Vol-01, Issue 03, Jun 2015.

AUTOMATIC VISUALIZATION TOOL-PGT

[1]**Anu Pandita**
[1]**K J Somaiya Institute of Engineering & IT, Sion, Mumbai, Maharashtra, India.**
[1]**anupandita@gmail.com**

Abstract — **Software visualization is a major focused area as the learning of the software technologies is made more easier and understandable via this core part of the software engineering. As we all know that seeing is believing and visualization has a greater effect on human mind in understanding things, the same idea if implemented in programming concept can contribute more skilled programmers to software field. In the upcoming software field ,the need of quality programmers is the need of the day. Hence to achieve quality programmers there is a need of educating the beginners to this field with basic and detailed knowledge of programming .A beginner , to understand the entire flow of the program just by reading the given source code is very difficult task ,thus only the source code proves insufficient to analyze and understand the programs .As the human character uniquely identifies and understands thousand words by a single graphical representation, We thus present a system, ―PGT (Path Generation Tool)‖which will aid the new programmers to trace the behaviour of the program with the help of graphical representation and parallel explanation of all the terms used in the program.**

Keywords— encryption, decryption, information security.

I. INTRODUCTION

The automatic visualization tool, ―PGT‖ is a system which mainly focus on the novice form ie. New programmers for their qualitative education purpose The main motive of this project is to provide effective means to understand the behaviour of the program, for students who are absolutely new to programming languages. Fresher in engineering fields mainly in computer and information technology field finds it very difficult to understand the program by looking at the source codes. The behaviour of the program is not understood completely by reading just the source code, this in turn effects the quality of the programmers developed in the software industry.
Here comes the main role of PGT, it generates path from the source code so that a programmer can understand the meaning of the program easily.PGT analyses source code of program by using parser which is important part of the compiler generated by source and then can generate the path of the entire program automatically based on the results. It is difficult task for novice users to understand the behaviour of the program only by reading the source code. Hence, something to visualize the behaviour of the program is required. PGT analyses the syntax of the source code of C/C++ or java programs by using the parser generated by source code, and then generates the graphical flow of the source code and their corresponding explanation automatically based on the results of the analysis.

ADVANTAGES OF PGT

The existing visualization tool VISUSTIN V6 generates flowchart of the source code but does not generate path with explanation to illustrate the behaviour of the program, whereas PGT generates path to a program which explains the behaviour of the program. By using PGT large scale and complex program path can be shown separately. Parallel explanation of terms used in the programming language.

II. SYSTEM ARCHITECTURE

PGT consists of two parts

Parser: - It analyses the input program and stores the result of the analysis.

P-Generator: - P generator outputs the path using the analysed data. The output of the parser is provided as the input of the P generator.

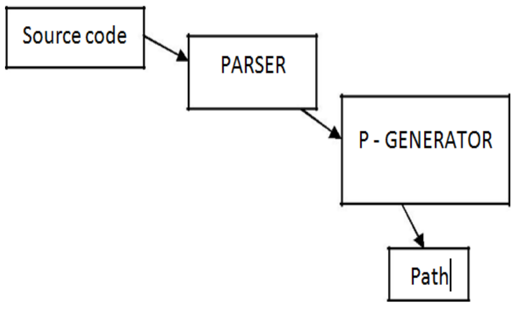


Fig.1 Architecture of Proposed System

The architecture diagram of the PGT represents the interaction of the system from the primary process of accepting input from the user , the flow of the accepted input across various modules of the system where the input is converted into suitable output which forms a corresponding input for the next module. Thus the entire procedure of the conversion of input to corresponding output traced in the architecture diagram.

MODULE 1 : PARSER

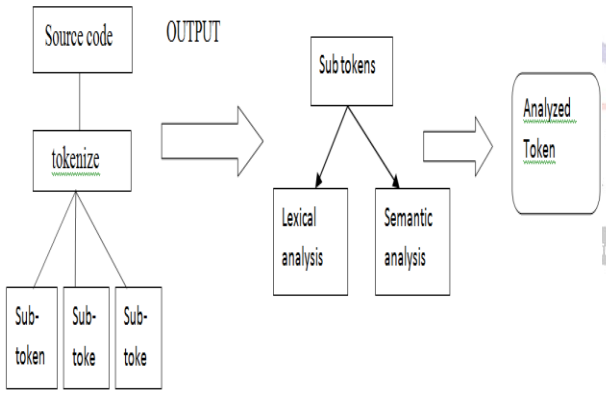


Fig.3 Block diagram of Parser

III. SYSTEM DESCRIPTION

Methodology determines various methods included in the implementation of the project for its successful completion. Methodology includes various techniques adopted and standard algorithm that needs to be implemented as a part of the implementation process. The methodology techniques and algorithms used can be discussed module by module of the project.

This module mainly includes two parsing techniques for source code reading

1) Lexical analysis

2) Syntax analysis

Lexical analysis is also called as the tokenizer. A lexical analyser is a patter matcher. A lexical analyser recognizes strings of characters as tokens.

Lexical analysis

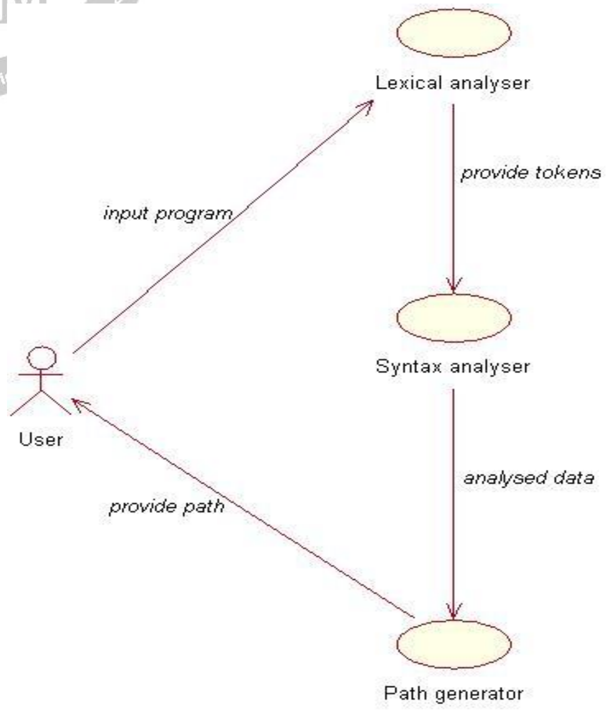
It is the process of converting a sequence of characters into a sequence of tokens. It is a low-level part of the analyser; mathematically, a finite automaton based on a regular grammar. The lexical analyser is usually a function that is called by the parser when it needs the next token

Syntax analysis

It is the second phase of the compiler after lexical analysis. It is also called as hierarchical analysis or parsing. It groups tokens of source program into grammatical checking and then it generates the parse tree

IV. ALGORITHMS

Various standard algorithms are used by the parser which also needs to be used in the project



Vol-01, Issue 03, Jun 2015.

1)LALR/LR Parsing algorithm

LR parsers have the potential of providing the fastest parsing speed of all parsing algorithms LR Parsing, or Left-to-right Right-derivation parsing, uses tables to determine when a rule is complete and when additional tokens must be read from the source string LR parser generators construct these tables by analyzing the grammar and determining all the possible "states" the system can have when parsing. LALR Parsing, or "Lookahead LR parsing", is a variant of LR Parsing . LR Parsing combines related "configuration sets" thereby limiting the size of the parse tables. As a result, the algorithm is slightly less powerful than LR Parsing but much more practical.

The LR algorithm checks the next token on the input queue against all tokens that expected at that stage of the parse. If the token is expected, it is "shifted". This action represents moving the cursor past the current token. The token is removed form the input queue and pushed onto the parse stack.

Fig.3 use case diagram

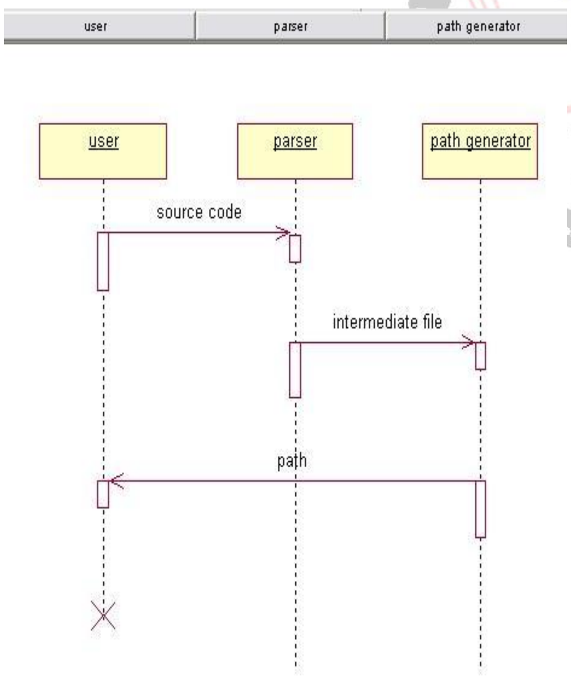


Fig.4 sequence diagram

V. OVERALL METHODOLOGY

1)Various parsing techniques stated above are used for the sourcecode reading where parser will ask laxer for a new token.
2)The token is matched with the syntax rule,if it matches a corresponding node is created using LALR or any parsing algorithm.
3) If no match is found parser will store the token internally and search for matches,if no matches found exception is raised.
4) Further the identified tokens using path generators are matched to various predefined shapes and path is generated using various graphical representing algorithms.

VI. CONCLUSION

A tool ilustrated above can generate the path of program ,thereby visualizing the source code graphically , as well as parallely explain the code terms would be of intense help to the beginner programmers to understand the programming language throughly at very basic level thereby creating quality programmers to the software industry which inturn in help in the development of the IT Industry.

REFERENCES

[1] C/C++ VIS: Automatic Program Visualizatio with Object and Sequence Diagrams Using the C/C++ Debug Interface (JDI),‖ S. Diehl (Ed.): Software Visualization.

[2] Yoshihiro Kita, Tetsuro Katayama, Shigeyuki Tomita,―Implementation and Evaluation of an Automatic Visualization Tool ‗PGT‘ for Programming Education‖, 2005.

[3] Avron B. and Shirley T., "Good Programers Are Hard To Find: An Alternative Perspective on the Immigration of Engineers,‖ Stanford Computer Industry Project, Research Note, 1996.

[4] Software Visualization and Education (chapter 3 in lecture notes in computer science, International Seminar, Dagstuhl Castle, Germany, May 20-25, 2001,Revised Papers)

[5] John Hamer, ―A Lightweight Visualizer for C/C++ ‖, Department of Computer Science.

[6] Andreas Zeller and Dorothea L¨utkehaus, ―DDD—A Free Graphical Front-End for UNIX Debuggers‖, 1995.

[7] Baeza-Yates R.A., Quezada G. and Valmadre G., ―Visual Debugging and Automatic Animation of C Programs,Software Visualization, Vol.7, pp.46-58, 1997.