

SPL-1 Project Report, 2019

Project Name :Auto-Commenting

Course: Software Project Lab I

Course No: SE 305

Submitted by

Md. Nadim Ahmed

BSSE Roll No. : 1028

BSSE Session: 2017-2018

Supervised by

Dr. Kazi Muheymin-Us Sakib

Designation: *Professor*

of Information Institute Technology

University of Dhaka

Submission Date

29-05-2018

Table of Contents

1.Introduction	4
1.1.Background study	5-7
1.2.Challenges	7-8
2.Project Overview	8
3.User Manual	10-15
4.Conclusion	16
5.References	16
Appendix.....	16

1.Introduction

Code comments are an integral part of a code. They improve program comprehension .The lack of code comments is a common problem to understand a unknown code. Therefore, it is beneficial to generate code comments automatically. In this project, it is proposed a general approach to generate code comments automatically by analyzing codes.

The project implementation is started with collecting the first year students' code of BSSE 10th batch .Then, the project will be implemented with two main tasks:-

1. Commenting on functional blocks of an uncommented source code.
2. Commenting after finding similarities between uncommented and commented source code.

1.1. Background study

Lexical analysis

Lexical analysis, lexing or tokenization is the process of converting a sequence of characters (such as in a computer program or web page) into a sequence of tokens (strings with an assigned and thus identified meaning). A program that performs lexical analysis may be termed a *lexer*, *tokenizer*, or *scanner*, though *scanner* is also a term for the first stage of a lexer. A lexer is generally combined with a parser which together analyze the syntax of programming. A *lexeme* is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token.

Common token names are

- identifier: names the programmer chooses;
- keyword: names already in the programming language;
- separator (also known as punctuators): punctuation characters and paired-delimiters;
- operator: symbols that operate on arguments and produce results;
- literal: numeric, logical, textual, reference literals;
- comment: line, block.

Syntax

In programming, syntax refers to the rules that specify the correct combined sequence of symbols that can be used to form a correctly structured program using a given programming language. Programmers communicate with computers through the correctly structured syntax, semantics and grammar of a programming language.

Semantics

Semantics is a linguistic concept separate from the concept of syntax, which is also often related to attributes of computer programming languages. The idea of semantics is that the linguistic representations or symbols support logical outcomes, as a set of words and phrases signify ideas to both humans and machines.

Abstract Syntax Tree

An abstract syntax tree (AST), or just syntax tree, is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code. The syntax is "abstract" in the sense that it does not represent every detail appearing in the real syntax, but rather just the structural, content-related details. For instance, grouping parentheses are implicit in the tree structure, and a syntactic construct like an if-condition-then expression may be denoted by means of a single node with three branches.



Figure 1: Abstract Syntax Tree

1.2 Challenges

Implementing a new software solution carries with it a number of challenges. The process can be overwhelming, confusing and lengthy. Implementing this project there are lot of challenges that I have faced. Some of them are:-

- Handling large code for the first time
- Learning and understanding algorithm
- Reading Abstract Syntax Tree
- Reading Syntactically similarity

- Implementing Abstract Syntax Tree
- Matching similarity of two code

2. Project Overview

I have divided my whole project into four different parts. They are:-

- Commenting on functional blocks
- Implementation of syntactically similarity
- Implementation of semantically similarity
- Implementation of Abstract Syntax Tree

3. User Manual

3.1 Commenting on functional blocks

Input Function:-

```
void partition(int *arr,int low,int high)
```

```
{  
  
    int pivot,temp,i,j;  
  
    pivot=arr[high];  
  
    for(i=low-1,j=low;j<high;j++)  
  
    {  
  
        if(arr[j]<pivot)  
  
        {  
  
            i+=1;  
  
            temp=arr[j];  
  
            arr[j]=arr[i];  
  
            arr[i]=temp;  
  
        }  
  
    }  
  
    temp=arr[high];  
  
    arr[high]=arr[i+1];  
  
    arr[i+1]=temp;  
  
    return i+1;  
  
}
```

Output Function:-

void partition(int *arr,int low,int high)//The name of the function is ***partition***.The return type of the function is ***void***.The parameters of the function respectively are (*arr,low,high) and the data type of the parameters respectively are (int,int,int).

```
{  
  
    int pivot,temp,i,j;
```



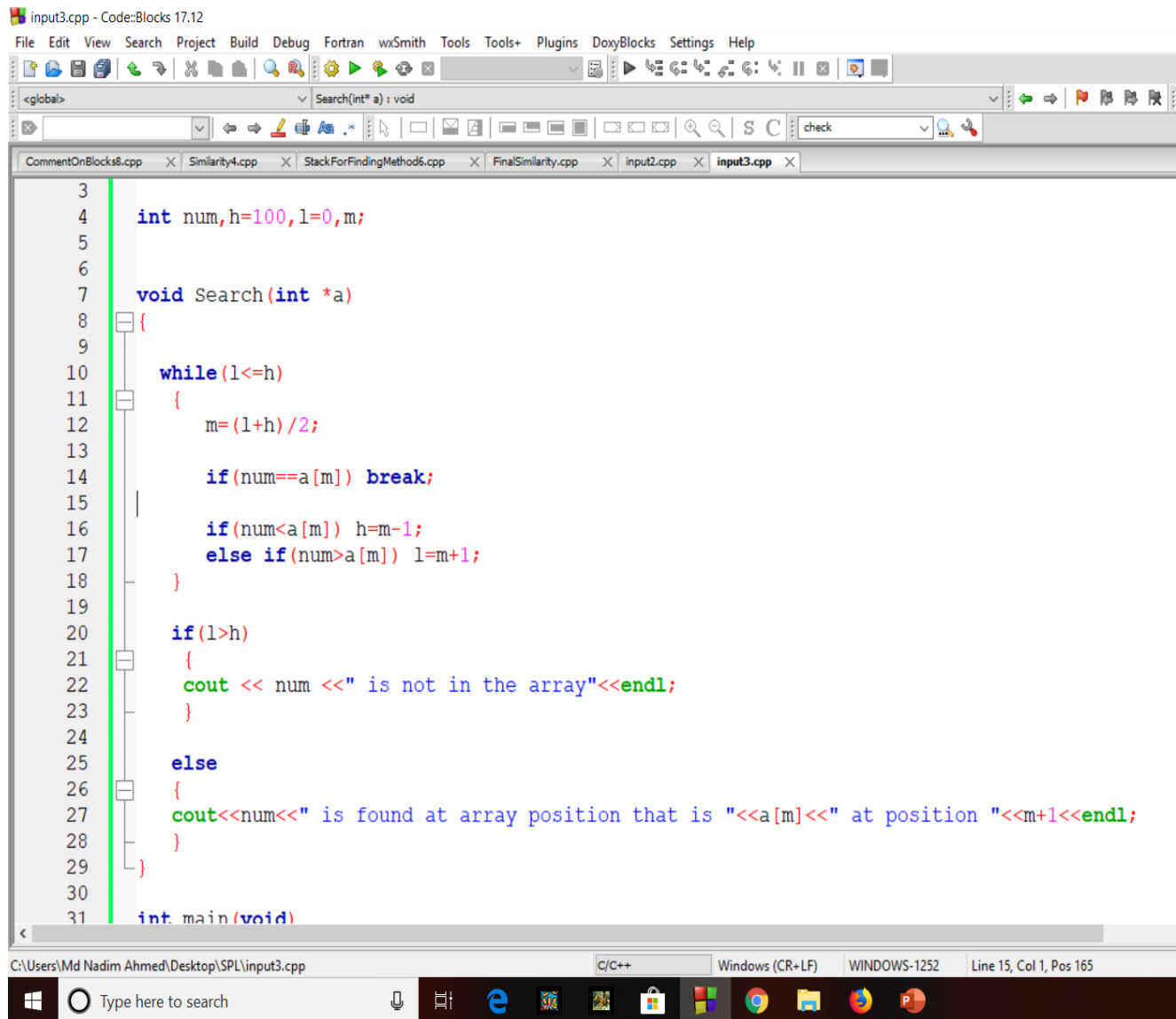
```
pivot=arr[high];
```

for(j=low-1,j=100;j++)//This is a for loop which starts with 1 and ends with till the value .The for loop is incremental. The variable name is `***i***` and the data type of the variable is `***int***`.

```
{  
    if(arr[j]<pivot)  
    {  
  
        i+=1;  
        temp=arr[j];  
        arr[j]=arr[i];  
        arr[i]=temp;  
    }  
}  
temp=arr[high];  
arr[high]=arr[i+1];  
arr[i+1]=temp;  
return i+1;  
}
```

3.2 Syntactically Similarity

This similarity is implemented by using string of two source codes. Then the both strings are compared with edit distance algorithm.



```
input3.cpp - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

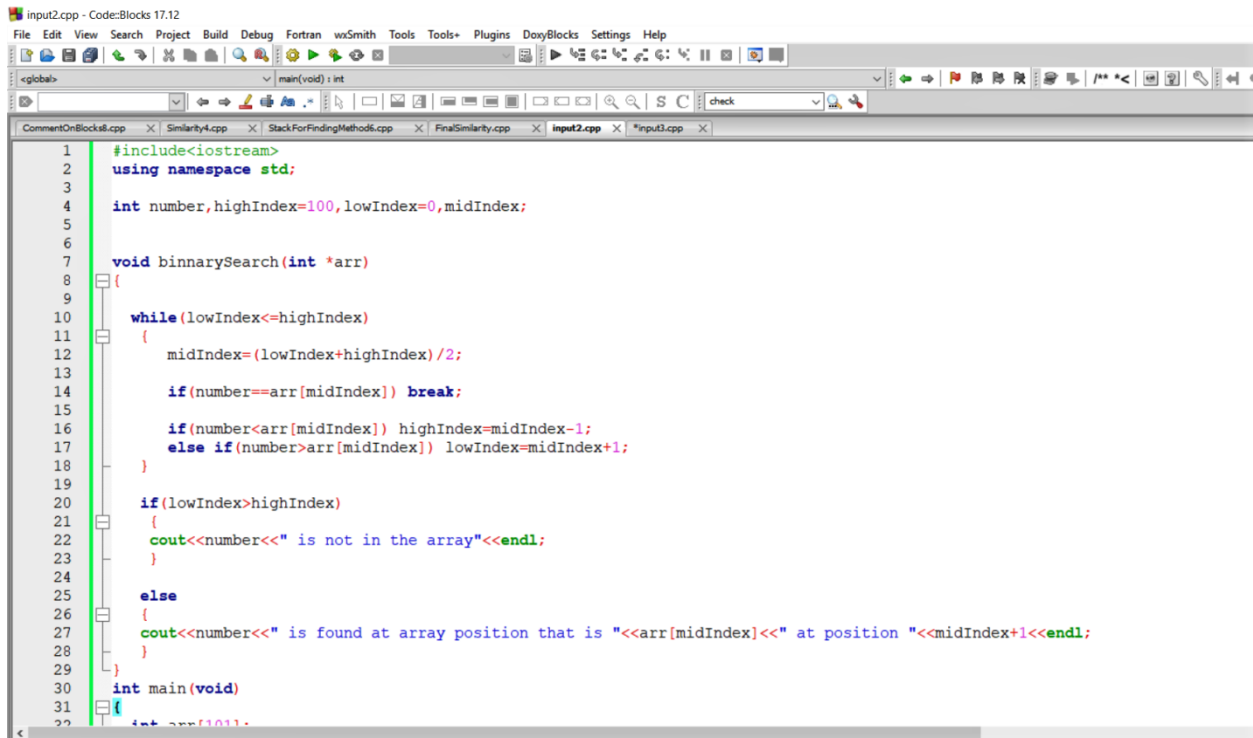
<global> Search(int* a) : void
check

CommentOnBlocks8.cpp x Similarity4.cpp x StackForFindingMethod6.cpp x FinalSimilarity.cpp x input2.cpp x input3.cpp x

3
4 int num,h=100,l=0,m;
5
6
7 void Search(int *a)
8 {
9
10 while(l<=h)
11 {
12     m=(l+h)/2;
13
14     if(num==a[m]) break;
15
16     if(num<a[m]) h=m-1;
17     else if(num>a[m]) l=m+1;
18 }
19
20 if(l>h)
21 {
22     cout << num << " is not in the array"<<endl;
23 }
24
25 else
26 {
27     cout<<num<< " is found at array position that is "<<a[m]<< " at position "<<m+1<<endl;
28 }
29 }
30
31 int main(void)
```

C:\Users\Md Nadim Ahmed\Desktop\SPL\input3.cpp C/C++ Windows (CR+LF) WINDOWS-1252 Line 15, Col 1, Pos 165

Figure 2:User manual: Input 1



The screenshot shows the Code::Blocks IDE with the file 'input2.cpp' open. The code implements a binary search algorithm. It includes the `<iostream>` header and uses the `std` namespace. A global variable `number` is declared with `highIndex=100`, `lowIndex=0`, and `midIndex`. The `binnarySearch` function (note the typo) takes an integer array `arr` and performs a binary search. It uses a `while` loop to find the element, updating `midIndex` as $(lowIndex + highIndex) / 2$. If the element is found, it prints the position. If not, it prints a message. The `main` function is partially visible at the bottom.

```
1  #include<iostream>
2  using namespace std;
3
4  int number,highIndex=100,lowIndex=0,midIndex;
5
6
7  void binnarySearch(int *arr)
8  {
9
10     while(lowIndex<=highIndex)
11     {
12         midIndex=(lowIndex+highIndex)/2;
13
14         if(number==arr[midIndex]) break;
15
16         if(number<arr[midIndex]) highIndex=midIndex-1;
17         else if(number>arr[midIndex]) lowIndex=midIndex+1;
18     }
19
20     if(lowIndex>highIndex)
21     {
22         cout<<number<<" is not in the array"<<endl;
23     }
24
25     else
26     {
27         cout<<number<<" is found at array position that is "<<arr[midIndex]<<" at position "<<midIndex+1<<endl;
28     }
29 }
30
31 int main(void)
32 {
33     int arr[101];
```

Figure 3:User manual: Input 2

```

1  #include<iostream>
2
3
4  KKKKII=II=IKKK*IKI=II=I+I/IKI=IKKKIIII=I=IKKKIIII=I+IKI=IKKKIKKKIKKKIKKKIKKKII=IKI+IKKKKKIKKI=IIII++II=IIKKIKI
5  KKKKKI=IK=IKKIK*KKK=IK=K+I/IKK=KKKKKKKI=K=IKKKKKK=K+IKKIKKKKKIKKKKKKKIKKKKKIKK+IKKKKKIKKI=IIII++KI=IKIKKI
6  30
7  Process returned 0 (0x0)   execution time : 1.222 s
8  Press any key to continue.
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28  for (int j=0; j<=len2; j++)
29  {
30
31      if (i==0)
32          dp[i][j] = j;
33
34      else if (j==0)
35          dp[i][j] = i;
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Figure 4:Output of Syntactically Similarity

3.3 Sementically Similarity

This similarity is implemented by splitting the functions of two codes. Then the functions are concatenated with a main function format. By using a function named “system”, the codes are run from the terminal and got output.If the output of both codes are same for same input.Then, we can tell that the codes are syntactically similar.


```
148 void binnarySearch(int *arr)
149 {
150     while(lowIndex<=highIndex)
151     {
152         midIndex=(lowIndex+highIndex)/2;
153         if(number==arr[midIndex]) break;
154         if(number<arr[midIndex]) highIndex=midIndex-1;
155         else if(number>arr[midIndex]) lowIndex=midIndex+1;
156     }
157     if(lowIndex>highIndex)
158     {
159         cout<<number<<" is not in the array"<<endl;
160     }
161     else
162     {
163         cout<<number<<" is found at array position that is "<<arr[midIndex]<<" at position "<<midIndex+1<<endl;
164     }
165 }
166 Enter the file name to compile :
167
168
169
170
171 system(command);
172 //system("./a.out");
173 system("a.out");
174
175 return 0;
176 }
```

Figure 5:User manual: Output 1

```
tran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
x Input.txt x Output.txt x SemanticSimilarity.cpp x Input.cpp x InputSem.cpp x Output.cpp x
C-1) "C:\Users\Md Nadim Ahmed\Desktop\SPL\SemanticSimilarity.exe"
aker
Make
{
    if(arr[j]<arr[minimumIndex])
    {
        minimumIndex=j;
    }

    if(minimumIndex!=i)
    {
        int temp=arr[i];
        arr[i]=arr[minimumIndex];
        arr[minimumIndex]=temp;
    }
}
100}
Output file started
the
Enter the value of array.
lena 3 4 2 7 1
1
2
3
4
7
Enter a file name :
mmman
);
out";
);
```

Figure 6:User manual: Output 1

 "C:\Users\Md Nadim Ahmed\Desktop\SPL\SemanticSimilarity.exe"

Enter a file name :

Input.cpp

```
int countingSort(int *arr,int n ,int *crr)
```

```
{
```

```
    int b=20;
```

```
    int *brr;
```

```
    brr = new int[b] ;
```

```
    for(int i=0 ; i<b ; i++)
```

```
    {
```

```
        brr[i]=0 ;
```

```
    }
```

```
    for(int i=0 ; i<n ; i++)
```

```
    {
```

```
        ++brr[arr[i]] ;
```

```
    }
```

```
    for(int i=1 ; i<b ; i++)
```

```
    {
```

```
        brr[i]=brr[i-1]+brr[i] ;
```

```
    }
```

```
    for(int i=n-1 ; i>=0 ; i--)
```

```
    {
```

```
        crr[brr[arr[i]]-1] = arr[i] ;
```

```
        --brr[arr[i]] ;
```

```
    }
```

```
}
```

Output file started

Enter the value of array.

3 4 2 7 1

1 2 3 4 7

Process returned 0 (0x0) execution time : 49.422 s

Press any key to continue.

Figure 7:User manual: Output 2

Lexing.txt - Code::Blocks 17.12

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

CommentOnBlocks8.cpp x Input.txt x Output.txt x SemanticSimilarity.cpp x Input.cpp x InputSem.cpp x Output.cpp x Tokenization.cp

```
1  int dataType
2  main IF
3  ( FIRST_BRACKET_OPEN
4  ) FIRST_BRACKET_CLOSE
5  { SECOND_BRACKET_OPEN
6  int dataType
7  n IF
8  , COMA
9  i IF
10 , COMA
11 m IF
12 = ASSIGNMENT
13 0 INT_LIT
14 , COMA
15 flag IF
16 = ASSIGNMENT
17 0 INT_LIT
18 ; SEMICOLON
19 cout IDENT
20 < LESSER
21 < LESSER
22 "Enter the Number to check Prime: " STRING_LITERAL
23 ; SEMICOLON
24 cin IDENT
25 > GREATER
26 > GREATER
27 n IF
28 ; SEMICOLON
29 m IF
30 = ASSIGNMENT
31 n IF
32 / DIVISION
33 2 INT_LIT
34 ; SEMICOLON
35 for IF
36 ( FIRST_BRACKET_OPEN
37 i IF
38 = ASSIGNMENT
39 2 INT_LIT
40 ; SEMICOLON
41 i IF
42 < LESSER
43 <= LE
44 = ASSIGNMENT
```

Figure 8:Lexical Analysis

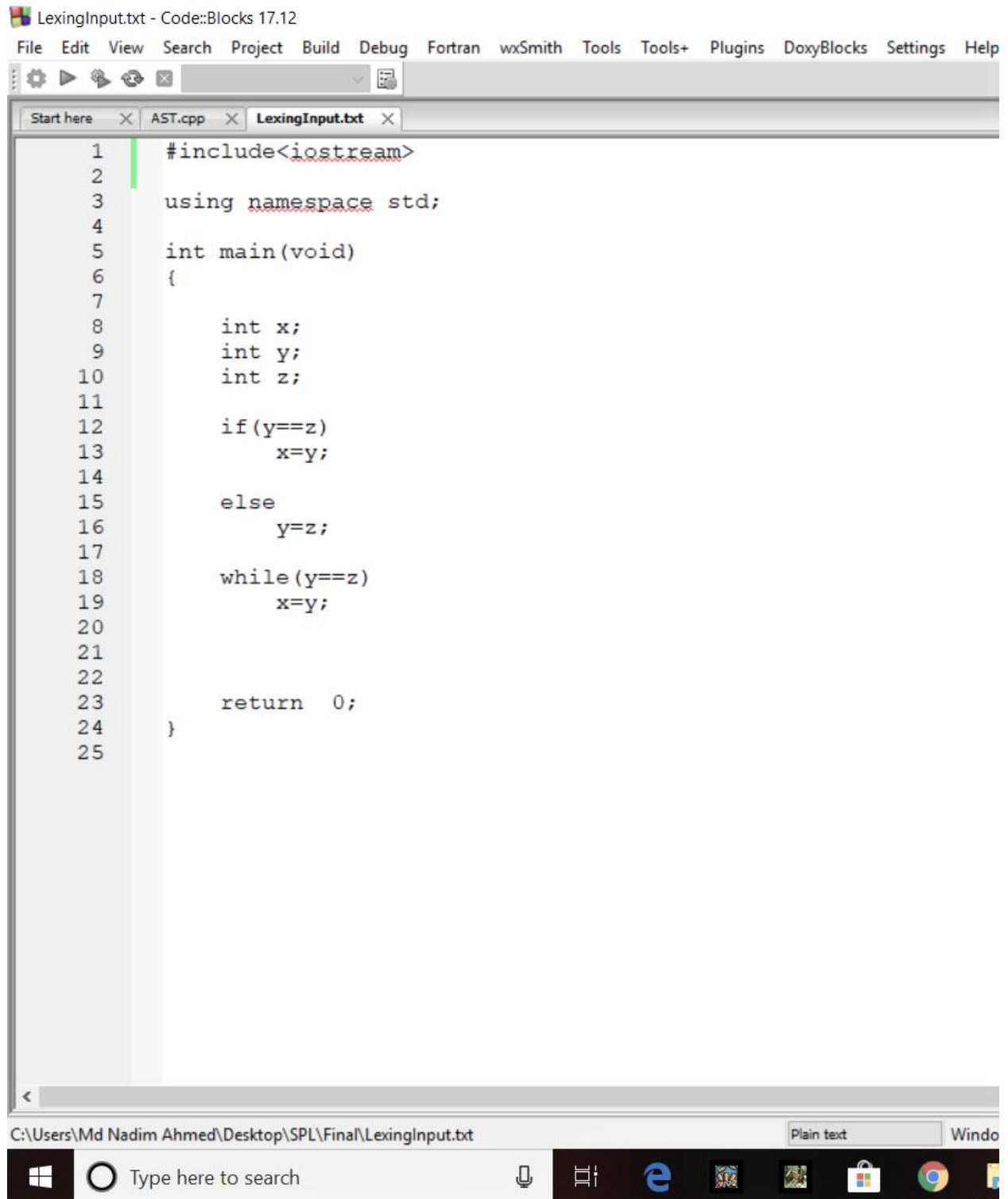



Figure 9:Input File

 "C:\Users\Md Nadim Ahmed\Desktop\SPL\Final\AST.exe"

The File has been opened

Abstract Syntax Tree Traversal in Pre-Order:

```
Program_Starting
declaration_list
declaration
function_declaration
parameters (VOID)
local_declarations
local_declare
data_type (int)
Identifier (x)
local_declare
data_type (int)
Identifier (y)
local_declare
data_type (int)
Identifier (z)
statement_list
statement
if_statement
statement
expr_stmt
Identifier (y)
Equation (==)
Identifier (z)
statement
expression_statement
Identifier (x)
ASSIGNMENT (=)
Identifier (y)
else_statement
statement
expression_statement
Identifier (y)
ASSIGNMENT (=)
Identifier (z)
```

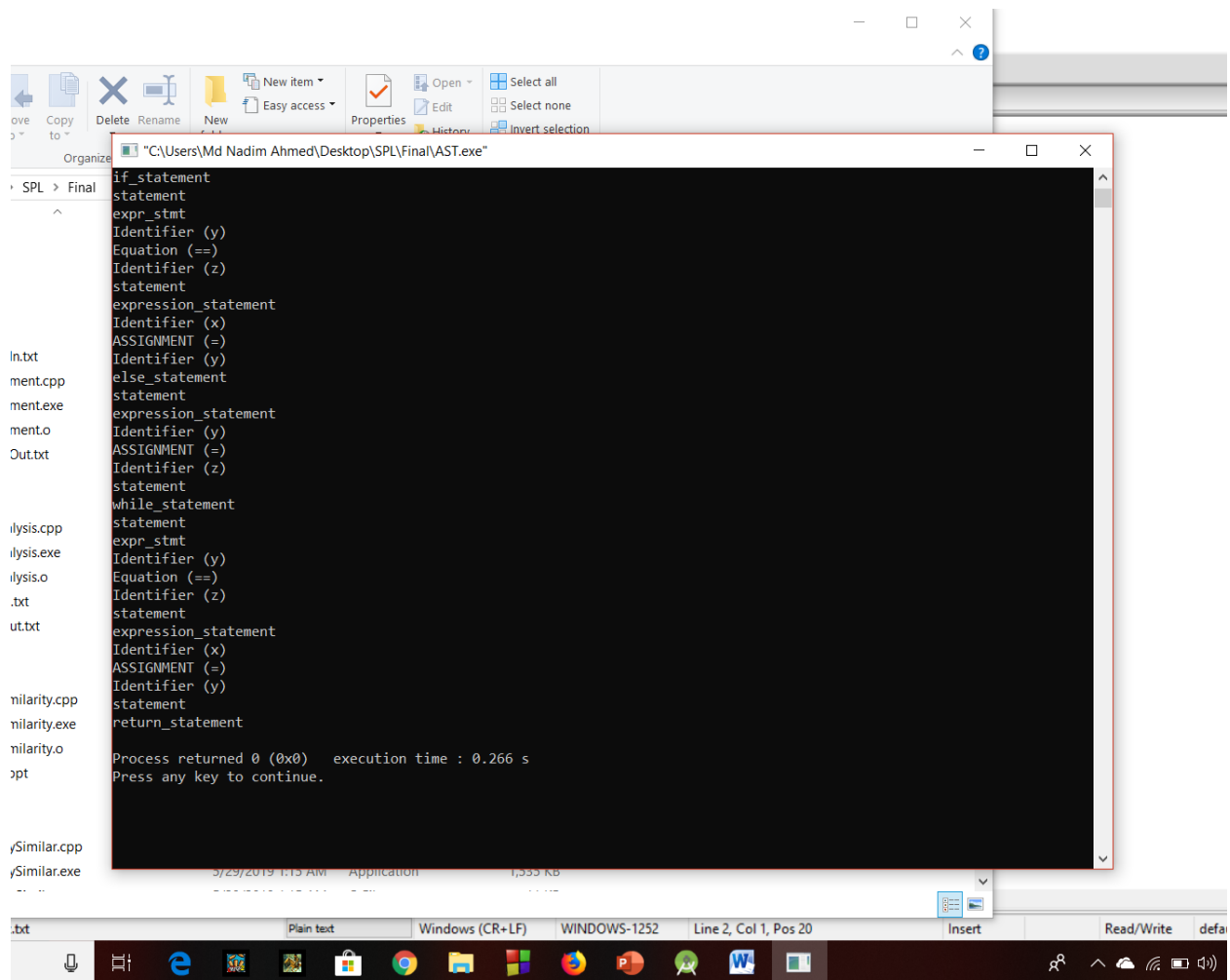


Figure 9:Abstract Syntax Tree Traversal.

4. Conclusion

Implementing this project helps me to improve my coding skill and I have learned to handle large code for the first time. I hope it will help me to deal with difficulties in future. This project was quiet challenging and I gained a lot of experience from it. I want to thank my supervisor for guiding me a lot during this project.

5. References

<https://ece.uwaterloo.ca/~lintan/publications/clocom-saner15.pdf>

https://en.wikipedia.org/wiki/Lexical_analysis

<https://www.techopedia.com/definition/3959/syntax>

https://en.wikipedia.org/wiki/Abstract_syntax_tree

Appendix

In this project, I have implemented auto-commenting in a source code for first year student's code. In future, I want to implement this project for a large code.