Programming technique by using java

Second class
Teaching by salma.h.abdalla
Computer Engineering &Information Technology

References:

1-Errence W.ratt&Marvin V.Zelkowitz",Programming languages,desigen and implemention"Prentice HallInt.1996.

Lesson 1:

Machine Language

A COMPUTER IS A COMPLEX SYSTEM: consisting of many different components. But at the heart -- or the brain, if you want -- of the computer is a single component that does the actual computing. This is the **Central Processing Unit**, or CPU. In a modern desktop computer, the CPU is a single "chip" on the order of one square inch in size. The job of the CPU is to execute programs.

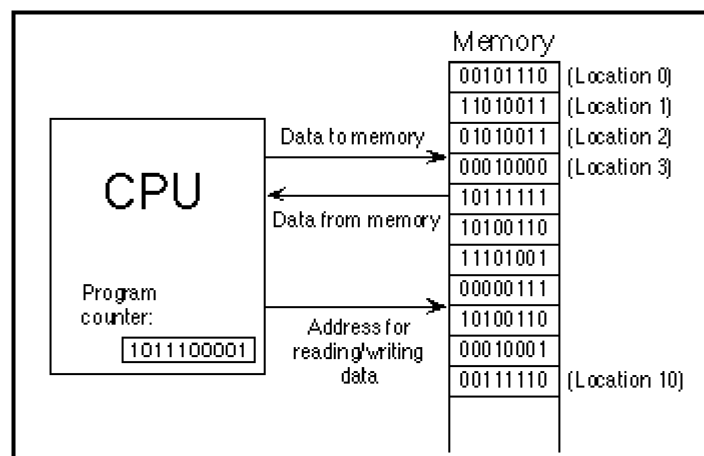
A **program**: is simply a list of unambiguous instructions meant to be followed mechanically by a computer

machine language:A computer is built to carry out instructions that are written in a very simple type of language .

CPU executes a program, that program is stored in the computer's main memory (also called the RAM or random access memory). In addition to the program, memory can also hold data that is being used or processed by the program. Main memory consists of a sequence of locations. These locations are numbered, and the sequence number of a location is called its address. An address provides a way of picking out one particular piece of information from among the millions stored in memory.

the operation of the CPU is fairly straightforward (although it is very complicated in detail). The CPU executes a program that is stored as a sequence of machine language instructions in main memory. It does this by repeatedly reading, or fetching, an instruction from memory and then carrying out, or executing, that instruction. This process -- fetch an instruction, execute it, fetch another instruction, execute it, and so on forever -- is called the fetch-and-execute cycle

So, you should understand this much about how computers work: Main memory holds machine language programs and data. These are encoded as binary numbers. The CPU fetches machine language instructions from memory one after another and executes them. It does this mechanically, without thinking about or understanding what it does -- and therefore the program it executes must be perfect, complete in all details, and unambiguous because the CPU can do nothing but execute it exactly as written. Here is a schematic view of this first-stage understanding of the computer:



Programming technique by using java

Second class
Teaching by salma.h.abdalla
Computer Engineering &Information Technology

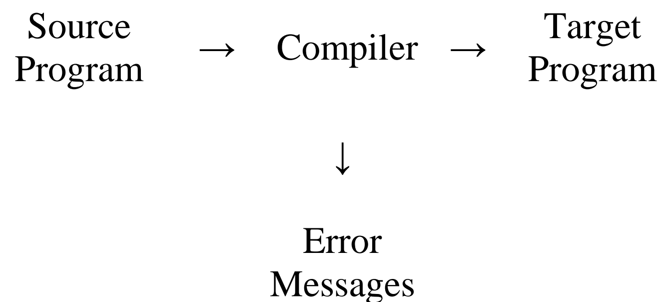
Lesson 2

Compilers

Translator:

A translator is program that converts a source program into an (object program). The source program is written in a (source language) and the object program belong to an (object language).

The translation process should also report the presence of errors in the source program.



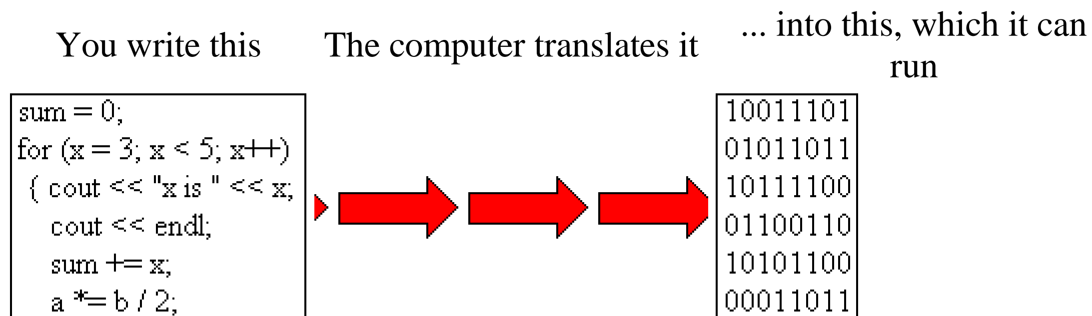
There are two parts of compilation.

1- The analysis part breaks up the source program into constant piece and creates an intermediate representation of the source program.

2- The synthesis part constructs the desired target program from the intermediate representation.

Compilers and Interpreters

Compilers :These are programs which translate computer programs from high-level languages such as Pascal, C++, Java or JavaScript into the raw 1s and 0s which the computer can understand, but the human programmers cannot:



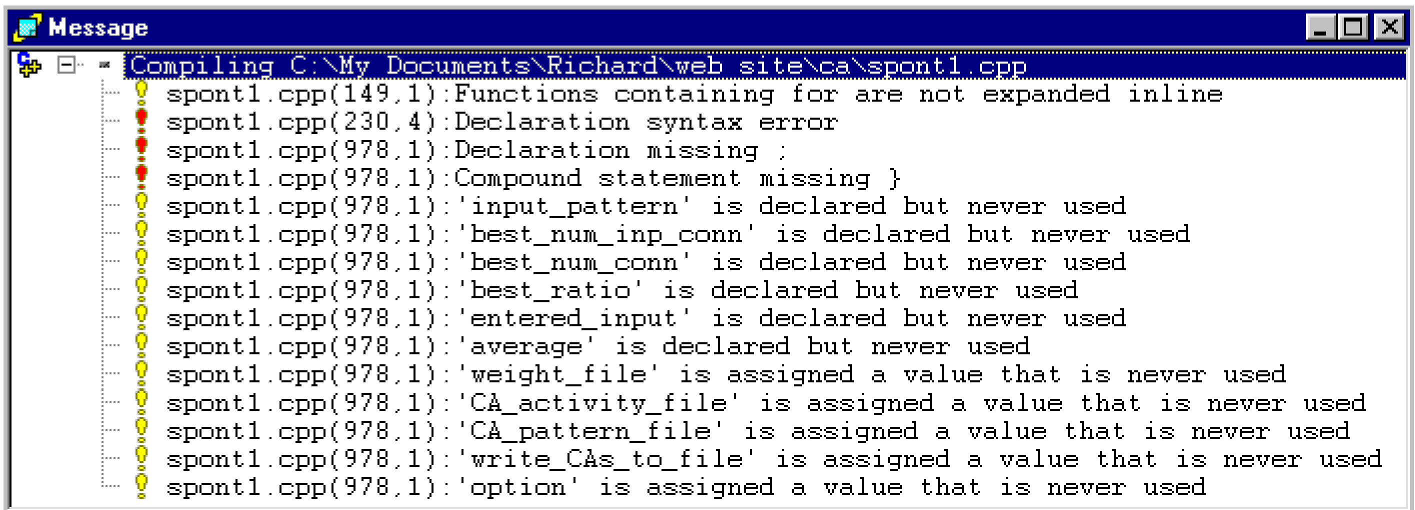
Compilers

Compilers were the first sort of translator program to be written. The idea is simple: You write the program, then hand it to the compiler which translates it. Then you run the result.

Task of compiler

1- The compiler takes the file that you have written and produces another file from it. In the case of Pascal programs, for instance, you might write a program called **myProg.pas** and the Pascal compiler would translate it into the file **myProg.exe** which you could then run. If you tried to examine the contents of **myProg.exe** using, say, a text editor, then it would just appear as gobbledy-gook.

2- The compiler has another task apart from translating your program. It also checks it to make sure that it is grammatically correct. Only when it is sure that there are no grammatical errors does it do the translation. Any errors that the compiler detects are called compile-time errors or syntax errors. If it finds so much as one syntax error, it stops compiling and reports the error to you. Here is an example of the C++ compiler reporting a whole list of errors:



Most "serious" languages are compiled, including Pascal, C++ and Ada.

Interpreters

interpreter :is a program that translates a high-level language into a low-level one, but it does it at the moment the program is run. You write the program using a text editor or something similar, and then instruct the interpreter to run the program. It takes the program, one line at a time, and translates each line before running it: It translates the first line and runs it, then translates the second line and runs it etc. The interpreter has no "memory" for the translated lines, so if it comes across lines of the program within a loop, it must translate them afresh every time that particular line runs. Consider this simple Basic program:

```
10 FOR COUNT = 1 TO 1000
20   PRINT COUNT * COUNT
30 NEXT COUNT
```

Line 20 of the program displays the square of the value stored in `COUNT` and this line has to be carried out 1000 times. The interpreter must also translate that line 1000 times, which is clearly an inefficient process. However, interpreted languages do have their uses, as we will see in a later section.

Examples of interpreted languages are Basic, JavaScript and LISP.

. The main **advantages** of compilers are as follows:

- They produce programs which run quickly.
- They can spot syntax errors while the program is being compiled (i.e. you are informed of any grammatical errors before you try to run the program). However, this does not mean that a program that compiles correctly is error-free!

The main **advantages** of interpreters are as follows:

- There is no lengthy "compile time", i.e. you do not have to wait between writing a program and running it, for it to compile. As soon as you have written a program, you can run it.
- They tend to be more "portable", which means that they will run on a greater variety of machines. This is because each machine can have its own interpreter for that language. For instance, the version of the BASIC interpreter for the PDP series computers is different from the QBasic program for personal computers, as they run on different pieces of hardware, but programs written in BASIC are identical from the user's point of view.

How does a compiler work?

Compiling a program takes several stages of processing, which I have outlined below. The principles which are explained below also apply to interpreters, with the exception that the interpreters translate each program one line at a time before running it, and then moving on to the next line. The process may be summarised in this diagram:

`Tokenising` → `Syntax analysis` → `Semantic analysis` → `Translation`

Tokenising

This part of the process is also sometimes called *Lexical Analysis*. It involves turning the program from a series of characters into a series of *tokens* that represent the building blocks of a program. The tokens are keywords of the language i.e. important words such as *if*, *print* or *repeat*), variable names and mathematical operators (+, *, brackets etc.).

The tokeniser takes each of the characters in turn, and as soon as it recognises a legitimate token, it reports it to the next stage of the process. Each token has a label and a type, so a variable "count" in a program would have the label corresponding to its name ("count") and the type "variable name". The tokeniser is also responsible for ignoring comments in the

program - these are words and phrases inserted purely for the benefit of any human reading the program, and they have no function.

Syntax Analysis

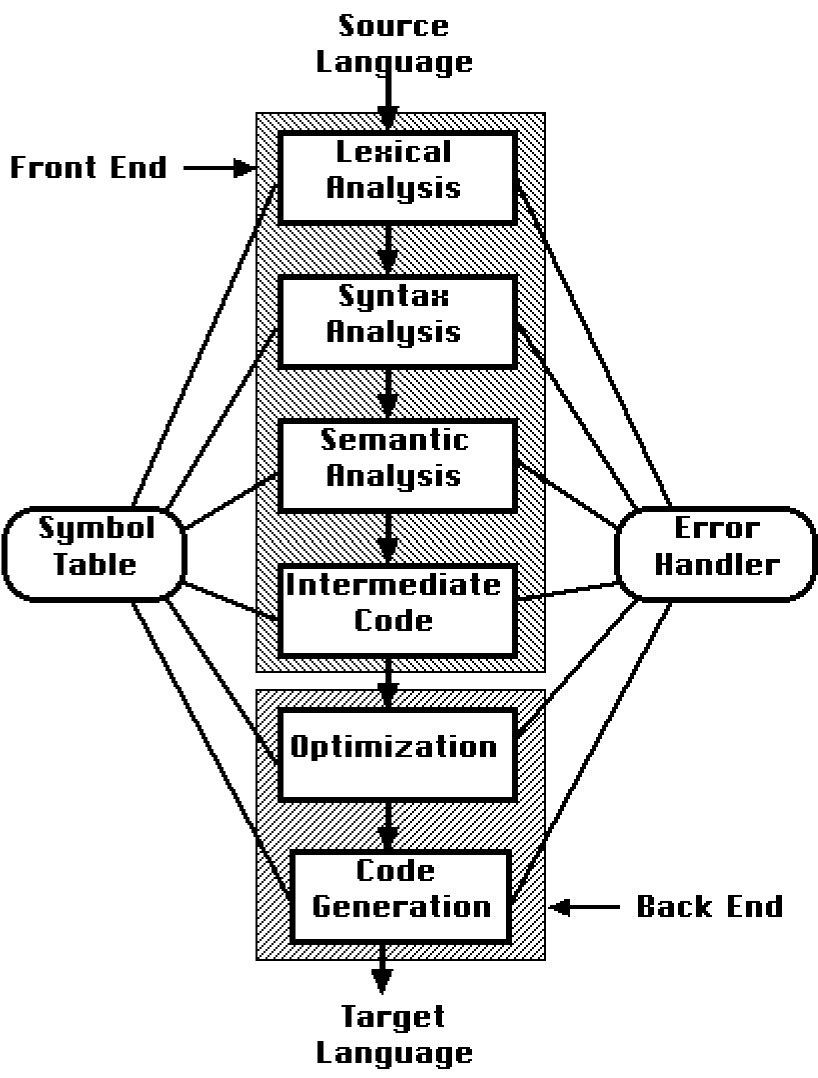
Syntax means "grammar" and the syntax analyser in a compiler checks that the right tokens appear in the right order to make grammatically correct instructions. For instance, in C++, the instruction `xyz++;` is syntactically correct, but the instruction `+;xyz+` is not - the order of the tokens is wrong.

Semantic Analysis

The word "semantics" refers to *meaning*, and the semantic analyser checks the meaning of the program. This refers to aspects such as whether the variables have been declared, e.g. `xyz++;` may be syntactically correct, but if the variable `xyz` has not been declared, then it is semantically incorrect!

The semantic analyser checks not only variable declarations and scope, but whether the program has entered or left loops, or subroutines, whether classes are accessed correctly etc.

Translation



Compiler Phases

1- lexical Analysis

is the interface between the source program and the compiler , reads the source program one character and write it into a sequence of atoic unit called (token)

Token :-represents a sequence of characters can be treated as a single logical entity.

Type of Token

1. **1- Constant : 1,2,3,4.5 ,5E6**
2. **2- Identifers :**

Objects and Object-oriented Programming

structured programming. The structured programming approach to program design was based on the following advice: To solve a large problem, break the problem into several pieces and work on each piece separately; to solve each piece, treat it as a new problem which can itself be broken down into smaller problems eventually, you will work your way down to problems that can be solved directly, without further decomposition. This approach is called top-down programming