International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 06 Issue: 01 | Jan 2019 www.irjet.net p-ISSN: 2395-0072

INTRODUCTION TO COMPILER AND ITS PHASES

Prajakta Pahade[1], Mahesh Dawale[2]

[1,2]Computer science and Engineering, Prof. Ram Meghe College of Engineering and Management, Badnera, Amravati, Maharashtra, India.

ABSTRACT:- *Do you know what actually the compiler is? How the compiler works? In order to know everything about the compilers, let us have a introduction in this review paper. We know that all software which runs the computer system is written in some particular programming language. But, the computer can understand only the Low Level Language i.e of the form 0's and 1's. The input given to the computer is in High Level Language and need to convert into the required machine language. It needs to be translated into the required form before getting executed. The software systems that do this compilation or translation is nothing but called as the process of compilation.*

In this paper, we are going to learn about what actually a compiler is, how it works and is phases in short.

KEYWORDS: Compiler, Translation, Source code, Programming language, software, high level language.

1. INTRODUCTION:

The concept of compilers was introduced by an American Computer Scientist Grace Brewster Murray Hopper in 1952, for A-0 programming Language. The FORTRAN Team which was led by John Backus is credited for introducing first complete compiler in 1957. A compiler converts the source code into binary instructions for architecture of processor by making it less portable. A cross compiler can generate binary code for user machine which has different processor with code compilation. It is a computer software that transforms computer code which is written in one language into another programming language. They are similar to translators which support digital devices, primarily computers. It is likely to perform many operations such as pre-processing, lexical analysis, parsing, semantic analysis (syntax-directed translation), conversion of input programs to an intermediate representation, code optimization and code generation.

1.1 TYPES OF COMPILER:

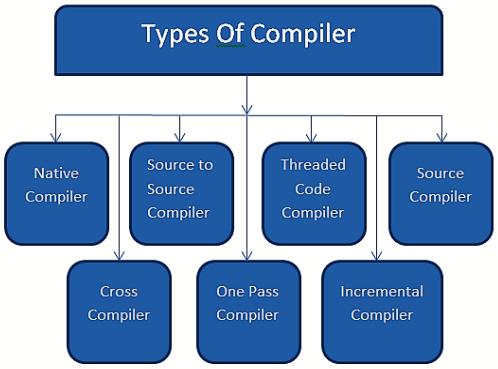

Fig 1. Types of Compiler.

In order to convert the source code into machine language code, the compiler has the types as described below:

i) NATIVE CODE: The compiler used to compile a source code for same type of platform only [1].

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

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) ONE PASS COMPILER: compiles whole process in a single pass[1].

v) THREADED CODE COMPILER: It replace string by appropriate binary code.

vi) INCREMENTAL COMPILER: It compiles only changed lines from source code and update object code.

vii) SOURCE COMPILER: It converts source code high level language in assembly language only.

1.2 WORKING OF COMPILER:

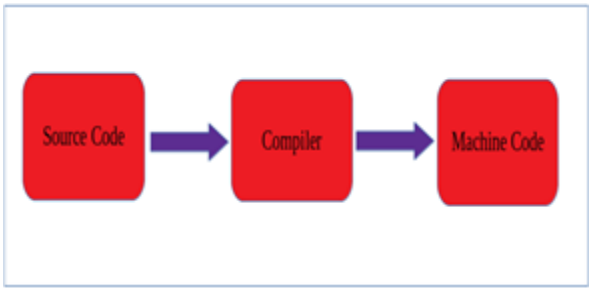

Fig 2. Working of Compiler

To know about the working of compiler, it is quite complicated to know about it. The executable code may be a sequence of machine instructions which is directly executed by CPU or it may be intermediate representation which is interpreted by virtual machine[2]. The data or the program is available in the high level language. But the computer understands only the low level language i.e language of the form 0's and 1's[2]. So, compiler is used to convert that source language into the machine language. The compiler converts that source language into the binary form which is understood by the machine and then later execution takes place.

2. PHASES OF COMPILER:

Compiler operates in phases. Each phase transform the source program from one representation to another. There are six phases of compiler as shown in Fig. Symbol table and error also interact with these phases. They are described with the help of example **position=initial + rate * 60** as follows.

1. Lexical Analysis: It is a first phase of compiler also known as scanner because it scans the source code as stream of characters and convert it into meaningful sequence of characters which is called as lexemes. It generate 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. It separates characters of source language into groups called as tokens that logically belong together[3]. The lexical analyzer of the e.g is written as **<id,1><=><id,2><+><id,3><*><60>.** Here, the characters can be mapped into following token passed.

i) Position is a lexem mapped into token <id,1>, where id is abstract symbol standing for identifier and 1 points to symbol table entry.

ii) Assignment symbol<=> is a lexem mapped into token with no attribute value.

iii) Initial is a lexem mapped into token <id,2>, where 2 points to symbol table entry.

iv) Plus is a lexem mapped into token <+>.

v) Rate is a lexem mapped into token <id,3>, where 3 points to symbol table entry.

vi) Multiplication is a lexem mapped into token <*>.

vii) 60 is a lexem mapped into token <60>.

The tokens in programming language include:

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

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

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

Punctuation symbol such as (,), {, }, paranthesis or commas.

The output of Lexical Analyser is passed to the next phase i.e Syntax analysis.

2. Syntax Analysis: It is also known as parsing as it generates parse tree from the token produced. It checks whether the expression made by tokens follow the syntax or not. The interior node of tree represent string of token i.e operation and children represents the arguments of operations [3]. The syntax tree is as follows:

The above fig tells that * has higher precedence so it should be performed first than +. The grammar is used to prove whether given statement in language is legal or not and hence is also called as context free grammar leading to formation of abstract syntax tree[3].

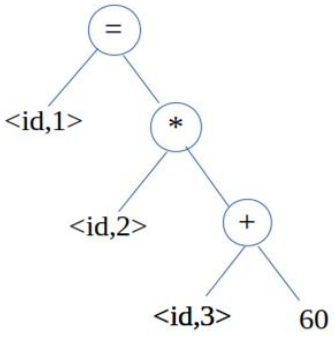

Fig 3. Syntax analysis form

3. Semantic Analysis: It uses syntax tree and information to check source program. It checks whether the parse tree to be constructed follows the rules of languages or not[3]. Type information saves it in either the syntax tree or symbol table for subsequent use during intermediate code generation. It also keeps track

to identify their types and expressions whether identifiers are declared before it is used or not[3].

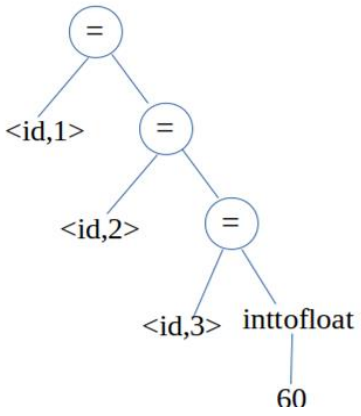

Fig 4. Semantic analysis form

4. Intermediate Code Generator: After semantic analysis, compiler generates intermediate code from the source code. It represents as abstract machine for target machine. If B is in between high level language and machine language, intermediate code which is to be generated is such as it makes easier to translate into target machine code. It may construct one or more intermediate representation which can have variety of array[4]. This representation has two important properties.

➢ It should be easy to produce.

➢ It should be easy to translate into target machine.

An intermediate form also called three address code consist of sequence of assembly like instructions with these operands per instruction is used acting each operand like a register.

The intermediate code for the given expression is as follows.

t1=inttofloat(60)

t2=id3*t1

t3=id2+t2

id1=t3

5. Code optimization: It attempts to improve intermediate code to perform better (faster) target code result. Other objectives may be desired such as shorter code consuming less power. A simple code generation algorithm followed by this phase is reasonable way to generate good target code. The optimiser can convert integer to float[3]. It must follow three rules as:

i) The o/p code must not change the meaning of program.

ii) Optimization should increase speed of program with less resource.

iii) It should itself be fast and should not delay overall compilation process.

It is written in code optimisation form as follows:

t1=id3*60.0

id1=id2+t1

6. Code Generation: It also keeps track of idea. It takes input or intermediate representation of source program and maps it into target language. It is the final phase of compiler. It generates object code of some lower level programming language[3]. Assembly language, the code generated by code generator has following meaningful minimum property to carry exact meaning of source code. If target language is machine code registers or memory location then intermediate instructions are translated into sequence of machine instruction that perform some tasks[3]. The intermediate might get translated into machine code.

LDF R2, id3

MULF R2,R2,#60.0

ADDF R1,R1,R2

STF id2,R1

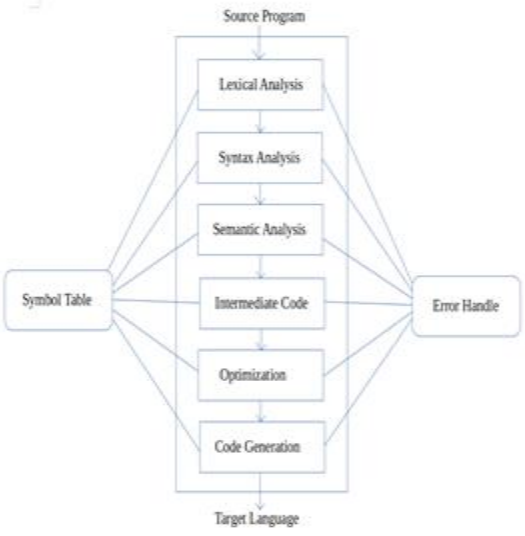

Fig 5. Phases of Compiler

3. TOOLS OF COMPILER:

A programming tool is a computer program which is used by the software developers to create, debug, maintain and to support other programs and applications. The basic tools such as source code editor or compilers are used continuously. The tools for compiler designing are

Yacc and Lax

i) Yacc: It stands for Yet Another Compiler-Compiler as it associates with writing compiler which is a computer program specially for UNIX operating system developed by Stephen C. Johnson. It is a Look Ahead Left-to-Right parser generator which generates parser trying to make syntactic sense of source code specially LALR parser. It produces only a parser for syntactic analysis. It requires external lexical analyser to perform first tokenization stage which follows parsing stage later. It is officially known as parser. It is used to analyse structure of input stream and operate of big picture by checking its syntax.

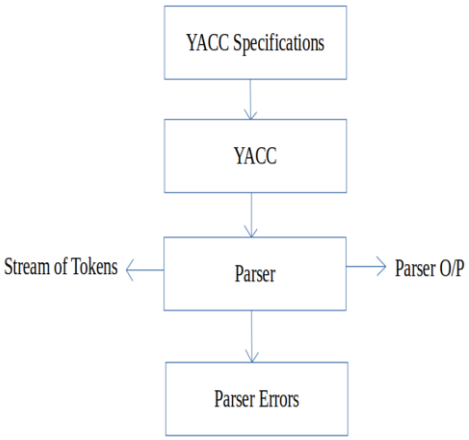

Fig 6. Yacc Tool

ii) Lex: It is a computer program that generates lexical analysers which is commonly used with yacc parser generator. It is originally written by Mike Lesk and Eric Schmidt. It reads an input stream which specifies lexical analyser and output the source code by implementing lexer in C programming language. It has a structure similar to yacc file which are divided into three sections separated vy lines containing two percent signs as follows,. The definition section define macros and imports header file written in C. The rules section associates with regular expression pattern with c statements. The c code section contains C statements and functions that are copied to generate source file. In large programs, it is convenient to place this code in a separate file linked in at compile time.

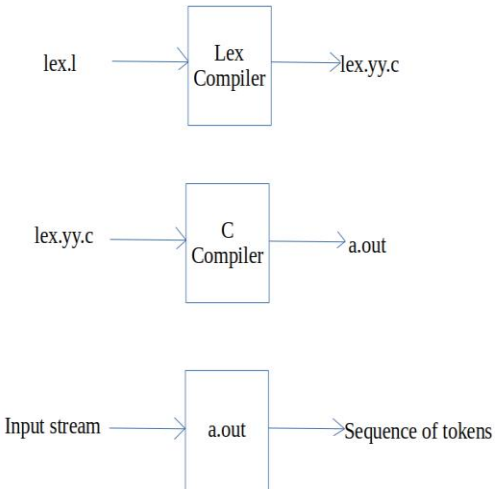

Fig 7. Lex Tool

4. FEATURES:

i) Error Handling: No one can able to write code in a single step. People make the mistakes and these mistakes are handled by the compiler. A compiler is able to run through translation process which spots the mistake in source code and it reports all the errors in the form of report. Then the programmer goes back and fixes the problem in the source code.

ii) Error checking is not perfect: The compiler will check the error but will report where it is not there and hence we can say that error checking is not perfect. This happens due to assumption made to the initial code[8].

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 optimisation: Compiler translates source code into machine code but the source code gets translated into machine code in many different ways. It makes the software to run faster and occupy less memory.

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[8].

5. ADVANTAGES AND DISADVANTAGES:

Every particular thing has 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. It has no second application or

package that user has to keep up-to date. If program is compiled for Windows, A precompiled package can run faster than interpreter compiling source code in real time.

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. For a programmer or software company which tries to get a product out to the widest possible audience, this means maintaining multiple versions of the source code for the same application.

ii) Compile Times: It compiles source code into machine code. The small programs, that many new programmers code take less amounts of time to compile, larger application suites can take more amounts of time to compile. When programmers are free but wait for the compiler to finish, this time can add up especially during the development stage.

CONCLUSION:

In this paper, we learnt the basics about compiler which includes the working about how compiler plays a role in translating the source code into the required machine language. We came to know about the phases of compilers and the types of compilers along with its characteristics and tools used for programming languages.

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.