# An Efficient and Simplest Algorithm for Intelligent Comments Generation of the Input Code that is in High Level Language

Syed Hussain Abid, Huma Ayub Vine

**Abstract**— Coding and development is the core phase in software development life cycle (SDLC) of software engineering (SE). This phase involves transformation of hypothetical logic into concrete logical program by using any programming language. Every student of technical education has to do specific programming at some level of his pedagogical or technical career due to the transferal of every related activity on computer framework. Getting started with coding is a bit challenging due to the use of high level languages such as C++. It is even harder if you have to understand and alter the code of someone else .We propose a system that will generate comments on a pre-written code and will provide an ease to the beginners and high techs in understanding that pre-written code if the one has to alter a pre-written code. This system will be a great help for those who want to study coding by visualizing exemplary codes as our system will generate a report in human understandable language.

**Index Terms**— it includes Software Engineering (SE), SDLC, High Level Language, Parsing, Algorithm, Artificial Intelligence and Neural Networks.

## 1 Introduction

Information Technology (IT) is playing an essential role in the encroachments of this world. It is an affirmed reality and widely acknowledged that new development in this world is not possible without the aid of computer. A nation's ability to solve problems and initiate and sustain economic growth

Depends partly on its capabilities in science, technology, and innovation [6].Keeping this in view every institution is enhancing their researches in the field of IT to offer their contributions in the progress and prosperity of humanity. Software has its fundamental share in this progress. Students of engineering learn many high level languages by which problems can be solved logically by writing program in the learnt high level languages. Many languages like C++, Java, C#, python and many other are used for this purpose. Beginners of a programming language face a lot of difficulty in understanding the Syntax and semantics of programming language as it is an excepted reality that "it takes 10 years for a novice to become an expert programmer" [2]. This subject is quite critical and cannot be understood without a teacher's assistance because if a novice has a code that he has to understand than he might face much problem in understanding that code snippet. It would be a great help for him if he has some handouts or notes regarding the logic he is trying to understand.

We purpose a comment generating system that will generate description about every line of input code. This system will generate handout notes which will have description in human readable language of every line i.e. every code line following its description.This system will generate intelligent description that will not merely increase the readability of the code but also will be a great assistance to understand the logic of code written in high level language with the help of description generated in natu-ral language in this way it can be a great help for beginners.This system will not only be a help for beginners but will also support the transformation of a high level language into a natural language at all levels and will be a great asset for technical document generation. Software documentation is a critical document that acts as a communication medium between members of development team and a high proportion of Software cost is incurred in documentation [7]. This system will facilitate a user in the following ways:

- User can input a program in the given area and enter.
- Software system will generate comments in front of each line of code (Fig.1).

User can also print a well formatted report in which comments will be placed after each line (Fig. 2).

## 2 Practical Considerations

Generating description in natural language primarily require the high level language to be parsed. One thing that we assure before parsing is that the code entered for generating its documentation is syntactically and semantically correct because we are not making or merging compiler in our system to correct errors. Parsing the given code means to convert it into tokens, Tokens are the small meaning full chunks of any language [4]. Tokenization is useful both in linguistics (where it is

- *Syed Hussain Abid is currently pursuing MS degree program in Software Engineering in University of Engineering and Technology, Taxila, Pakistan, PH: 92-331-5548669.*
  *E-mail: hussainabid99@yahoo.com*
- *Huma Ayub Vine is currently teaching in masters degree program in Software Engineering in University of Engineering and Technology, Taxila, Pakistan, E-mail: huma.ayub@uettaxila.edu.pk*

a form of text segmentation), and in computer science, where it forms part of lexical analysis [3]. If the language under consideration is C++ and the given code and Tokens created by parser will be like:

```
# include <iostream.h>
void main()
{
int a=10;
}
```

| Tokens | |
|---|---|
| # | ) |
| include | { |
| < | int |
| iostream.h | a |
| > | = |
| void | 10 |
| main | ; |
| ( | } |

Parser will generate a tokenized array which will be used in the comment generation of the input code. The logic we are using for this purpose is expected token logic. For this purpose Parse tree diagrams will be used.A concrete syntax tree or parse tree or parsing tree [1] is an ordered, rooted tree that represents the syntactic structure of a string according to some formal grammar.Parse tree diagrams are tools that are used to see the possible tokens after a given token. Expected tokens are specific to every language and they can be used to generate algorithm for comment generation of the input code

## 2.1 Sample Examples

We will have 2 examples of parse tree diagrams which will elaborate the use of these in generating descriptions.

### 2.1.1 Example # 1

This example is a parse tree diagram of a scenario if the encountered token is a Data type name. C++ has various data types like int, float, char, double and so on. Their general parse tree diagram covering all scenarios is in Fig.1.

This Figure illustrates that what are the possibilities when a Data type is encountered. For example if I have “int” data type, following statements can be expected which are covered in the tree diagram:

- int var1;
- int var1, var2, var3,…….varN;
- int var1=2;
- int var1=20, var2=24;
- int FunctionName(Arguments…..)
- int arr1[10];
- int arrMulti[2][3];
- And so on.

### 2.1.2 Example # 2

This example illuminates the parse tree diagram of an Arithmetic operation. C++ has many arithmetic operators like +, -, *, /, %, ++, += and so on. In the parse tree diagram each notation is used as a separate token. A general parse tree diagram of the whole scenario in which operator is encountered is in Fig.2. If the operator encountered is a “+” following cases van be there:

- Operand = Operand + Number;
- Operand = Operand + Operand;
- Operand++;
- ++Operand;
- Operand += Number;

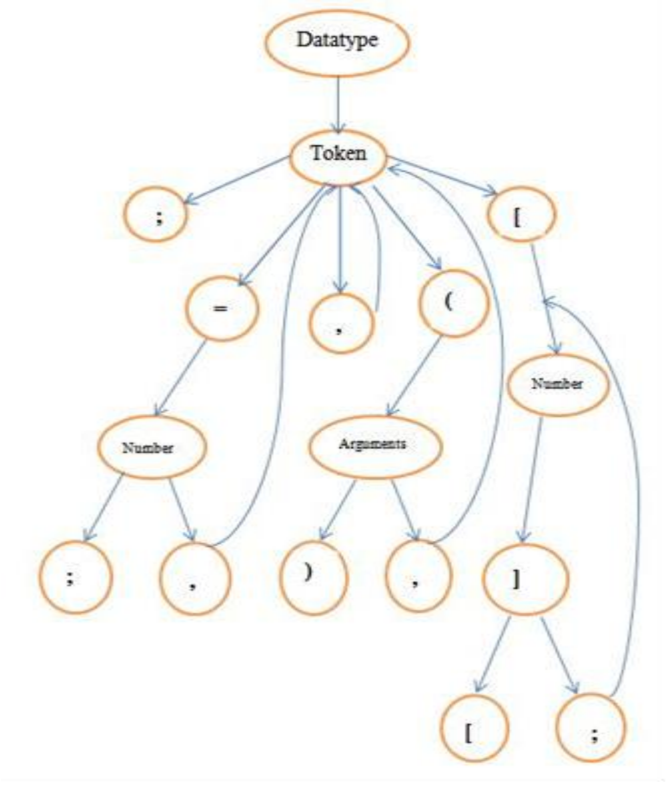


FIG. 1 PARSE TREE DIAGRAM OF DATA TYPE

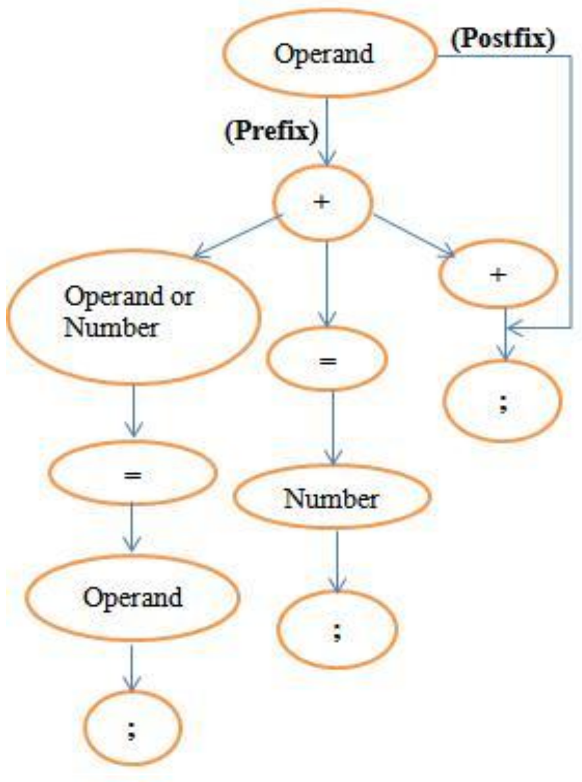


FIG. 2 PARSE TREE DIAGRAM OF OPERATORS LIKE “+”

### 2.2 Generalized Algorithm

As already explicated, algorithm will follow the expected token logic. We are tracing the expected token with the help of parse tree diagrams. When any token is encountered it is searched and small chunks are added to it to give a meaning full sentence. The small chunks or phrases will be taken referring to some authorized book of that programming language for instance C++ i.e. Programming Fundamentals [5].This generalized algorithm is:

```
String Sentence=”NULL”;
token_array [] =GetTokenizedArray (InputCode);
For i= start  → token_array. Length ()
Start
If (token_array [i] = = Keyword)
  Start
   If (token_array [i + n] = = ExpectedToken)
     Start
Sentence + = “This” + token_array [i] + “is using”
                  token_array [i + n] + “for this purpose”;
i = i + n;
    End If
   If (token_array [i + n] = = ExpectedToken)
    Start
//Multiple Ifs can be there for all expected tokens//
    End If
 End If
End For
Print (Sentence);
Sentence.Empty ();
```

When a Keyword is encountered, the algorithm will investigate between all scenarios of the keyword and take action according to the expected token found. It will look for all the expected tokens and a final generalized statement will be obtained after considering all cases.

### 2.3 Intelligent Algorithm

This algorithm can be made intelligent by using a dictionary of the encountered tokens after keywords. In this case algorithm will always search within the dictionary and will take actions accordingly. We can illustrate by a simple example of Classes.

If a code has multiple classes class A and class B and both invoke some functions. How can algorithm know that which functions is of which class? For this we have to maintain some dictionary which will contain the name of class and the functions it have. So when ever any function is invoked, the algorithm will also provide information that it is the function of which class. In this way it will be able to generate more effective descriptions.

### 2.4 Complexity analysis

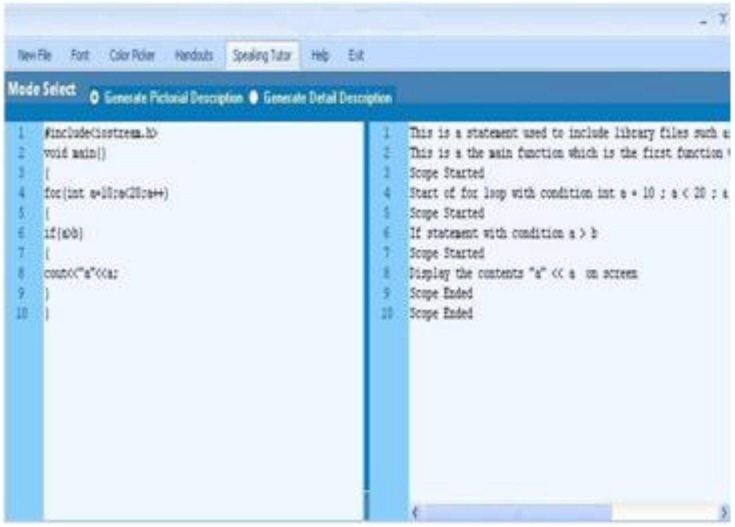
Selection of algorithm is considered an important factor for achieving maximum throughput and minimum run time for executing task. For finding complexity of any algorithm, the main goal behind it is to achieve a function which gives the efficiency of the algorithm in the form of the data measures which is to be processed by algorithm. Time complexity is another factor which describes the amount of time an algorithm takes in terms of the amount of input to the algorithm [9]. Basically it calculates the number of memory accesses, comparisons between statements, amount of time inner loop execution and execution time taken by set of elementary statements into account.

Sometimes Space complexity function is also considered which describes the amount of memory (space) an algorithm takes in terms of the amount of input to the algorithm [9]. In many cases the space complexity is ignored because the space has minimal effect but sometime this effect cannot be ignored.

Concerning with our proposed commenting based algorithm approach, the time complexity of this algorithm is calculated and it is found that the best case analysis of this algorithm is $\Omega(n)$ and worst case analysis is $O(n)$, which is linear in time complexity.

## 3 Sample Design

We have implemented this algorithm and prolific results were obtained. In Fig. 3 the software generated comments against every line of code is shown while Fig. 4 shows the handout generated by software.



```
#include<iostream.h>
void main()
{
for(int a=10;a<20;a++)
{
if(a>b)
{
cout<<"a"<<a;
}
}
```

```
This is a statement used to include library files such a
This is a the main function which is the first function
Scope Started
Start of for loop with condition int a = 10 ; a < 20 ; a
Scope Started
If statement with condition a > b
Scope Started
Display the contents "a" << a  on screen
Scope Ended
Scope Ended
```

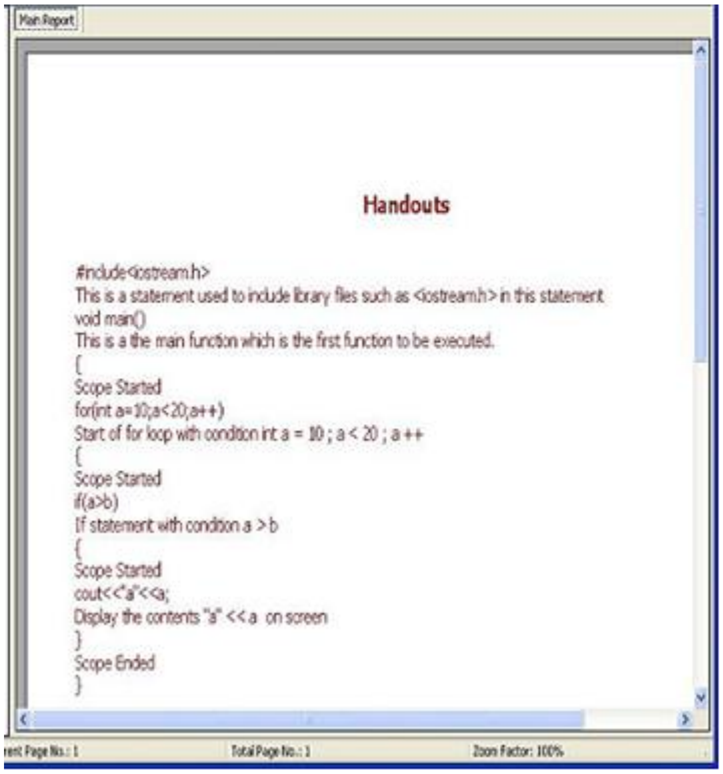
Fig. 3 The software generated comments against every line of code

## 4 Future Work

This algorithm can be enhanced using artificial intelligence and neural networks which can result in more intelligent detail of the input code. Intelligent detail will increase the undersatnding of the code any will be a great support toll for novice programmers.

## 5 Conclusion

This system is a great help for novices and if used can reduce confusion level and discrepancy between conceptual and theoretical knowledge of beginners. It can be a great asset in software engineering documentation because " Software engineering is not to produce a working software system only, but also documents such as system design, user manual, and so on" [8].



Handouts

```
#include<iostream.h>
This is a statement used to include library files such as <iostream.h> in this statement
void main()
This is a the main function which is the first function to be executed.
{
Scope Started
for(int a=10;a<20;a++)
Start of for loop with condition int a = 10 ; a < 20 ; a ++
{
Scope Started
if(a>b)
If statement with condition a > b
{
Scope Started
cout<<"a"<<a;
Display the contents "a" << a  on screen
}
Scope Ended
}
```

FIG. 4 HANDOUTS GENERATED BY THE SOFTWARE SHOWING EACH LINE OF CODE FOLLOWED BY THE COMMENT.

## Acknowledgment

I (Syed Hussain Abid) surely want to acknowledge Assisstant Professor Huma Ayub Vine for her precious time she devoted for doing the complexity analysis of the algorithm.

## References

[1] http://en.wikipedia.org/wiki/Parse_tree#cite_ref-0.

[2] Soloway, E. & Spohrer, J. (1989). Studying the Novice Programmer, Lawrence Erlbaum Associates, Hillsdale, New Jersey. 497 p.

[3] Huang, C., Simon, P., Hsieh, S., & Prevot, L. (2007).Rethinking Chinese Word Segmentation: Tokenization, Character Classification, or Word break Identification.

[4] Alfred V. Aho, Monica S. Lam, Ravi Sethi, 2007, Compilers Principles Techniques and Tools, New Delhi, Pearson Education, ISBN: 978-81-317-2101-8.

[5] Robert Lafore, Object Oriented Programming in C++, 3rd edition.

[6] Acharya, T., A.S. Daar, and P. Singer. 2003. Biotechnology and the U.N. Millennium Development Goals, Nature Biotechnology 21(12):1434–36.

[7] Sommerville, I. 2009. Software Engineering, 6th Edition. Harlow, UK: London: Pearson Education Ltd.

[8] R. Pressman, 2010, Software Engineering – A Practitioner's Approach (7th Edition), McGraw Hill.

[9] http://www.cs.utexas.edu/users/djimenez/utsa/cs1723/lecture2.html