[FRTSSDS- June 2018]
DOI: 10.5281/zenodo.1288409

ISSN 2348 – 8034
Impact Factor- 5.070

GLOBAL JOURNAL OF ENGINEERING SCIENCE AND RESEARCHES

STUDY OF FRONT END AND BACK END MODEL OF COMPILER

Piyush Choudhary
Prestige Institute of Engineering, Management & Research, Indore (M.P)

ABSTRACT

No two languages are alike, every language has their own unique style and methodology. If we talk about human languages there are wide no. of languages available in the world but technically computer understands only Machine Languages. Machine code is written in binary bits pattern (or hexadecimal pattern), which requires programmer exceptionally skilled with machine language. Understanding the need of programmers, various tools are developed to convert one language into another language. One of the tools to transform one language into another is Compiler. This paper presents the study of structure of compiler viz. Front End and Back End Model of Compiler. First we define the definition of compiler along with the brief history of compiler and then moved towards compiler structure: Front End and Back End model of Compiler

***Keywords**- Compiler, Lexical Analyzer, Syntax Analyzer, Semantic Analyzer, Lexemes, Sentinels, POSIX notation, Quadruples.*

I. INTRODUCTION

A Compiler is a program that reads a program written in one language known as Source Language and Translate it into an equivalent program in another language known as Target Language. As an important part of this translation process, the compiler reports to its user, the presence of error in the source program.

The Compiler takes a source program as higher level language such as C, Pascal, FORTRAIN and convert it into low level language or a machine level language such as assemble language.

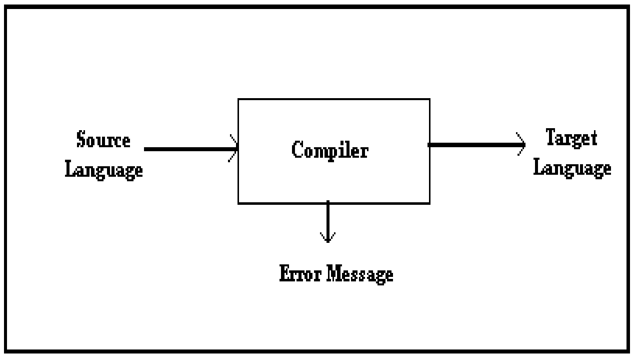

Fig: Compiler : A Basic Model

Compiler is a system software: As we all know that there are two types of software:
a) System Software
b) Application Software

Application Software are the collection of programs that are used to perform a particular task, such as Calculator.

System Software consists of a variety of programs that supports the application of a computer. They were developed to make computers better adapted to the needs of their users. It allows us to focus on application without knowing details of machine.

Examples of System Software are text editor, Compiler, Loader or Linker, Debugger, Macro Processor, Operating System etc.

Text Editor
Text Editor is used to create and modify the program.

Compiler
Compiler translates the user program into machine language

Loaders
Loaders are system programs that prepare machine language programs for execution.

Debugger
Debugger helps to detect errors in the program

Translator
Translator is used to translate assembly code into machine code. This translator is called an Assembler.

All these above mentioned system software integrate in a special software called **Operating System**.

II. DISCUSSIONS

Compiler : A Brief History
In the early days of computers, programmers had to be exceptionally skilled since all the coding was done in Machine Language. Machine Language code is written in binary bits pattern (or hexadecimal pattern). With the evolution of assembly language, programming was made simpler. These languages used mnemonics, which has close resemblance, with symbolic instructions and executable machine codes. A programmer must pay attention to far more details and must have complete knowledge of the processor in use.

Towards the end of the 1950s, machine independent language were introduced. These language had a high level of abstraction. Typically, a single high level instruction is translated into several executable machine language instructions.

Grace Murray Hopper conceptualized the first translator for the A-0 programming language from his experimental studies. This was coined as a compiler in the early 1950s.

John Backus at IBM introduces the first complete compiler for FORTRAIN in 1957.

Compiler Structure: Analysis and Synthesis Model
The compilation can be done in two parts:

1. Front End
2. Back End

In Front End the source program is read and broken down into constituent pieces. The syntax and the meaning of the source string is determined and then an intermediate code is created from the input source program.

In Back End part this intermediate form of the source language is taken and converted into an equivalent target program. During this process if certain code has to be optimized for efficient execution then the required code is optimized.

The Front End and Back End model is shown below:

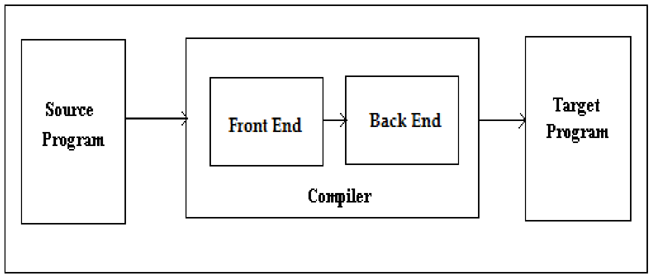

Fig Front End and Back End Model of Compiler

The Front End part is carried out in three subparts and Back End is carried out in another three subparts:

Lexical Analysis: The lexical analysis is also called Scanning. It is the phase of compilation in which the complete source code is scanned and is broken up into group of strings called token. A token is a sequence of characters having a collective meaning.

For example: If some assignment statement in source code is,

Total = count + rate*60;

Then the lexical analysis phase broken up this statement into series of tokens as:

Total	-	identifier
=	-	assignment symbol
count	-	identifier
+	-	additive operator
rate	-	identifier
*	-	multiplicative operator
60	-	number

The blank characters which are used in the programming statement are eliminated during the lexical analysis phase.

Syntax Analysis: The syntax analysis is also called **Parsing**. In this phase the tokens generated by the lexical analyser are grouped together to form a hierarchical structure. The syntax analysis determines the structure of the source string by grouping the tokens together. The hierarchical structure generated in this phase is called **Parse Tree** or **Syntax Tree**.

For e.g.: For the expression, total = count + rate * 60
the parse tree can be generated as follows:

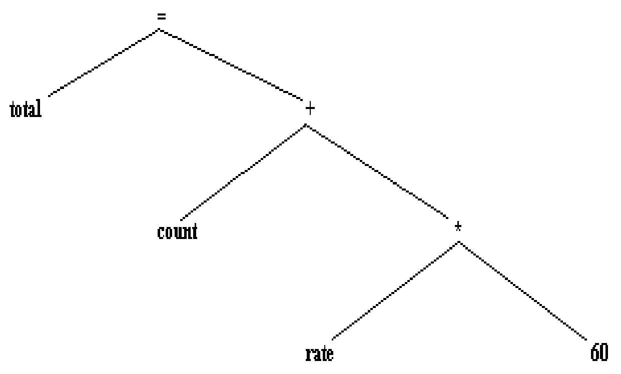

Fig: Parse Tree

Semantics Analysis: Once the syntax is checked in the syntax analysis phase the next phase i.e semantic analysis determines the meaning of the source string.

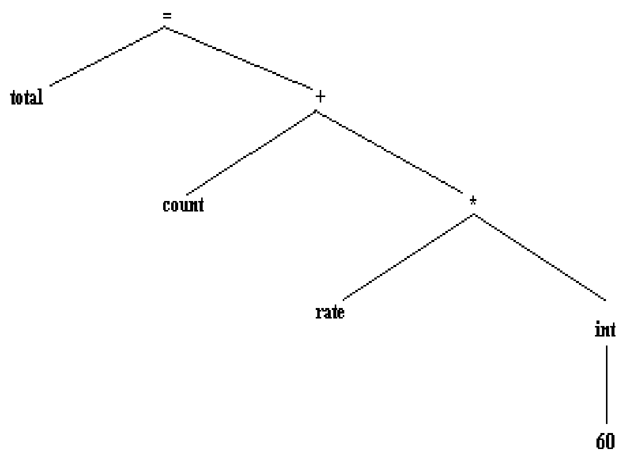

Fig: Semantic Analysis

Intermediate Code Generation: The intermediate code is a kind of code which is easy to generate and this code can be easily converted to target code. This code is in variety of forms such as three address code, quadruples, triples, posix.

For e.g.: The three address code for total = count + rate * 60 will be:

```
t1:= (int) 60
t2:= rate * t1
t3:= count + t2
total:= t3
```

Code Optimization: The code optimization phase attempts to improve the intermediate code. This is necessary to have a faster executing code or less consumption of memory. Thus by optimizing the code the overall running time of the target program can be improved. This phase doesnot provide any intermediate code, it simply improves the code.

Code Generation: In code generation phase the target code gets generated. The intermediate code instructions are translated into sequence of machine instructions.

```
For e.g.: Mov rate, R1
          Mul   #60,R1
          Mov  count , R2
         Add R2,R1
         Mov R1, total
```

Symbol Table Management

To support the analysis and synthesis model of compiler a symbol table is maintained. The task of symbol table is to store identifiers used in the program.

The symbol table also stores information about attributes of each identifier. The attributes of identifiers are usually its types, its scope, information about the storage allocated for it.

The symbol table also stores information about the subroutines used in the program. In case of subroutines the symbol table stores, the name of the subroutine, number of arguments passed to it, types of these arguments, the method of passing these arguments and return types if any.

Error Detection and Handling

In compilation each phase detects errors. These errors must be reported to error handler whose task is to handle error, so that the compilation can proceed.

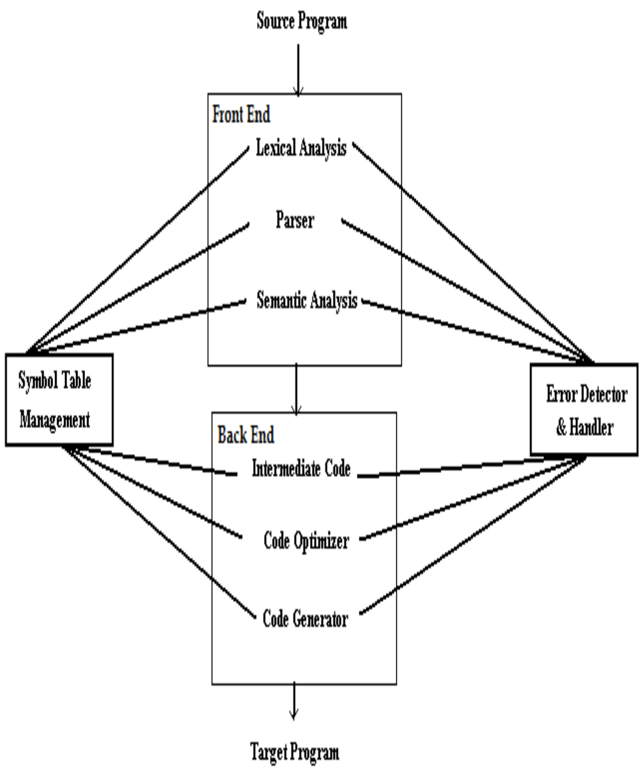

Fig: Front End and Back End Model of Compiler

III. CONCLUSION

The structure of compiler is difficult to understand but suggests a base for future studies in the field. This paper provides a base to learn compiler design by knowing the role of each phase of compiler and their parts as Front End and Back End model of compiler. This study enable the science policy makers and administrators to understand and grasp the growth, development and impact of research and to know the countries, institutions and the individual scientists who are active in particular field of research activity.

REFERENCES

[1] Charu, Chetna, Monika (2014), Research Paper on Phases of Compiler IJIRT Volume 1 Issue 5
[2] Muchnick. Advanced Compiler Design and Implementation
[3] Robert Morgan, Building an Optimizing Compiler
[4] Y.N. Srikant , P. Shankar, The Compiler Design Handbook: Optimizations
[5] Vaibhav,Vikas (2013), IJIRES, Volume 9, Issue 2
[6] Web:http://compilerdesigndetails.blogspot.in/2012/02/v-behaiouralurldefaultvmlo.html
[7] Web:http://www.codeproject.com/KB/dotnet/clr/compiler_new.gif
[8] Web:http://conferences.computer.org/sc/2012/papers/1000a012.pdf
[9] Web: http://www.cs.indiana.edu/~dyb/pubs/nano-jfp.pdf
[10]Compilers: Principles, Techniques and Tools Hardcover- Import, 31 Aug 2006 by Alfred V Aho (Author), Jeffrey D. Ullman(Author)

ISSN 2348 – 8034
Impact Factor- 5.070

[11]Andrew W. Appel Modern Compiler Implementation in ML, Cambridge University Press, Cambridge, England 1998

[12]E.Koutsofios and S.C. North, Drawing graphs with dot, AT & T Bell Laboratories, Murray Hill, NJ, 1993

[13]SJain PChoudhary and RRaghuwanshi, Study of Analysis and Synthesis Model of Compiler, PACE Journal Vol6 Issue 1