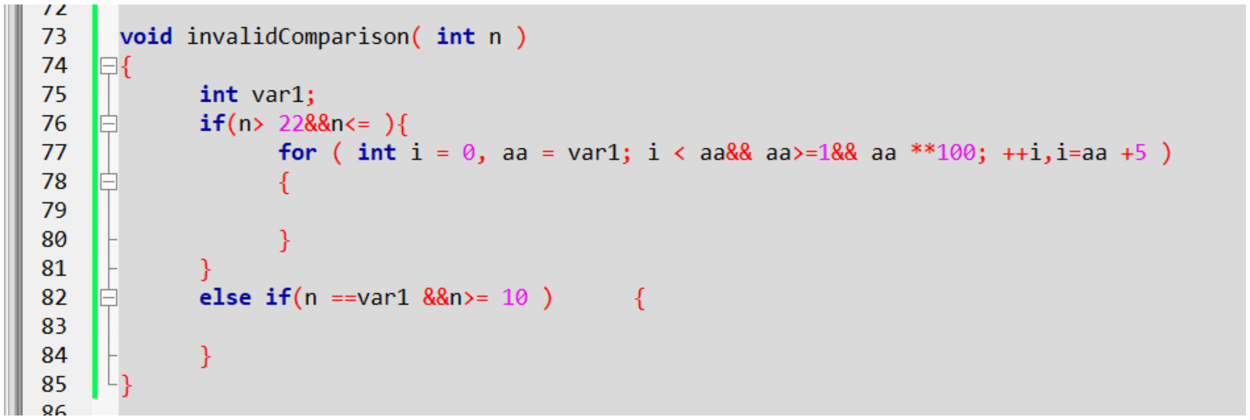
Fig no: F8(output part)

7.2.2. Validation Errors:

If a user writes code with invalid expressions such as a comparison expression without the two entities that are being compared, that is a validation error. Other types of invalid expressions a user can write are invalid variable names i.e. 1var, invalid expressions i.e. a+++++b etc.

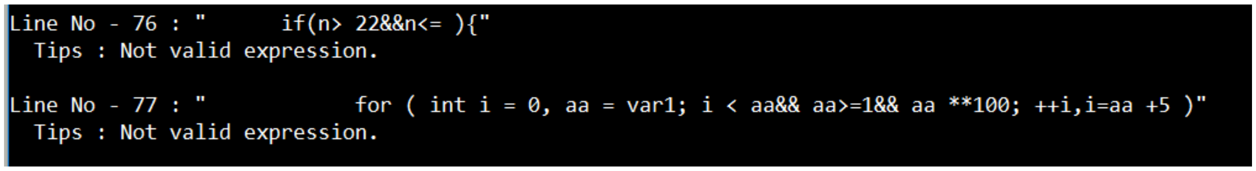
7.2.2.1. Invalid Comparisons

It can happen in any condition statement. In this particular case, condition expression validation is the biggest concern. Here, some screenshots of this type of error are given below(In Fig no: F4 and Fig no: F5).



```
void invalidComparison( int n )
{
      int var1;
      if(n> 22&&n<= ){
            for ( int i = 0, aa = var1; i < aa&& aa>=1&& aa **100; ++i,i=aa +5 )
            {

            }
      }
      else if(n ==var1 &&n>= 10 )      {

      }
}
```

Fig no: F9(input part)

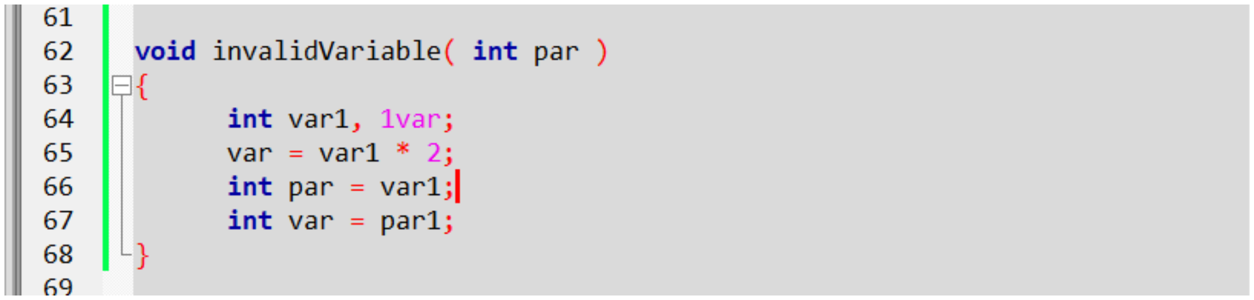


```
Line No - 76 : "        if(n> 22&&n<= ){"
  Tips : Not valid expression.

Line No - 77 : "              for ( int i = 0, aa = var1; i < aa&& aa>=1&& aa **100; ++i,i=aa +5 )"
  Tips : Not valid expression.
```

Fig no: F10(output part)

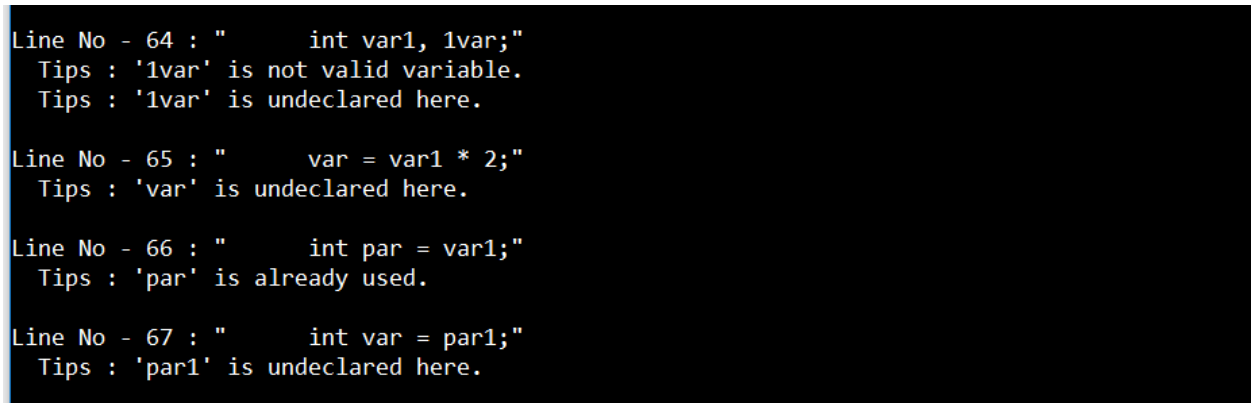
7.2.2.2. Invalid Identifier

In C language, an identifier name has a format. When these naming conventions are not followed it results in an error when parsing the code. For example, 'var1' is a valid identifier but '1var' is not. This type of error is shown in Fig no: F11 caused by the code in Fig no: F12.



```
61
62  void invalidVariable( int par )
63  {
64          int var1, 1var;
65          var = var1 * 2;
66          int par = var1;
67          int var = par1;
68  }
69
```

Fig no: F11(input part)

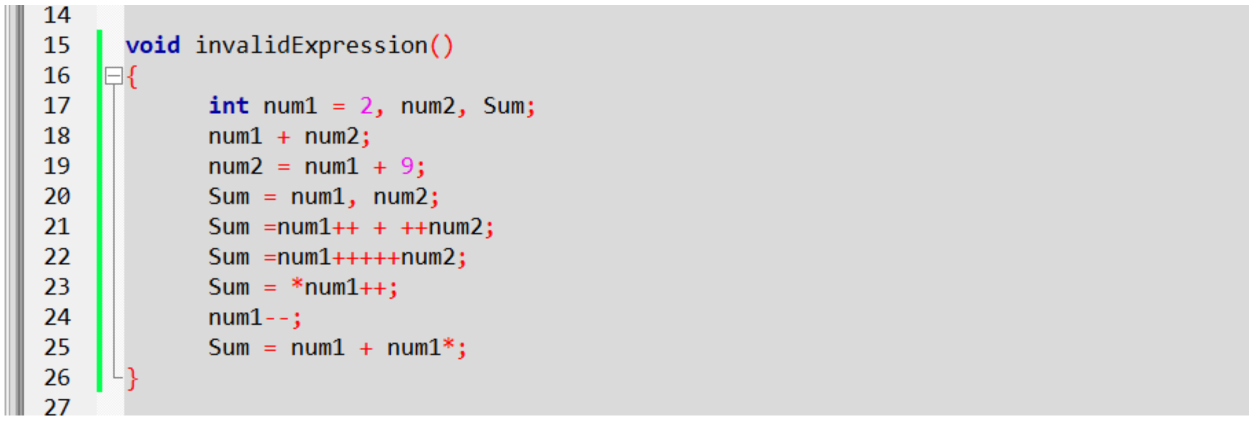


```
Line No - 64 : "        int var1, 1var;"
  Tips : '1var' is not valid variable.
  Tips : '1var' is undeclared here.

Line No - 65 : "        var = var1 * 2;"
  Tips : 'var' is undeclared here.

Line No - 66 : "        int par = var1;"
  Tips : 'par' is already used.

Line No - 67 : "        int var = par1;"
  Tips : 'par1' is undeclared here.
```

Fig no: F12(output part)

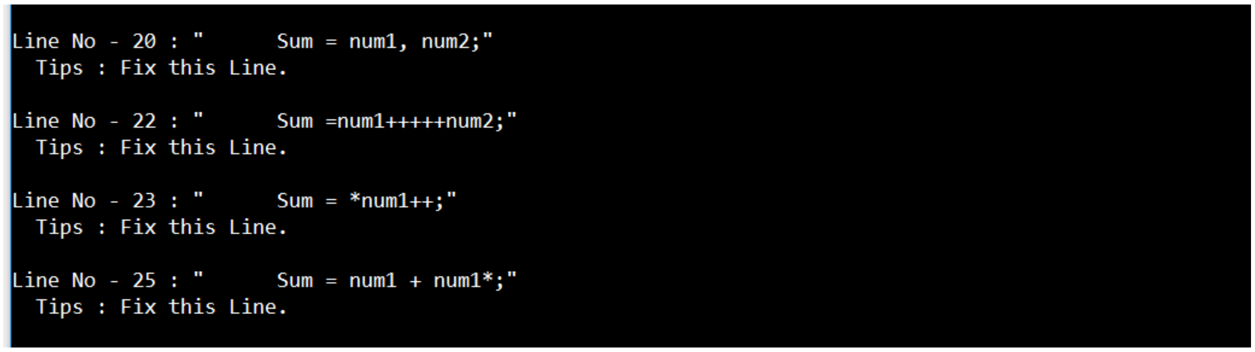
7.2.2.3. Invalid Expression

In C language, there are some conventions for writing expressions and if these conventions are not followed it will result in an error. For example in figure F13 the expression Sum=num1+++++num2 is invalid. The lack of space makes it impossible for a compiler to deconstruct the two variables from the line. The resulted error is shown in figure F14.



```
void invalidExpression()
{
        int num1 = 2, num2, Sum;
        num1 + num2;
        num2 = num1 + 9;
        Sum = num1, num2;
        Sum =num1++ + ++num2;
        Sum =num1+++++num2;
        Sum = *num1++;
        num1--;
        Sum = num1 + num1*;
}
```

Fig no: F13(input part)



```
Line No - 20 : "       Sum = num1, num2;"
  Tips : Fix this Line.

Line No - 22 : "       Sum =num1+++++num2;"
  Tips : Fix this Line.

Line No - 23 : "       Sum = *num1++;"
  Tips : Fix this Line.

Line No - 25 : "       Sum = num1 + num1*;"
  Tips : Fix this Line.
```

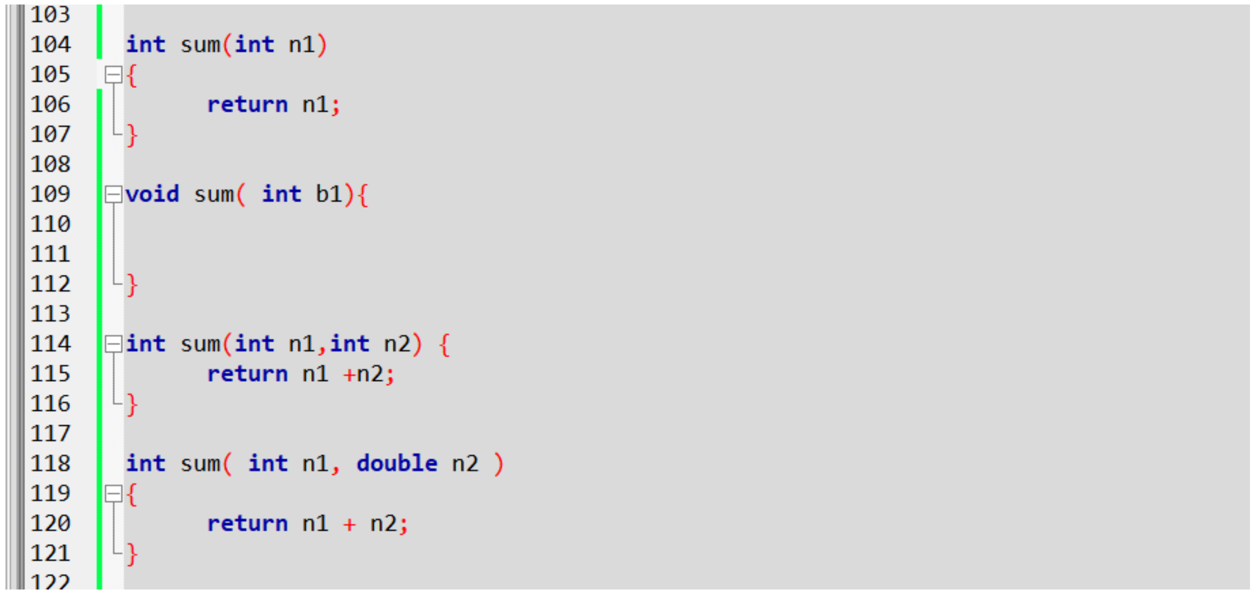
Fig no: F14(output part)

7.2.3. Multiple Instances

Multiple attributes or methods using the same name can create ambiguity in the program. This will lead to an inability to convert the program into machine code by the compiler as there are multiple instances of an attribute/ method with the same name that does different things and C does not support polymorphism.

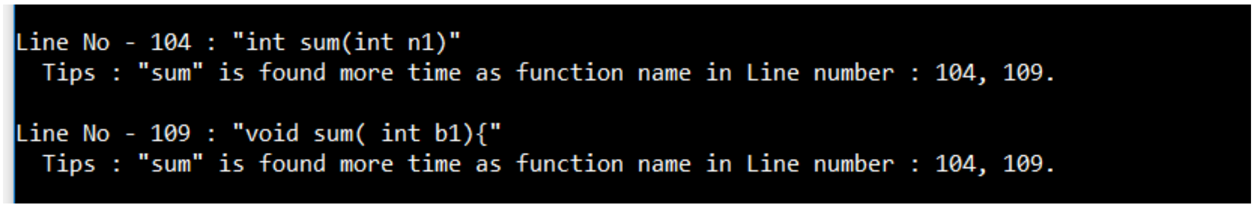
7.2.3.1. Multiple Instance of Methods

The user might declare two or more methods with the same name. This will result in an error shown in fig no: F16 and this is caused by the code snippet shown below in fig no: F15 where there are multiple instances of the method called “sum” in lines 104, 109, 114, and 118.



```
103
104  int sum(int n1)
105  {
106          return n1;
107  }
108
109  void sum( int b1){
110
111
112  }
113
114  int sum(int n1,int n2) {
115          return n1 +n2;
116  }
117
118  int sum( int n1, double n2 )
119  {
120          return n1 + n2;
121  }
122
```

Fig no: F15(input part)

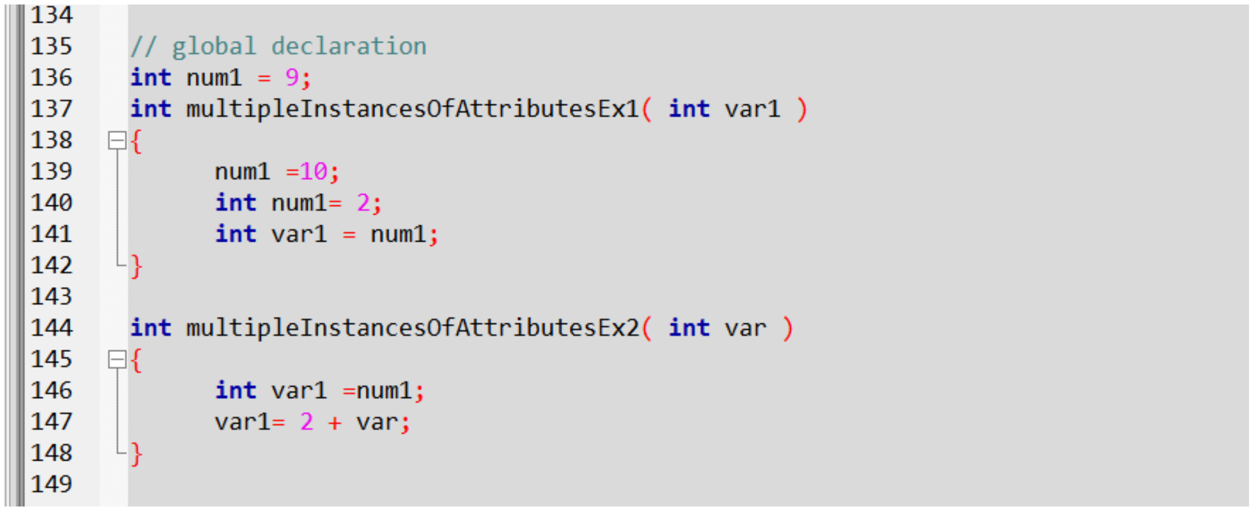


```
Line No - 104 : "int sum(int n1)"
  Tips : "sum" is found more time as function name in Line number : 104, 109.

Line No - 109 : "void sum( int b1){"
  Tips : "sum" is found more time as function name in Line number : 104, 109.
```

Fig no: F16(output part)

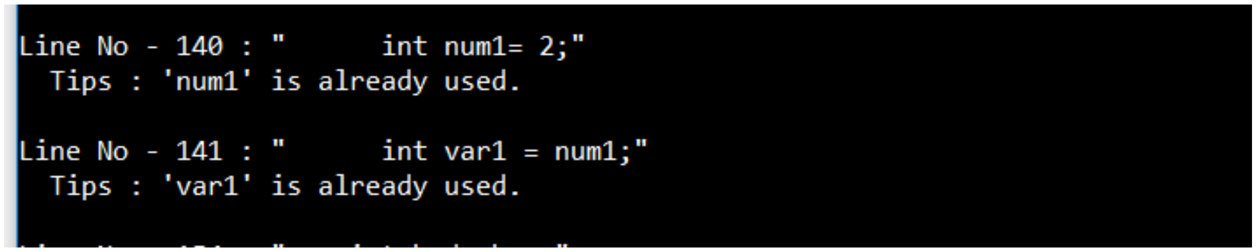
7.2.3.2. Multiple Instance of Attributes

The user might declare two or more attributes with the same name. If there are multiple global attributes declared in the same program with the same name or there are multiple attributes declared in the same scope with the same in it will result in an error. This is shown in the following figures F17 and F18.



```
134
135  // global declaration
136  int num1 = 9;
137  int multipleInstancesOfAttributesEx1( int var1 )
138  {
139        num1 =10;
140        int num1= 2;
141        int var1 = num1;
142  }
143
144  int multipleInstancesOfAttributesEx2( int var )
145  {
146        int var1 =num1;
147        var1= 2 + var;
148  }
149
```

Fig no: F17(input part)



```
Line No - 140 : "       int num1= 2;"
  Tips : 'num1' is already used.

Line No - 141 : "       int var1 = num1;"
  Tips : 'var1' is already used.
```

Fig no: F18(output part

7. Future Scope

I want to increase its effectiveness and correctness. And also I have a plan to generate output for the future.

8. Conclusion

So, as shown in the given screenshots, this program can successfully detect the syntactical errors of code written in C. And furthermore a feature of this program works as a fairly simple text editor for writing C code with error checking.

9. GitHub Link

Project repository Link: https://github.com/inzamam-inz/Software-Project-Lab-1

10. References

What is a Compiler? - Definition from Techopedia
EECS 473 - Compiler Design
Introduction to Abstract Syntax Trees
Tutorial Points, Compiler Design - Lexical Analysis