COMPILER LECTURES

COMPUTER SCIENCE

3RD CLASS

M.SC. SAMER AL-YASSIN

2017-2018

LECTURE 1

Programming Languages :

Hierarchy of Programming Languages based on increasing machine independence includes the following:

1- Machine – level languages.
2- Assembly languages.
3- High – level or user oriented languages.
4- Problem - oriented language.

1- Machine level language: is the lowest form of computer. Each instruction in program is represented by numeric code, and numerical addresses are used throughout the program to refer to memory location in the computer's memory.

2- Assembly language: is essentially symbolic version of machine level language, each operation code is given a symbolic code such ADD for addition and MULT for multiplication.

3- A high level language such as Pascal, C.

4- A problem oriented language provides for the expression of problems in specific application or problem area .examples of such as languages are SQL for database retrieval application problem oriented language.

Using a high-level language for programming has a large impact on how fast programs can be developed. **The main reasons for this are**:

- Compared to machine language, the notation used by programming languages is closer to the way humans think about problems.
- The compiler can spot some obvious programming mistakes.
- Programs written in a high-level language tend to be shorter than equivalent Programs written in machine language.

Another advantage of using a high-level level language is that the same program can be compiled to many different machine languages and, hence, be brought to run on many different machines.

- Actually we are trying to convert the high-level language (the source-code we written) to Low-level language (Machine Language). This process involves four stages and utilizes following 'tools':

1. Pre-processor

2. Compiler

3. Assembler

4. Loader/Linker

Translator

A translator is program that takes as input a program written in a given programming language (the source program) and produce as output program in another language (the object or target program). As an important part of this translation process, the compiler reports to its user the presence of errors in the source program.

If the source language being translated is assembly language, and the object Program is machine language, the translator is called **Assembler.**

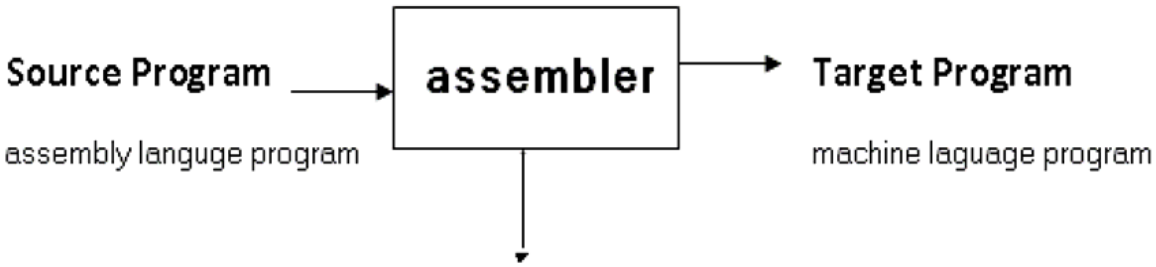


Fig (1)

A translator, which transforms a high level language such as C in to a particular computers machine or assembly language, called **Compiler.**

Another kind of translator called an **Interpreter** process an internal form of the source program and data at the same time. That is interpretation of the internal source from occurs at run time and an object program is generated Fig (2) which illustrate the interpretation process.

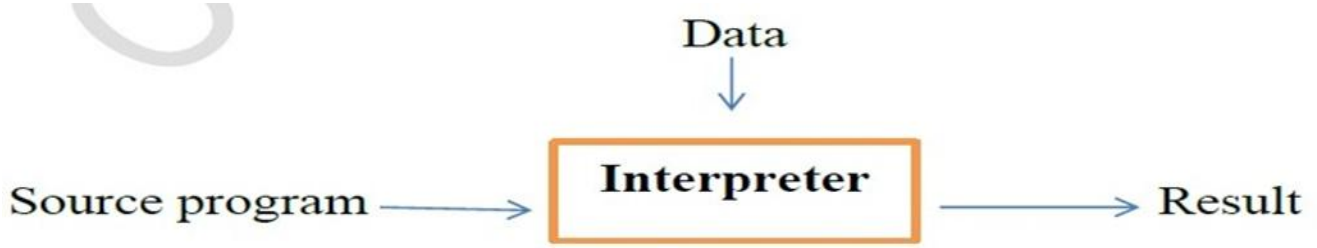


Fig (2)

Note :

The machine-language target program produced by a compiler is usually much faster than an interpreter at mapping inputs to outputs. An interpreter, however, can usually give better error diagnostics than a compiler, because it executes the source program statement by statement.

Example 1.1:

Java language processors combine compilation and interpretation, as shown in Fig. 3. A Java source program may first be compiled into an intermediate form called ***bytecodes.*** The ***bytecodes*** are then interpreted by a virtual machine. A benefit of this arrangement is that ***bytecodes*** compiled on one machine can be interpreted on another machine, perhaps across a network.

In order to achieve faster processing of inputs to outputs, some Java compilers, ***called just-in-time compilers***, translate the ***bytecodes*** into machine language immediately before they run the intermediate program to process the input.

Source program

Translator

intermediate program —»H Virtual

Machine output

input —*A*

Fig (3) : A hybrid compiler

Compiler

Is a program (translator) that reads a program written in one language, (the source language) and translates into an equivalent program in another language (the target language).

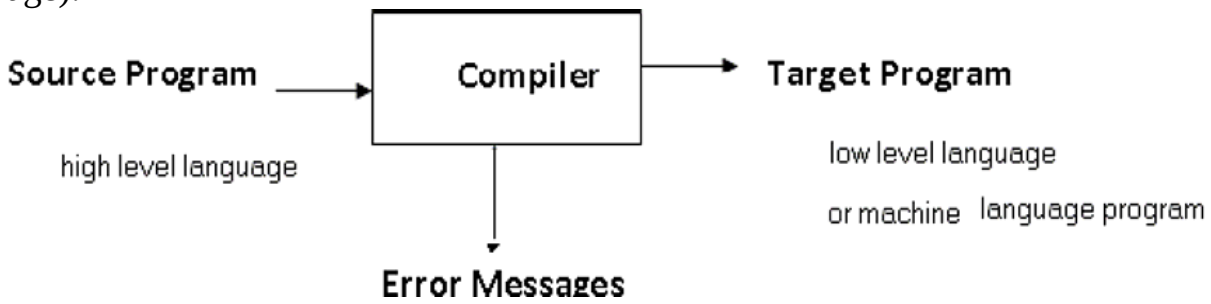


Fig (4)

The time at which the conversion of the source program to an object program occurs is called (***compile time***) the object program is executed at (*run time*).

Fig (5) illustrates the compilation process Note that the program and data are processed at different times, compile time and run time respectively.

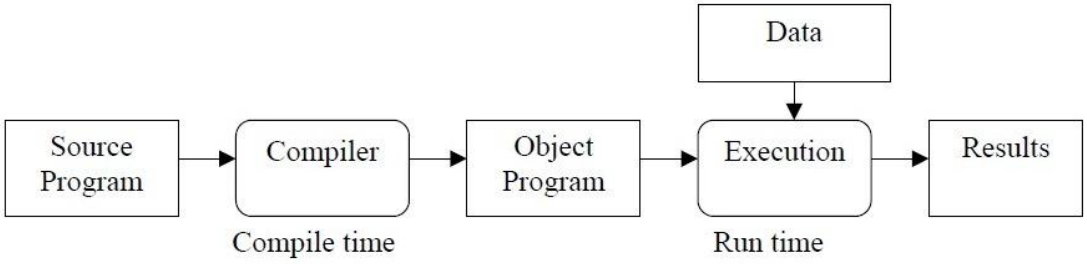


Fig (5) Compilation process

- **What is the difference between a compiler and an interpreter?**
- **What are the advantages of (a) a compiler over an interpreter**
  (b) An interpreter over a compiler?

Compiler	Interpreter
Has Lexing, parsing and type-checking	Has Lexing, parsing and type-checking
generating code from the syntax tree	the syntax tree is processed directly to evaluate expressions and execute statements
typically faster	typically slower
is often complex than writing an interpreter	is often simpler than writing a compiler
is complex than interpreter to move to a different machine	is easier to move to a different machine

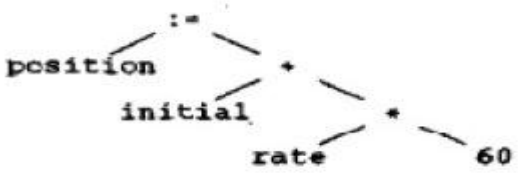
The Analysis - Synthesis model of compilation:

Up to this point we have treated a compiler as a single box that maps a source program into a semantically equivalent target program. If we open up this box a little, we see that there are two parts to this mapping: ***analysis*** and ***synthesis***.

The analysis part breaks up the source program into constituent pieces and imposes a grammatical structure on them. It then uses this structure to create an intermediate representation of the source program.

The analysis part also collects information about the source program and stores it in a data structure called a ***symbol table***, which is passed along with the intermediate representation to the synthesis part. The analysis part is often called the ***front end*** of the compiler;

The *synthesis* part constructs the desired target program from the intermediate Representation and the information in the symbol table**.** And it's also called the ***back end.*** Constructs the desired target program from the intermediate representation. During analysis, the operations implied by the source program are determined and recorded in a hierarchical structure called a tree. Often, a special kind of tree called a syntax tree is used, in which each node represents an operation and the children of a node represent the arguments of the operation. For example, a syntax tree for an assignment statement is shown in fig (6) below:



Syntax tree for position := initial + rate * 60.

Fig(6)

Compiler structure

A compiler operates in phases, each of which transforms the source program from one representation to another. A typical decomposition of a compiler is shown in fig (7).

1- lexical analysis :

The lexical analyze is the first stage of a compiler. Its main task is to read The input characters and produce as output a sequence of tokens that the Parser uses for syntax analysis.

2- syntax analysis (parsing)

The syntax analysis (or parsing) is the process of determining if a string of tokens can be generated by grammar. Every programming language has syntactic structure of well-formed programs. Syntax Analyzer takes an out of lexical analyzer and produces a large tree.

3- Semantic analysis

The semantic analysis phase checks the source program for semantics errors and gathers type information for the subsequent code-generation phase. It uses the hierarchical structure determined by the syntax-analysis phase to identify the operators and operands of expressions and statements. Semantic analyzer takes the output of syntax analyzer and produces another tree.

4- Intermediate code generation

Generate an explicit intermediate representation of the source program. This Representation should have two important properties, it should be easy to produce and easy to translate into target program.

5- Code Optimization

Attempts to improve the intermediate code so that faster running machine code will result.

6- code generation

Generates a target code consisting normally of machine code or an assemble Code. Memory locations are selected for each of the variables used by the program. Then intermediate instructions are each translated in to a sequence.