SLIIT ACADEMY

FCIT – Semester 1

PROGRAMMING SKILLS -I

PROGRAMMING LANGUAGES

Ms. Ovini Seneviratne

Course Outline

- Module name: Programming Skills - I
- Credits: 3.5
- Duration: One Semester
- Delivery method
  - Lectures : 2 hours/week
  - Tutorial : 1 hour/week
  - Self study : 2 hours/week

Assessment Criteria

- Mid Term Examination : 30%
- Assignment (in class) : 10%
- Final Examination : 60%

To pass this module, need to obtain a pass mark in both "Continuous Assessments" and "End of the Semester Examination"

Programming Languages

LECTURE 01

Learning Outcomes

End of this lecture you will be able to learn ,

LO1 : To identify the use of programming languages

LO2 : To Discuss different generations of computer languages.

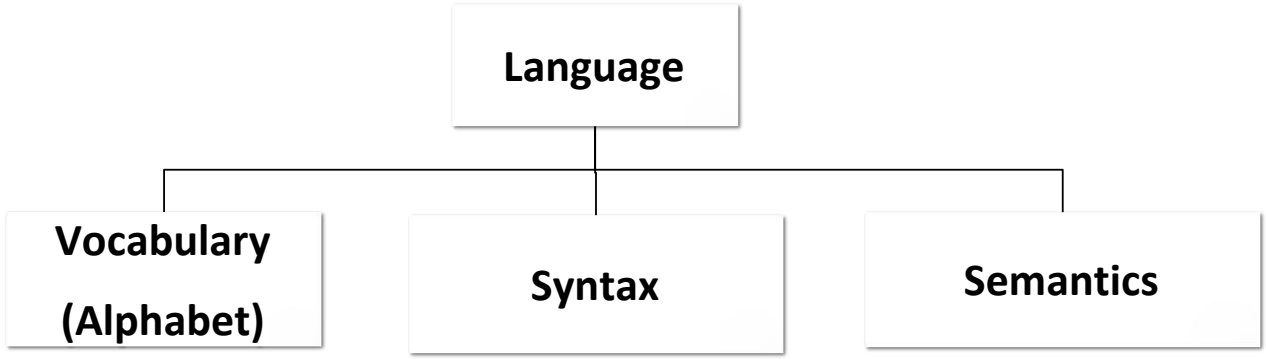
LO3 : To Discuss the role of translators

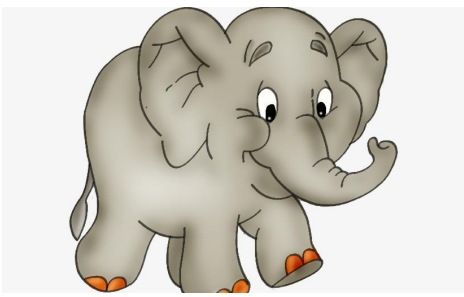
LO4 : To Compare of different levels/generations of languages.

What is a Programming Language?

- Vocabulary + Set of grammatical rules
- For instructing a computer or computing device
- To perform specific tasks.

Components of Programming Languages





Syntax and Semantics

Consider the following sentences

1. Snake is a mammal.
2. Snake not mammal is.
3. Snake is a reptile.

Which of the above statements are correct???

Generations and Levels

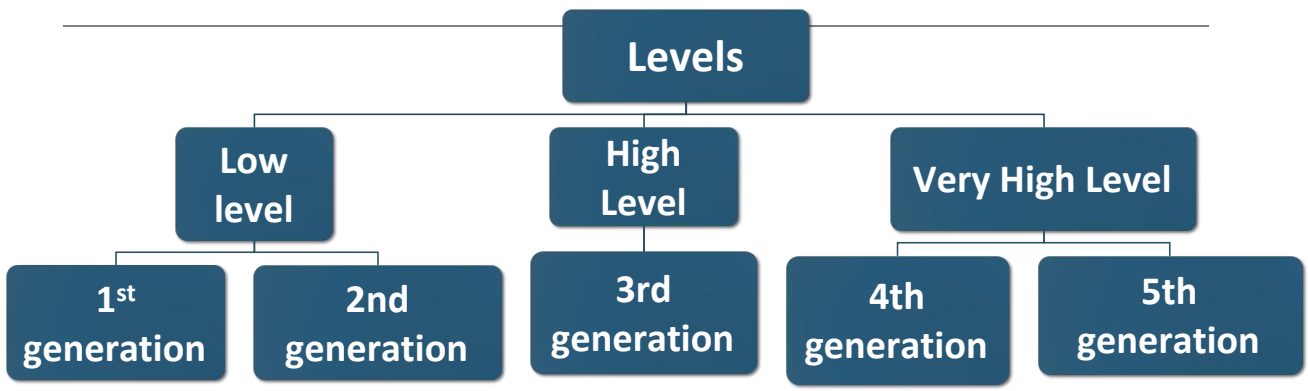
Languages are grouped into generations

- based on when they were first conceived.
- There are five such generations.

Languages are grouped by levels

- based on how much they are close to the machine language
- There are three such levels

Generations and Levels



Language Generations

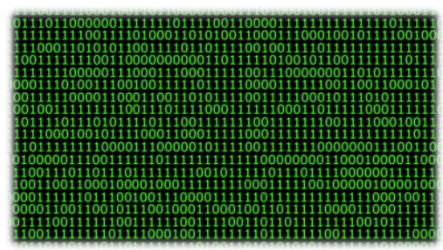
Language	Generation	Level
Machine Language	First	Low level
Assembly Languages (Mnemonics)	Second	Low level
Procedural Languages (FORTRAN, COBOL, C, C++, Visual Basic, Java)	Third	High level
Query Languages (SQL,Post Script)	Fourth	Very high level
Artificial Languages (Mercury, OPS5, and Prolog)	Fifth	Very high level

Machine Language - 1st Generation

- The most basic type of computer language.
- Different types of hardware use different machine code.

Example: IBM computers use different machine language than Apple computers.

- Hardware dependent
- Easy to understand and execute.

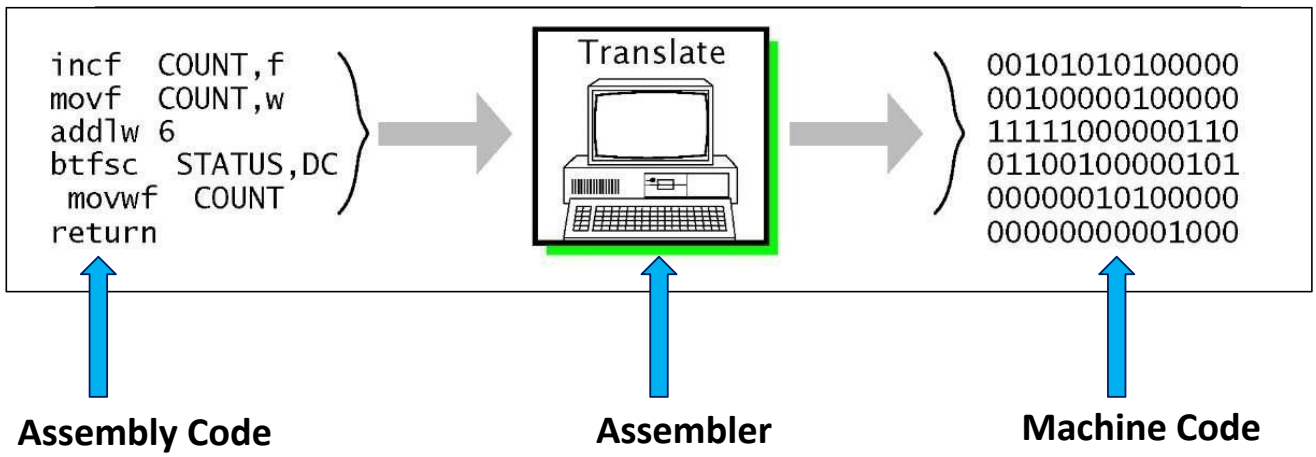


Assembly Language - 2nd Generation

- Somewhat easier to work with than machine languages.
- Use **cryptic English-like phrases** to represent strings of numbers.
- Each numeric instruction is assigned a short name called **mnemonics**.

  Example : 28 – LOAD 69 - ADD
- The instructions are translated into object code, using a translator called an **assembler**.

Assembly Language - Translation Process



Assembly Language - Translation Process

```
;CLEAR SCREEN USING BIOS
CLR: MOV AX,0600H        ;SCROLL SCREEN
     MOV BH,30           ;COLOUR
     MOV CX,0000         ;FROM
     MOV DX,184FH        ;TO 24,79
     INT 10H             ;CALL BIOS;
;INPUTTING OF A STRING
KEY: MOV AH,0AH          ;INPUT REQUEST
     LEA DX,BUFFER       ;POINT TO BUFFER WHERE STRING STORED
     INT 21H             ;CALL DOS
     RET                 ;RETURN FROM SUBROUTINE TO MAIN PROGRAM;
; DISPLAY STRING TO SCREEN
SCR: MOV AH,09           ;DISPLAY REQUEST
     LEA DX,STRING       ;POINT TO STRING
     INT 21H             ;CALL DOS
     RET                 ;RETURN FROM THIS SUBROUTINE;
```

Assembly code

Assembler

```
0001010010110101010101010101010100010
1110110101010101010101110010100010110
0010100101010010111101111010111010
1001010010110101010101010101010110110
0110100100110010111101111010100010
0001000101011101010101000101010111010
1010100101010010101101110101110101011
0001010010110101010101010101010100010
```

Object code

Procedural Language–3rd Generation

- The first to use **true English-like phrasing**.
- 3GLs are **portable**, meaning the object code created for one type of system can be translated for use on a different type of system.
- **Independent of hardware** - Can use Procedural Languages to write programs for many kinds of computers.
- Require translators
  - Compiler / Interpreter

Procedural Language– 3rd Generation

- The program code is written as a sequence of instructions.
- User has to specify **"what to do"** and also **"how to do"** (step by step procedure).

Example

```
int main ()

{
      int num1=10;
      int num2=20;
      int num3=30;
      int sum = 0;

      sum = num1 + num3;

      cout<<"Sum =" << sum<< endl;

      return 0;
}
```

Non - Procedural Languages - 4 th Generation

- Non-procedural - concentrate on **what you want to do** rather how you are going to do it.
- Consist of statements like statements in a human language.
- Commonly used in database programming and scripts.
- Example: **SQL , PostScript , Relational database-oriented languages , VisualAge, Perl, Maple , Mathematica , Python , Ruby**
- 4GLs may use a text-based environment (like a 3GL) or may allow the programmer to work in a visual environment, using graphical tools.

Fifth - Generation Languages

- Based around solving problems using constraints given to the program, rather than using an algorithm written by a programmer.
- Contain visual tools to help develop a program.
- Used mainly in artificial – intelligence, knowledgebase systems, natural language processing.
- Ex: Mercury, OPS5, and Prolog

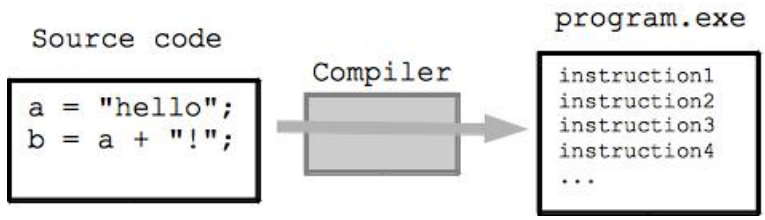
Translators

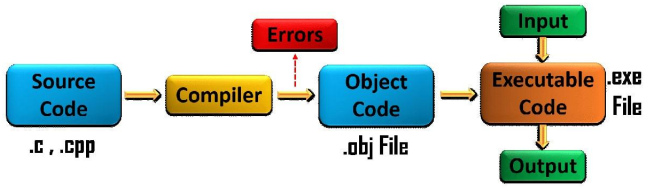
- Analyses the program (source code) and **writes an equivalent machine language program** (object code).
- Three kind of translators,
  - Compiler
  - Interpreter
  - Assembler
- Depends on two things; **the programming language the program was created with** and **the processor it is being translated for**

Compiler

- Compilers are translators **used before a program is run.**
- To produce a complete machine language program (stored inside disk).
- The entire program is first translated to machine code, and this code is then executed.
- Takes large amount of time to analyze the source code
- Each language requires its own compiler.

  Example: Pascal compiler and a Java compiler





Main Functions of a Compiler

- Checks for **syntax errors** in a program, for statements which do not conform to the grammatical rules of the language.
- Produces a listing of the program, indicating errors, if any.
- If there are no syntax errors, it generates machine code equivalent to the given program.

Interpreters

- As each statement of the high-level program is encountered it is translated and executed.
- **Translation is done on the fly.**
- Takes less amount of time to analyze the source code.
- No object code is created in the process, so no executable program file is stored on the disk for later use.

Summary of Translators

Interpreter	Compiler	Assembler
Translates high level language into machine code.	Translates high level language into machine code.	Translates assembly language into machine code.
Translate one statement at a time.	Translate whole code at once.	
Needed every time you run the program.	Only need once to create the object code.	
Once an error is encountered it is notified and no further code is scanned	Generates error report after translation of entire code.	
Takes less amount of time to analyze the source code but the overall execution time is comparatively high.	Takes high amount of time to analyze the source code but the overall execution time is less.	Runs quickly since the translation is in between two low level languages.

Debugging

- **Debugging** is a process of **identifying , analyzing and fixing errors**
- There are 3 types of errors
  - **Syntax errors**
  - **Logical errors**
  - **Run-time errors**

Syntax Errors

- When the code given does not follow the syntax rules of the programming language.
- With compiled languages, you will run into any syntax errors at compile time, and they will have to be corrected before the program can run.
- For interpreted languages, a syntax error would pop up during run time.
- Examples include:

  misspelling a statement, such as writing pint instead of print

  missing brackets, like opening a bracket, but not closing it.

Logical Errors

- The program **can run but does not do what it is expected to do** .
- Logic errors can be caused by the programmer:

  - incorrectly using logical operators.

  - incorrectly using Boolean operators.

  - unintentionally using the same variable name at different points in the program for different purposes
- Does not usually stop a program from running. The program will run, but not function as expected.

Run - time Errors

- An error that takes place **during the running of a program** .
- An example is writing a program that tries to access the sixth item in an **array** that only contains five items.
- A runtime error is likely to crash the program.

  Ex – Division by zero

  Never ending loops

Summary

- Levels & Generations
- Program Translators
- Debugging