

More Quick Notes, Telegram: @campus_handout / https://t.me/campus_handout

Chapter One: Introduction

- Programming is a skill/ knowledge of making the computer perform the required operation or task
- learning computer programming is learning how to talk to the computer
- **Programming Language**: a set different category of **written symbols** that **instruct computer hardware** to perform specified operations required by the designer
- **Skills** needed to be a **programmer**:
 - **Programming Language Skill**
 - **Problem Solving Skill**
 - **Algorithm Development**
 - ✓ sequence of simple and **human understandable set of instructions** showing the **step of solving the problem**
 - ✓ steps should **not be dependent on any programming language or machine**
- **syntax**
 - sets of rules that govern the symbols used in a programming language
 - determine how the programmer can make the computer hardware to perform a specific operation

1.1 Generations of programming language

- Programming languages are categorized into five generations
- The higher the level of a language, the easier it is to understand and use by programmers
- Programs written in high level languages need to be translated into machine language in order to be executed
- They are categorized into two broad categories
 - Low level languages
 - ✓ machine specific or dependent
 - ✓ first and second generation languages
 - High level languages
 - ✓ machine independent
 - ✓ can run on variety of computers
 - ✓ Third, fourth and fifth generation languages
 - ✓ Fourth & fifth - very high level languages
 - ✓ Example: COBOL, BASIC

1) First Generation (Machine languages, 1940's)

- Difficult to write
- Dependent on machine languages of the specific computer being used
- executed by the computer without the need for a translator
- execute very quickly and use memory very efficiently
- allow the programmer to interact directly with the hardware
- more powerful in utilizing resources of the computer

2) Second Generation (Assembly languages, early 1950's)

- Uses symbolic names for operations and storage locations

- Assembler(system program) translates a program written in assembly language to machine language
- not portable. i.e., different computer architectures have their own machine and assembly languages
- highly used in system software development

3) Third Generation (High level languages, 1950's to 1970's)

- Uses English like instructions and mathematicians were able to define variables with statements
- easier to use than assembly language
- need to be translated into machine language
- All third generation programming languages are procedural languages: programmer is expected to specify what is required and how to perform it

4) Fourth Generation (since late 1970's)

- simple, English like syntax rules
- non-procedural languages: easier to write, but you have less control over how each task is actually performed
- programmer is not required to write traditional programming logic, he/she concentrate on defining the input and output rather than the program steps required
- have a minimum number of syntax rules; saves time
- Example: structured query languages (SQL), report generators, application generators and graphics languages

5) Fifth Generation (1990's)

- used in artificial intelligence (AI) and expert systems; also used for accessing databases
- Are natural languages whose instruction closely resembles human speech

- require very powerful hardware and software because of complexity involved in interpreting commands in human language

1.2 Overview of Computers and Computer Organization

- computer divided into six logical units or sections

1) Input Unit

- obtains information from various input devices and places this information at the disposal of the other units

2) Output Unit

- takes information that has been processed by the computer and places it on various output devices to make the information available for use outside the computer

3) Memory unit

- retains information that has been entered through the input unit
- also retains processed information until that information can be placed on output devices

4) Central Processing Unit (CPU)

- is the computer's coordinator
- responsible for supervising the operations of the other sections

5) Arithmetic and Logic unit:

- part found inside the CPU

- responsible for performing mathematical calculations/operations & comparisons

6) Secondary Storage device

- A Programs/data used by other units are placed on it

1.3 The evolution of Operating Systems

- single-user batch processing
 - ✓ runs a single program/only one job at a time while processing data in groups or batches
 - ✓ It rarely utilizes the computer's resources
- Multiprogramming
 - ✓ many jobs or tasks from different programs or applications could be made to share the resources of the computer
 - ✓ involving the “simultaneous” operation of many jobs on the computer
 - ✓ achieve better utilization
- timesharing operating systems:
 - ✓ It is special case of multiprogramming
 - ✓ users access the computers through terminals
 - ✓ computer does not run the users job paralleling, but shares the CPU's time

1.4 Major Programming Paradigms

- Are features or properties observed in the development of programming languages

A.Procedural Programming

- based upon the concept of procedure call

- Possible benefits:
 - ✓ the ability to **re-use the same code** (function or procedure) at different places
 - ✓ an **easier way to keep track of program flow** than a collection of “GO TO” or “JUMP” statements
- better choice than simple sequential programming in many situations which involve moderate complexity or which require significant ease of maintainability

B. Structured Programming

- Process of writing a **program in small, independent parts**
- **easier to control** a program's development and to **design and test** its **individual component parts**
- Structured programs are **built up from units called modules**, which normally correspond to single procedures or functions
- It is subset or sub discipline of procedural programming
- most famous for **removing or reducing reliance on the GO TO statement**

C. Object-Oriented Programming

- Here a computer **program is composed of a collection of individual units, or objects** as opposed to traditional list of instructions to the computer
- give **more flexibility, easing changes to programs**
- **simpler to develop and maintain**

1.5 Problem solving process and software engineering

Software engineering

- Software engineering is the profession that creates and maintains software applications
- **method used in solving problems** in computer science and/or information systems is called the **software development life cycle**. It **has the following components**:
 - preliminary investigation
 - analysis
 - design
 - implementation
 - testing and maintenance

Problem Solving

- is the **process of transforming the description of a problem into the solution** by using our knowledge of the problem domain and by relying on our ability to select and use appropriate problem-solving strategies, techniques, and tools
- **A problem** is
 - ✓ an **undesirable situation** that **prevents** the organization from fully achieving its purpose, goals and objectives
 - ✓ a **gap** between the existing and the desired situation where problem solving will try to fill this gap
- two **approaches of problem solving**:
 - **Top down design**
 - based on the concept that the structure of the problem should determine the structure of the solution and what should be done in lower level

- disintegrate a larger problem into more smaller and manageable problems to narrow the problem domain
- Bottom up design
 - reverse process where the lowest level component are built first and the system builds up from the bottom until the whole process is finally completed

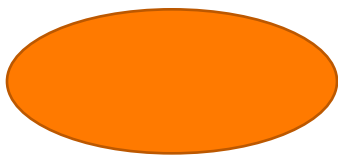
Basic Program development tips

- The program we design in any programming language need to be:
 - ✓ Reliable
 - always do what it is expected to do and handle all types of exception.
 - ✓ Maintainable
 - could be modified and upgraded when the need arises
 - ✓ Portable
 - possible to adapt the software written for one type of computer to another with minimum modification
 - ✓ Efficient
 - designed to make optimal use of time, space and other resources of the computer

Algorithm designing and modeling the logic (using flow chart)

- algorithm:
 - ✓ a solution to a problem

- ✓ describes the **sequence of steps** to be performed for the problem to be solved
- ✓ **finite set of well-defined instructions** for accomplishing some task which, given an initial state, will terminate in a corresponding recognizable end-state
- ✓ **algorithm should be:**
 - Precise and unambiguous
 - Simple
 - Correct
 - Efficient
- **flow chart:**
 - a **graphical way of expressing the steps** needed to solve a problem
 - a **schematic (diagrammatic description)** representation of a process
 - Basic flowcharting symbols are:



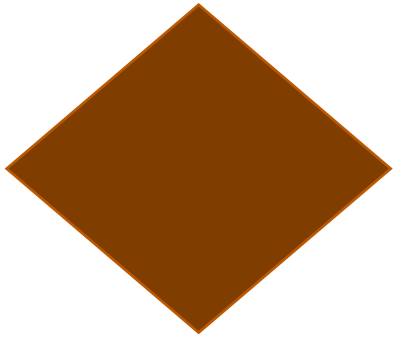
Terminal point



Input/output



Process



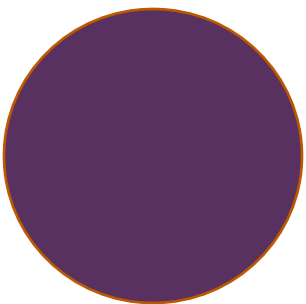
Decision



Flow line

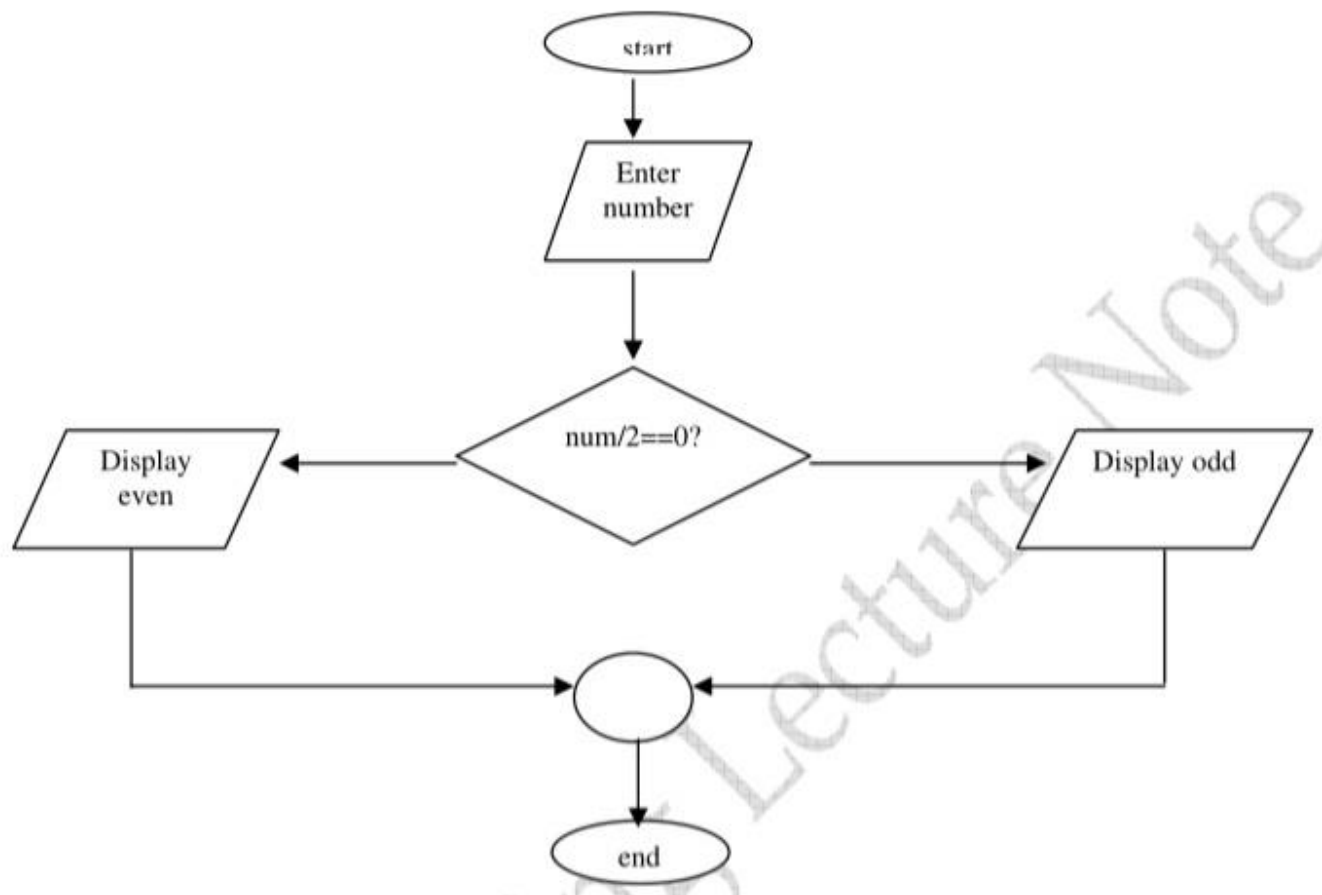


Inter-page connector



On-page connector

e.g: to check weather a number is positive or negative.



Compilers and Interpreters

- Any program written in a language other than machine language needs to be translated to machine language
 - ✓ The set of instructions that do this task are known as **translators**
- There are different kinds of translator software, among which translator and interpreters are of interest for most programmers
 - **Compilers**

- changes or translates the **whole source code** into executable machine code/object code
- E.g. C++, Pascal, FORTRAN, etc

➤ Interpreters

- translates a single high level statement and executes it and then goes to the next high level language line etc
- translate **line by line**
- E.g. QBASIC, Lisp etc

Mechanics of Creating a program (C++)

- C++ programs typically go through **five phases** to be executed:

➤ Edit

- accomplished with an **editor program**
- **Typing** C++ statements with the editor and making corrections if necessary
- programs **source file** is then **stored** on secondary storage device such as a disk with a **“.cpp” file name**

➤ Preprocess

- preprocessor program **executes automatically before the compiler's translation phase** begins
- The C++ preprocessor **obeys command** called **preprocessor directives**, which indicate that certain manipulations are to be performed on the program **before compilation**
- goes over the program text and carries out the instructions specified by the preprocessor directives

- preprocessor directives (e.g., #include)
- The preprocessor is invoked by the compiler before the program is converted to machine language
- The result is a modified program text which no longer contains any directives.

➤ Compile

- C++ compiler translates the program code
- C++ programs typically contain references to functions and data defined elsewhere, such as in the standard libraries
- The outcome may be incomplete due to the program referring to library routines which are not defined as a part of the program
- The object code produced by the C++ compiler typically contains “holes” due to these missing parts

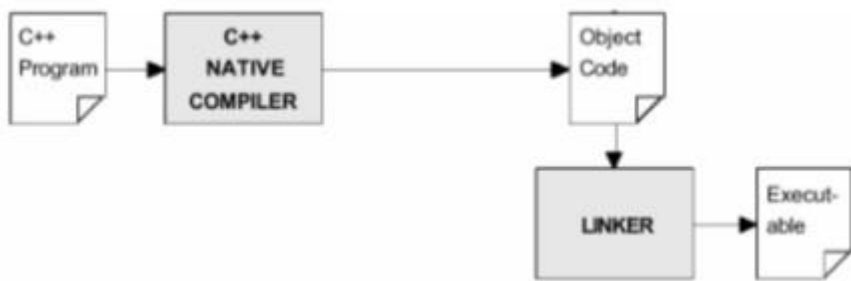
➤ Linking

- A linker links the object code with the code for the missing function to produce an executable image (with no missing pieces)
- the linker completes the object code by linking it with the object code of any library modules that the program may have referred to
- The final result is an executable file

➤ Loading

- the loader takes the executable file from disk and transfers it to memory
- Additional components from shared libraries that support the program are also loaded

- Finally, the computer, under the control of its CPU, executes the program



- ❖ In practice all these steps are usually invoked by a single command and the user will not even see the intermediate files generated