Compiler And Its Phases

Ms. Mayuri Dangare, Assistant Professor, Sinhgad College of Science, Pune & India, dangare.mayuri@gmail.com

Mr. Amit Kasar, Assistant Professor, International Institute of Information Technology, Pune, India. amitk@isquareit.edu.in

Abstract: **What is a compiler? How the compiler works? This paper tells us everything about compiler. We know that machine understands only low level language i.e. language in terms of 0's and 1's. All the software which is running on the computer system is written in some kind of language. The input to the computer system is in High Level Language and necessary to transit into the machine language. The software that translates the high level language into machine level or low level language is called as Compiler and the process of conversion is called as Compilation. This paper describes the process of compilation, the phases of compilation and how the translation occurs in short.**

Keywords — *Low level language, High level language, Translation, Compiler, source code, object code*

I. INTRODUCTION

The first implemented compiler was written by American scientist Grace Hopper in 1952, for A-0 language. The FORTRAN team led by John W. Backus at IBM introduced the primary commercially out there compiler, in 1957, which took 18 person-years to create. The first ALGOL fifty eight(58) compiler was completed by the top of 1958 by Friedrich L. Bauer, Hermann Bottenbruch, Heinz Rutishauser, and Klaus Samelson for the Z22 computer. By 1960, associate degree extended Fortran compiler, ALTAC, was available on the Philco 2000.

A compiler converts the source code into binary instructions for architecture of processor. A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running. For example, a compiler that runs on a Windows 7 PC but generates code that runs on Android smart phone is a cross compiler for e.g. a microcontroller of an embedded system. These systems contain no operating system. It perform different types of operations like semantic analysis, pre-processing, parsing, lexical analysis, conversion of input programs to an intermediate representation, code optimization and code generation.

II. COMPILER TYPES

For the conversion of the source code into machine language code, the compiler has types as described below:

i) NATIVE CODE: The compiler used to compile a source code for same type of platform only.

ii) ONE PASS COMPILER: It is a compiler which compiles the whole process in one pass only.

iii) SOURCE TO SOURCE COMPILER: The compiler that takes high-level language code as input and output source code of another high level language only.

iv) INCREMENTAL COMPILER: The compiler which compiles only changed lines from source code and update object code.

v) CROSS COMPILER: The compiler used to compile a source code for different kinds of platforms.

vi) THREADED CODE COMPILER: The compiler which simply replaces string by appropriate binary code.

vii) SOURCE COMPILER: The compiler which converts source code high level language in assembly language only.

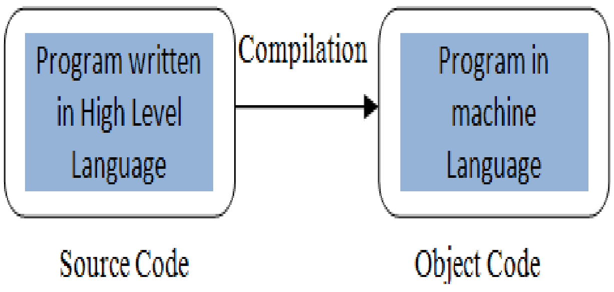
Program written in High Level Language
Compilation
Program in machine Language
Source Code
Object Code

Fig 1. Working of Compiler

A compiler could be a program that interprets the ASCII text file for an additional program from a artificial language into executable code. The ASCII text file is often during a high- level artificial language (e. g. Pascal, C, C++, Java, Perl, C#, etc.). The possible code is also a sequence of machine directions which will be dead by the processor

directly, or it's going to be associate degree intermediate illustration that's taken by a virtual machine. (e. g. Java byte code).

III. PHASES OF COMPILER

The process of compilation is partitioned into six phases as shown in figure. Each phase can interact with symbol table and error handling mechanism.

Let us understand the process with the help of example **Total= a + b * 60** as follows.

1. **Lexical Analysis:** It is also called as scanner. It is the first phase of compiler**.** It scans the source code as stream of characters. Lexical analysis means to separate words from the statements from the source code and using rules and regulations i.e. pattern it converts words into tokens.

These words are called as lexical units or lexeme.

It generates the tokens with the help of lexemes as <token_name, attribute_value> where token name is abstract symbol used during syntax analysis and attribute value points to entry in symbol table for this token.
For the above example lexical analyzer is written as
<id,1> <=><id,2><+><id,3><*><60>.
i) Token <id,1> : Here total is a lexeme, id is symbol standing for identifier and 1 points to symbol table entry.
ii) Token <=> : It is a lexeme mapped into token with no attribute value.
iii) Token <id,2> : a is a lexeme mapped into token <id,2>, where 2 points to symbol table entry.
iv) Token <+> : Plus is a lexeme mapped into token <+>.
v) Token <id,3> : b is a lexeme mapped into token <id,3>, where 3 points to symbol table entry.
vi) Token <*> : Multiplication is a lexeme mapped into token <*>
vii) Token <60> : 60 is a lexeme mapped into token <60>.

The tokens in programming language include:

Keyword such as do, if, for, while...etc.

Identifiers such as x, y, z....etc.

Operator symbol such as <, >, +, -, /, =.

Punctuation symbols such as (,), {, }, parenthesis or commas.

Some more functions of Lexical analyzer:

1) It identifies delimiters
2) Removes comments from program.
3) It removes extra blank lines and blank spaces from Source code
4) It builds tables like symbol table, literal table etc

The output of Lexical Analyzer is passed to the next

phase i.e. Syntax analysis.

2. Syntax Analysis: It is also known as parsing. It accepts tokens generated by lexical analysis.

Syntax analysis means to check the syntax of input statements with the help of stream of tokens from lexical analysis .The output of syntax analysis is parse tree.

The parser should know following things:

1) Format and structure of the language
2) Grammar of each and every statement
3) Syntax of all statements of language
4) How to store grammar in memory and how to use them

The syntax tree or parse tree for above example is as follows:

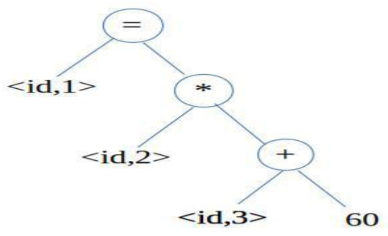

Fig 2. Syntax Tree

3. Semantic Analysis: The input of semantic analysis is parse tree. It checks the semantic (meaning) of the source text i.e. it checks the validity of argument types for the specific operation. It checks whether the parse tree to be constructed follows the rules of languages or not. Type information is saved in symbol table.

For e.g. It checks the identifiers are declared before its use.

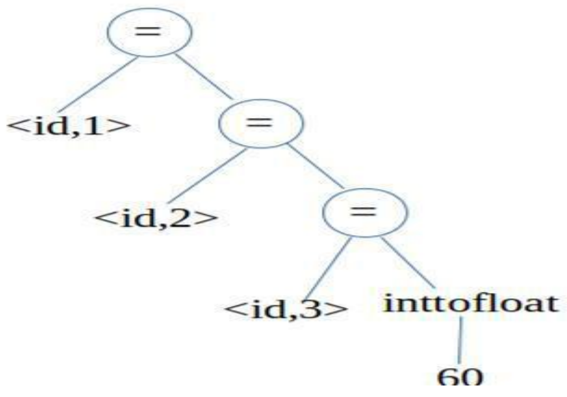

Fig 3. Semantic analysis form

4.**Intermediate Code Generator:** At the end of semantic analysis, compiler generates intermediate code. This intermediate code is easier to convert into machine code. It is easier to produce this code.

One of the intermediate codes used in many compilers is three address code. This code has at most three operands and it consist of sequence of instruction

The above example can be represented in three address code as follows.

t1=inttofloat(60)

t2=id3*t1

t3=id2+t2

id1=t3

5.**Code optimization:** It converts the intermediate code into faster executing code. There are different code optimization techniques.

a) Compile time evaluation: Some statements are there which can be executed at compile time to reduce the execution time. For e.g. constant folding, in which value of constant or operation can be replaced at the time of compilation.

b) Common sub expression: The expression is called if its value is computed previously and it is not changed since the previous computation. We can avoid recomputing and use the previously computed value.

c) Frequency reduction: The code written in high frequency region can be moved to low frequency region to reduce execution time.

d) Strength reduction: The execution time can be reduced by replacing high strength operation by low strength operation. For eg. * (multiplication) replaced by +(addition)

The optimized code for above example is as follows:

t1=id3*60.0
id1=id2+t1

6. **Code Generation:** This is the last phase of compiler.

It takes the intermediate code form and translates it into some machine instructions. These instructions are generally in assembly language.

```
LDF R2, id3
MULF R2, R2, #60.0
ADDF R1, R1, R2
     STF id2, R1
```

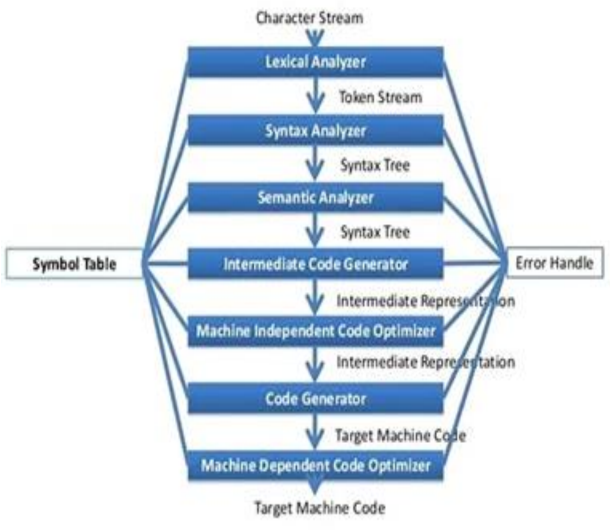

Fig.4 Phases of Compiler

IV. FEATURES

i) Error Handling: The code cannot be written in single step. People do mistakes and compiler should be able to handle those mistakes. Error handling is dealing with each phase of compiler. Compiler spots errors in program, the user go back and make the corrections in the source program listed by the compiler.

ii) Error checking is not perfect: The compiler cannot identify the logical errors.

iii) Executable file: The o/p of compiler is creation of executable file. It contains entire machine code running on CPU once the executable file has been loaded into main memory.

iv) Code optimization: Compiler translates source code into machine code through different phases and optimizes the code so that the code runs faster.

v) Makes source code independent: We know that machine code is CPU specific. Most programmers produce software using high level language. Company is paying to the programmer to produce code for different purposes .

V. ADVANTAGES AND DISADVANTAGES:

The advantages of compilers are as follows:

i) Self-Contained and Efficient: It is the major advantage that it is self-contained units which are always ready to get executed due to already compiled into machine language binaries. User does not require another package to be installed. Compiled code will execute faster than the interpreted code as once program is compiled its object file is created and saved. Such file is not created in interpreter.

ii) Hardware Optimization: The compiling of program can increase its performance. User send specific options to compilers regarding details of hardware the program will be running on. This allows the compiler to create machine language code which makes the most efficient use of the specified hardware, as opposed to more generic code. This allows advanced users to optimize the performance of a program on their computers.

The disadvantages are as follows:

i) Hardware Specific: A source code is translated into a required machine language using compiler.

ii) Compile Times: It compiles source code into machine code. The small program requires less compile time whereas the larger one will require large compile time.

VI. CONCLUSION

In this article we learn about basics of compiler. It includes how the source program is converted into target program through different phases. How errors are handled and how the machine efficient and faster running code is generated.

VII. REFERENCES

[1] Shamali Kokare, Divya Chauhan, Jyoti Mishra, Aarti Sakore, Prof. Manisha Singh, "Review Paper on Online Java Compiler", International Research Journal of Engineering and Technology (IRJET), Volume: 04 Issue: 03, Mar -2017.

[2] Ch. Raju1, Thirupathi Marupaka2, Arvind Tudigani3,"Analysis of Parsing Techniques & Survey on Compiler Applications", IJCSMC, Vol. 2, Issue. 10, October 2013,

[3] Charu Arora, Chetna Arora, Monika Jaitwal, "RESEARCH PAPER ON PHASES OF COMPILER", © 2014 IJIRT | Volume 1 Issue 5| ISSN : 2349-6002.

[4] Prof. Rajesh Babu, Prof. Vishal Tiwari, Prof. Jiwan Dehakar, "Parsing and Compiler design Techniques for Compiler Applications", International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169, Volume: 3 Issue: 2 449– 453.

[5] Muchnick . Advanced Compiler Design and Implementation.

[6] Robert Morgan. Building an Optimizing compiler.

[7] Y.N. SrikantP. Shankar. The Compiler Design Handbook: Optimizations