Study of Compiler Construction

Adilshah N. Jalgeri.[1], Balkrushna B. Jagadale.[2] , Rajendra S. Navale.[3]

Assistant Professors at Fabtech Technical Campus, College of Engineering and Research, Sangola

Abstract - Compiler construction is a commonly used software engineering implementation and this paper grants a compiler structure for adaptive computing. The outcome of this paper is to provide a concept of compiler design and its tools to implement an enhanced technique for compilation we have to know the characteristics of the program. This paper provides us implementation of compiler.

Keywords - Compiler, Phases of Compiler, Tools.

1. INTRODUCTION

Programming languages describes the interface between people and machines, all the software running on the computers are written in certain programming language. But, before a program can be run, it first must be translated into a form in which it can be executed by computer.The software systems that do this translation are called compilers.This paper is designed for understanding the basics of compiler Construction.

A Compiler is a program that reads a program written in one Language (source Language) and converts it in an equivalent program in another language (target language). Main part of the compiler is to report any errors in the source program that it detects during the translation process.

2. COMPILER CONSTRUCTION

A compiler is a nontrivial task so it’s better to structure the work in proper manner. To do so the compiler is divided into several phases with well-designed edges abstractly, these phases work in order each phase (except the first) taking the output from the previous phase as its input.

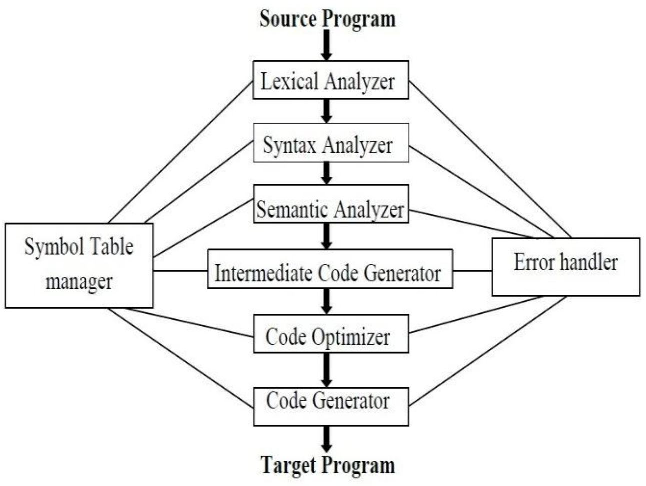

Fig1:-Phases of Compiler

A common division into phases is described below

2.1 Lexical Analysis

The first phase of a compiler is called *lexical analysis* or *scanning.* The lexical analyzer reads the stream of characters making up the source program and groups the characters into meaningful sequences called *lexemes.* For each lexeme, the lexical analyzer produces as output a *token* of the form

2.2 Syntax Analysis

The second phase of the compiler is *syntax analysis* or *parsing.* The parser uses the first components of the tokens produced by the lexical analyzer to create a tree like intermediate representation that shows the grammatical structure of the token stream. A typical representation is a *syntax tree* in which each interior node represents an operation and the children of the node represent the arguments of the operation.

2.3 Semantic Analysis

The *semantic analyzer* uses the syntax tree and the information in the symbol table to check the source program for semantic consistency with the language definition. It also gathers type information and saves it in either the syntax tree or the symbol table, for subsequent use during intermediate code generation. An important part of semantic analysis is *type checking,* where the compiler checks that each operator has matching operands

2.4 Intermediate Code Generation

In the process of translating a source program into target code, a compiler may construct one or more intermediate representations, which can have a variety of forms. Syntax trees are a form of intermediate representation they are commonly used during syntax and semantic analysis. After syntax and semantic analysis of the source program, many compilers generate an explicit low level or machine like intermediate representation, which we can think of as a program for an abstract machine. This intermediate representation should have two important properties it should be easy to produce and it should be easy to translate into the target machine.

2.5 Code Optimization

The machine independent code optimization phase tries to improve the intermediate code so that better target code will result. Usually better means faster, but other objectives may be desired, such as shorter code, or target code that consumes less power. Code optimization is a reasonable way to generate good target code.

2.6 Code Generation

The code generator takes as input an intermediate representation of the source program and maps it into the target language. If the target language is machine code, registers or memory locations are selected for each of the variables used by the program. Then the intermediate instructions are translated into sequences of machine instructions that perform the same task.

2.7 Symbol Table Management

An essential function of a compiler is to record the variable names used in the source program and collect information about various attributes of each name. And these attributes are handled by symbol table.

Here attributes may provide information about the storage allocated for a name, its type, its scope (where in the program its value may be used) and in the case of procedure names, such things as the number and types of its arguments, the method of passing each argument (for example by value or by reference) and the type returned.

2.8 Error Handler

Compiler can detect various types of errors such as

1. Lexical errors:-It consist of Misspelling, missing quotes around string texts

2. Syntactic errors: - It consists of Misplaced semicolons, extra or missing braces, Missing matching keywords.

3. Static semantic errors: - It consists of type mismatches, return values for void return method.

4. Logical errors: - Logical error may be = vs. ==

And these types of errors are recovered by following methods

1. Panic mode recovery

This method removes input symbols one at a time till one of a designated set of matching tokens is found.

2. Phrase level recovery

This method replaces a prefix of the remaining input by some string that allows the parser to continue.

3. Global correction

Here a minimal sequence of changes to obtain a globally least cost correction.

4. Error productions

Here addition of the error productions in the grammar is done.

3. COMPILER CONSTRUCTION TOOLS

Expert tools have been made to help implement various phases of a compiler. These tools use specialized languages for specifying and implementing specific components, and many use quite sophisticated algorithms. Some commonly used compiler construction tools include

3.1. Parser generators that automatically create syntax analyzers from a grammatical report of a programming language.

3.2. Scanner generators that produce lexical analyzers from a regular expression depiction of the tokens of a language.

3.3. Syntax directed translation engines that produce collections of sequences for walking a parse tree and generating intermediate code.

3.4. Code generator that produce a code generator from a group of rules for converting each operation of the intermediate language into the machine language for a target machine.

3.5. Data flow analysis engines that ease the gathering of information about how values are transferred from one part of a program to each other part. Data flow analysis is a key part of code optimization.

3.6. Compiler construction toolkits that offer a combined set of sequences for constructing various phases of a compiler.

4. APPLICATIONS OF COMPILER TECHNOLOGY

In addition to the development of a compiler design, compiler is not only about compilers, many people use the technology learned by studying compilers in institutions. Techniques used in compiler design can be applicable to many problems in computer science such as

1. Lexical analyzer can be used in text editors, information retrieval and pattern recognition.

2. Parser can be used in query processing system such as SQL.

3. Many of the software’s have complex front end for that the solution is techniques used in compiler design.

6. CONCLUSION

This paper outlines basics of the compiler construction and compiler tools. Here we described various phases of compiler which are used to construct a well-designed compiler we have also studied various applications of compiler.

7. FUTURE SCOPES

In further research we will study actual implementation of each phases used in compiler through programing language and some more advanced concepts in compiler.

REFERENCES

[1] Mahak Jain, Nidhi Sehrawat, Neha Munsi , "COMPILER BASIC DESIGN AND CONSTRUCTION", International Journal of Computer Science and Mobile Computing, Vol.3 Issue.10, October- 2014.

[2] Aastha Singh, Sonam Sinha, Archana Priyadarshi, "Compiler Construction" International Journal of Scientific and Research Publications, Volume 3, Issue 4, April 2013.

[3] Jatin Chhabra, Hiteshi Chopra, Abhimanyu Vats, "Research paper on Compiler Design", 2014 IJIRT, Volume 1 Issue 5.

[4] Alfred V. Aho, Alfred V. Aho, Ravi Sethi, "Compilers Principles, Techniques, & Tools" Second Edition.

Name of Auhtors, "Title of t

AUTHOR'S PROFILE

Adilshah N.Jalgeri. has received his Bachelor of Engineering degree in Computer Science and Engineering from V.V.P Institute of Engineering and Technology, Sholapur in the year 2013. And received M.Tech. degree with the specialization of Computer Network Engineering in S.I.E.T.Bijapur in the year 2015.His area of interest Computer Networks, System programming, Compiler Construction.

Balkrushna B. Jagadale. has received his Bachelor of Engineering degree in Computer Science and Engineering from BKEC, Bidar in the year 2013.And received M.Tech degree with the specialization of Computer Science and Engineering in P.D.A College of Engineering Kalburgi in the year 2015. His area of interest C, C++, Java.

Rajendra S.Navale has received his Bachelor of Engineering degree in Computer Engineering from S.C.O.E ,Pune in the year 2010.And received M.E degree with the specialization of Computer Networks in S.K.N College of Engineering Pune in the year 2013 His area of interest Computer Network, C#, Information and Cyber Security.