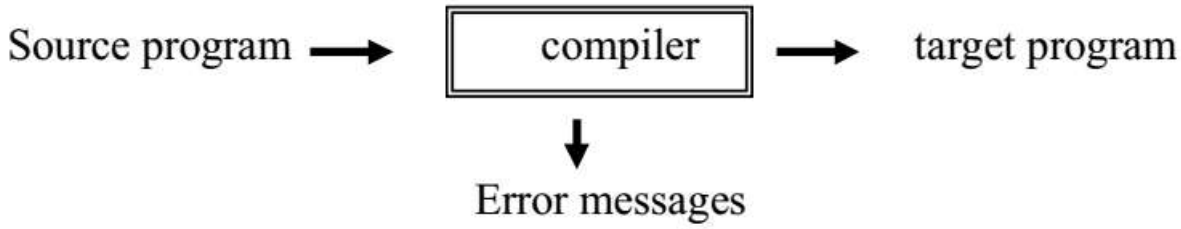
What is a compiler?

In order to reduce the complexity of designing and building computers, nearly all of these are made to execute relatively simple commands (but do so very quickly).
A program for a computer must be built by combining these very simple commands into a program in what is called machine language. Since this is a tedious and error- prone process most programming is, instead, done using a high-level programming language. This language can be very different from the machine language that the computer can execute, so some means of bridging the gap is required. This is where the compiler comes in.

A compiler translates (or compiles) a program written in a high-level programming language(source Language) that is suitable for human programmers into the low-level machine language(target language) that is required by computers. During this process, the compiler will also attempt to spot and report obvious programmer mistakes.



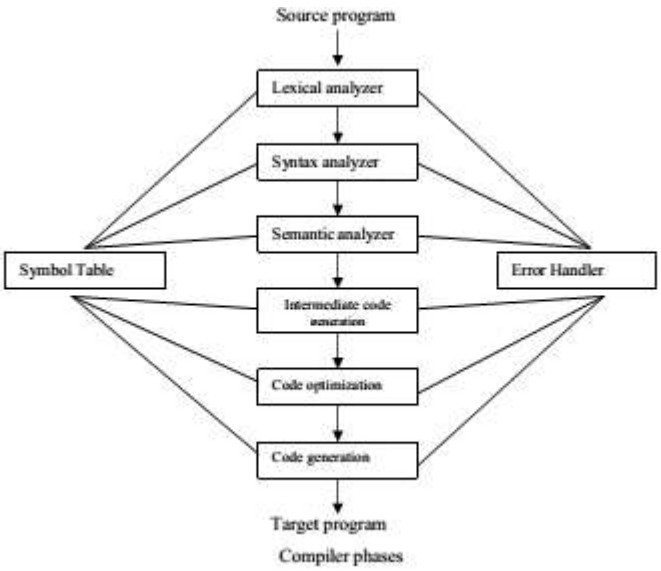
Compiled programs usually run faster than interpreter ones because the overhead of understanding and translating has already been done .However ,Interpreters are frequent easier to write than Compilers , and can more easily support interactive debugging of program .

• 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.

Compilation

Compilation refers to the compiler's process of translating a high-level language program into a low-level language program. This process is very complex; hence, from the logical as well as an implementation point of view, it is customary to partition the compilation process into several phases, which are nothing more than logically cohesive operations that input one representation of a source program and output another representation. A typical compilation, broken down into phases, is shown in the figure



Compiler phases

1. **Lexical Analyzer**: whose purpose is to separate the incoming source code into small pieces (tokens) , each representing a single atomic unit of language, for instance "keywords", "Constant "," Variable name" and "Operators".
2. **Syntax Analyzer** : whose purpose is to combine the tokens

into well-formed expressions (statements) and program and it check the syntax error

3. **Semantic Analyzer**: whose function is to determine the meaning of the source program.

4. **Intermediate Code Generator**: at this point an internal form of a program is usually created. For example:

Y=(a+b)*(c+b)

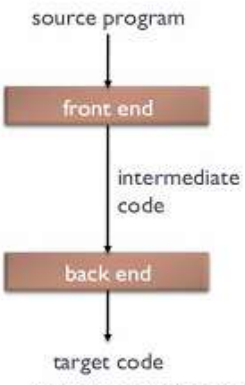
(+,a,b,t1)
(+,c,d,t2)
(*,t1,t2,t3)

5. **Code Optimizer** :Its purpose is to produce a more efficient object program (Run faster **or** take less space **or** both)

6. **Code Generator**: Finally, the transformed intermediate representation is translated into the target language.

The grouping of phases : the phases of compiler are collection into :

1. **Front-End** :It consists of those phases that depend on the *source language* and are largely independent of the *target machine* ,those include : (**lexical analysis** ,**syntax analysis** , **semantic analysis**, and **intermediate code generation**)
2. **Back-End** : Includes those phases of compiler that depend on the *target machine* and not depend on the *source language* . these include:(**code optimization phase** and **code generation phase**)



The grouping of Compiler Phases

Why learn about compilers?

Few people will ever be required to write a compiler for a general-purpose language like C, Pascal or SML. So why do most computer science institutions offer compiler courses and often make these mandatory?

Some typical reason are:

1. It is considered a topic that you should know in order to be "well-cultured" in computer science.
2. A good craftsman should know his tools, and compilers are important tools for programmers and computer scientists.
3. The techniques used for constructing a compiler are useful for other purposes as well.
4. There is a good chance that a programmer or computer scientist will need to write a compiler or interpreter for a domain-specific language.