Introduction to Computational Science and Basics of Computer Programming (CDSC 601)

Addisu G. Semie, PhD
Asst. Prof.,Computational Data Science Program,
Addis Ababa University
Email: addisu.semie@aau.edu.et

Course Outline For Part II

Chapter 1: Introduction

- Introduction to computer system
- The Von Newman Architecture
- Operating systems
- Programming language software
  - Translator (Compiler, interpreter)
  - linker
  - editor
- Algorithm Design
  - Flow chart
  - pseudo-code
- Fortran programming language
  - History
  - Versions, their similarities, differences

Chapter 2
Basic elements of Fortran 90/95

- Character set
- Structure of Fortran programming
- Basic elements of programming
  - Identifiers
  - Constants
  - Data types
  - Fortran Operators
  - Fortran expression
- Precedence rules of Operators
- Line Discipline
- Fortran I/O statements
  - List Directed Output statements
  - Sample Fortran program
  - Formatted Input statements
  - File I/O statements

Chapter 3
Fortran Control statements

- Control statement
- GOTO statement
- Branching statements
  - The IF .... ENDIF statement
  - The SELECT CASE statement
- Looping statements
  - The DO...END DO statement
  - The counting DO .... END DO statement
  - The implied DO loop
- The CYCLE and EXIT statement

Chapter 4
Fortran subprogram

- Basics of Fortran subprogram
- Program design strategy
- Why we need to modularize?
- Fortran Function
  - Fortran Intrinsic functions
  - Fortran External functions
- Fortran Subroutines
- parameter passing technique
- Internal Procedure Code
- Module Procedure Code
- External Procedure Code

Chapter 5
Array in FORTRAN Programming

- Array basics
- Array Declaration
- Using Array elements
- Array Initialization
- Multidimensional Array
- Multidimensional Array Initialization
- Using whole arrays and array subset
- Array I/O
- Arrays As Arguments in FORTRAN subroutine and function

Chapter 6
File processing

- How to create logical file (Opening file)
- How to read and write file
  - Binary file
  - Text file

References

Stephen J. Chapman (1998), Introduction to FORTRAN 90/95, 1st Edition, TATA McGraw-Hill, new Delhi

Chapter 1: Introduction

- **Outline**
  - Introduction to computer system
  - The Von Newman Architecture
  - Operating systems
  - Programming language software
    - *Translator (Compiler, interpreter)*
    - *linker*
    - *editor*
  - Algorithm Design
    - *Flow chart*
    - *pseudo-code*
  - Fortran programming language
    - *History*
    - *Versions, their similarities, differences*

Introduction to computer system

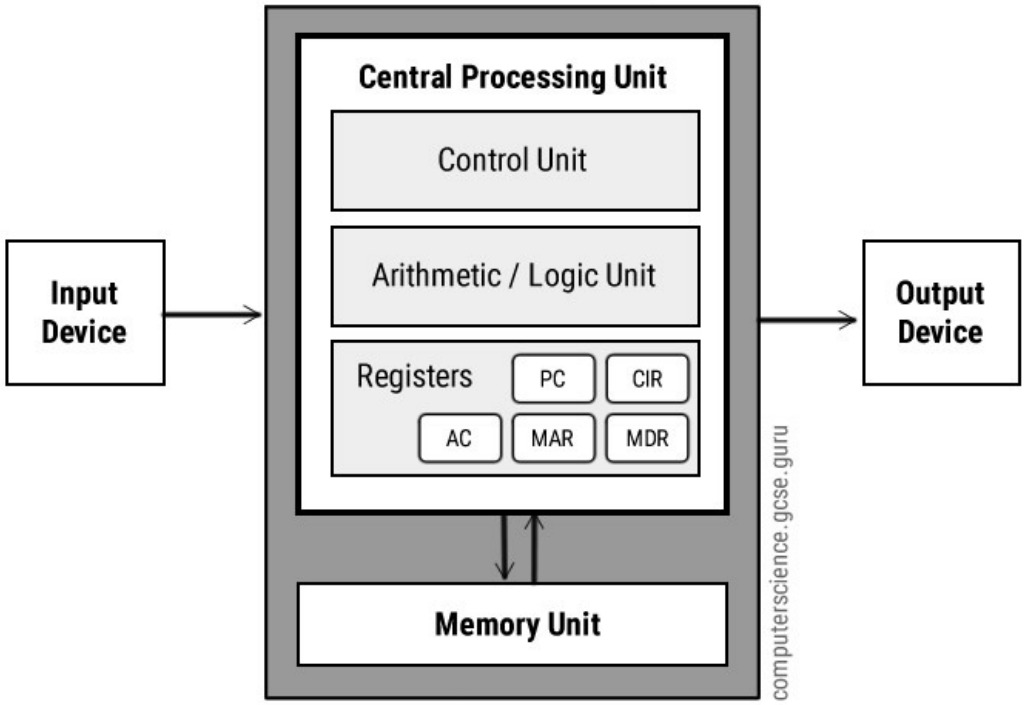
- *Computer* is a machine capable of storing and executing sets of instructions, called *programs*, in order to solve specific problems.
- Computer system is a generic term that may refer to wider scope in various sectors.
- However in this course, we refer a computer system as a system that integrates hardware and software to solve specific problem of our interest.
- The hardware is the physical device that does the actual task by getting all the necessary instruction from the software component.

Introduction to computer system

- The main components of the computer hardware includes:
  - Secondary storage device like hard disk, floppy disk, flash disk, magnetic tape, etc
  - Primary memory which is also called RAM
  - Central processing unit (ALU, CU, and Registers)
  - Output and input devices
  - Communication system
  - etc

The Von Neumann Architecture

- The Von Neumann Architecture is the state of art computer system architecture that has been in use for several decades (since invention of modern computers)
- The von Neumann architecture is a computer design model that uses a processing unit and a single separate storage structure to hold both instructions and data.
- The term "***stored-program computer"*** is generally used to mean a computer of this design



Operating systems

- Use of operating system started since 1950s.
- Older computers doesn’t have operating systems
- OS have two basic services
  - it offers *services* to application programs (such as hiding the technical details of disk access to applications)
  - it *coordinates* system resources if more than one program is started simultaneously.
- Some of the applications are
  - controlling and allocating memory and CPU time
  - Prioritizing system request
  - Controlling input and output devices
  - Facilitating networking and managing file system
  - manage sharing of peripheral devices like printers, scanners, etc

Operating systems

- OS can be multi-user, multi-tasking, multi processing, Multi-threading

- **Multi-user**
  - A multi-user Operating System allows for multiple users to use the same computer at the same time and/or different times.
  - Each user require independent terminals that requiest action from central device
  - Examples of multi-user Operating Systems.
    - Linux
    - Unix
    - Windows 2000

Operating systems

- **Multi-processor**
  - An Operating System capable of supporting and utilizing more than one computer processor.
  - Below are some examples of multiprocessing Operating Systems.
    - Linux
    - Unix
    - Windows 2000

Operating systems

- **Multi-tasking**
  - An Operating system that is capable of allowing multiple software processes to run at the same time.
  - Below are some examples of multitasking Operating Systems.
    - Linux
    - Unix
    - Windows 2000

Operating systems

- **Multithreading**
  - Operating systems that allow different parts of a software program to run concurrently.
  - Operating systems that would fall into this category are:
    - Linux
    - Unix
    - Windows 2000

Programming language software

- Programming language is a software designed to translate the procedures/steps one should follow to solve a given problem using computers

- Programming language is the key for success for development of software that solves various problems of organizations and institutions

- Programming languages can be broadly classified into three as
  - Machine language:
  - Assembly language
  - High-level language

Programming language software

- ***Machine language:***
  - set of instructions represented in the form of zeros and ones which is directly understandable by the machine.

- ***Assembly language***
  - An assembly language is a **low level language** for programming computers.
  - It implements a symbolic representation of the numeric **machine codes** and other constants needed to program a particular **CPU** architecture.
  - They are also called **mnemonic**
  - First developed in the 1950s

Programming language software

- ***High-level language***
  - A high-level programming language is a programming language that, in comparison to low level programming language, may be more abstract, easier to use, or more portable across platforms.
  - Such languages often abstract how CPU operations are performed such as memory access models and management of scope of a variable.

Programming language software

- To execute program written in high level programming language, it has to be in machine understandable form and the execution has to be made in a logical way

- Codes written in machine language can be executed without any problem only on the machine that understands its instruction set

- But if two or more machine language modules should get combined to solve a given problem, it may require a linker for that integration

Programming language software

- Source code of assembly language should be translated into machine executable form using the software called **assembler**.

- The assembler translates the assembly instruction **mnemonic** into **opcodes**

- This translation require resolving ***symbolic names*** for memory location

Programming language software

- In order to execute High level programming language source code, it should also be translated into machine code

- There are two ways of translating the human readable and less-machine dependent source into executable source code.

- These are compiler and interpreter

Programming language software

- A compiler is a program that translates a source program written in some high-level programming language into machine code for some computer architecture (such as the Intel Pentium architecture).

- The compiler can get relevant information about the computer architecture before starting the translation process

- The generated machine code can later be executed many times against different data without requiring recompilation of the source code.

Programming language software

- An ***interpreter*** reads an executable source program written in a high-level programming language as well as data for this program, and it runs the program against the data to produce some results.
- Note that both interpreters and compilers (like any other program) are written in some programming language (usually high level programming languages)

Programming language software

- The interpreter source program is machine independent since it does not generate machine code. Hence interpreter is much better than compiler
- However, an interpreter is generally slower than a compiler because it processes and interprets each statement in a program as many times as the number of the evaluations of this statement.
- Hence it is weaker than compiler

Programming language software

- It is up you to choose your preference on the translator
- But your choice is also depends on the programming language of your preference
- The following are some of the high level programming languages with their mode of translation

PL	Translation	PL	Translation
C	compiler	Perl	Both (interpreter & compiler)
Java	compiler	Python	both
Fortran	compiler	Java script	both
Java	compiler	PHP	both
C++	compiler	Basic	both
C#	compiler	Ada	compiler
Pascal	compile		

Programming language software

- In high level programming language, in order to generate the final executable program code, the source program object code must be integrated with various other object codes called libraries.

- This process is called **linking** libraries together.

- The program that does this is called **linker**

- Linker is important for both interpreted and compiled languages

Programming language software

- Most programming language has a special editor to write the source program.

- The program should be compiled and linked after its source code is generated. If there is an error, it has to be fixed

- In this course we will use the FTN95/2003 compiler

Programming language software

- Programming error can be classified as
  - ***Syntax error***
  - ***Run time error***
  - ***Logical error***

Programming language software

- ***Syntax error***
  - Is an error which is due to invalid grammar usage while writing the source code
  - Such program couldn’t be translated into machine language code (easily detected by the compiler or the interpreter)\
  - Example in the language of mathematic
    - {} > 0

Programming language software

- ***Run time error***
  - Which is an error encountered at the time of executing the program.
  - For example, in the language of mathematic (Assuming Domain of interest is the set of real numbers)
    - SQRT(X-Y) results a run time error if Y > X because it can not be computed
  - This error usually terminates the program abnormally.

Programming language software

- ***Logical error***
  - Which is an error in the computational logic to solve a given problem
  - This is an error from the view of the user but not from view the system
  - For example in the language of mathematics if you want to know the result of adding 4 on to 10 and multiplying by 5 and if you write this as
    - 4 + 10 *5
    - This gives an output but not what you desire
    - The output is 54 but what you were expecting 280

Programming language software

- Programming error can be avoided by **debugging**
- **Debugging** is the most time consuming activity in the programming life cycle
- Debugging syntax error is very simple compared to other types of errors
  - because the translator gives all the necessary information that needs to be fixed
- Debugging run time and logical error is very difficult to detect. Why?

Algorithm Design

- In the process of solving a problem, one of the most important aspect is understanding the problem and design a solution for it.

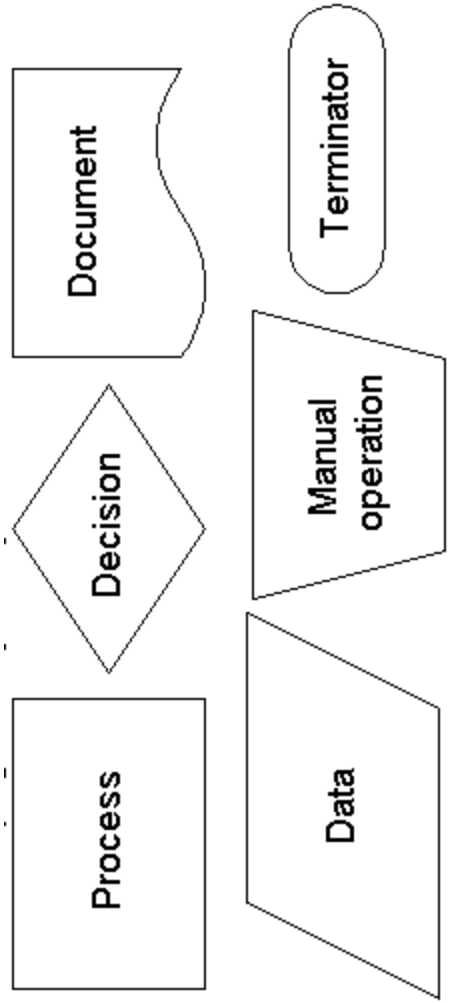
- This can be made in various ways.

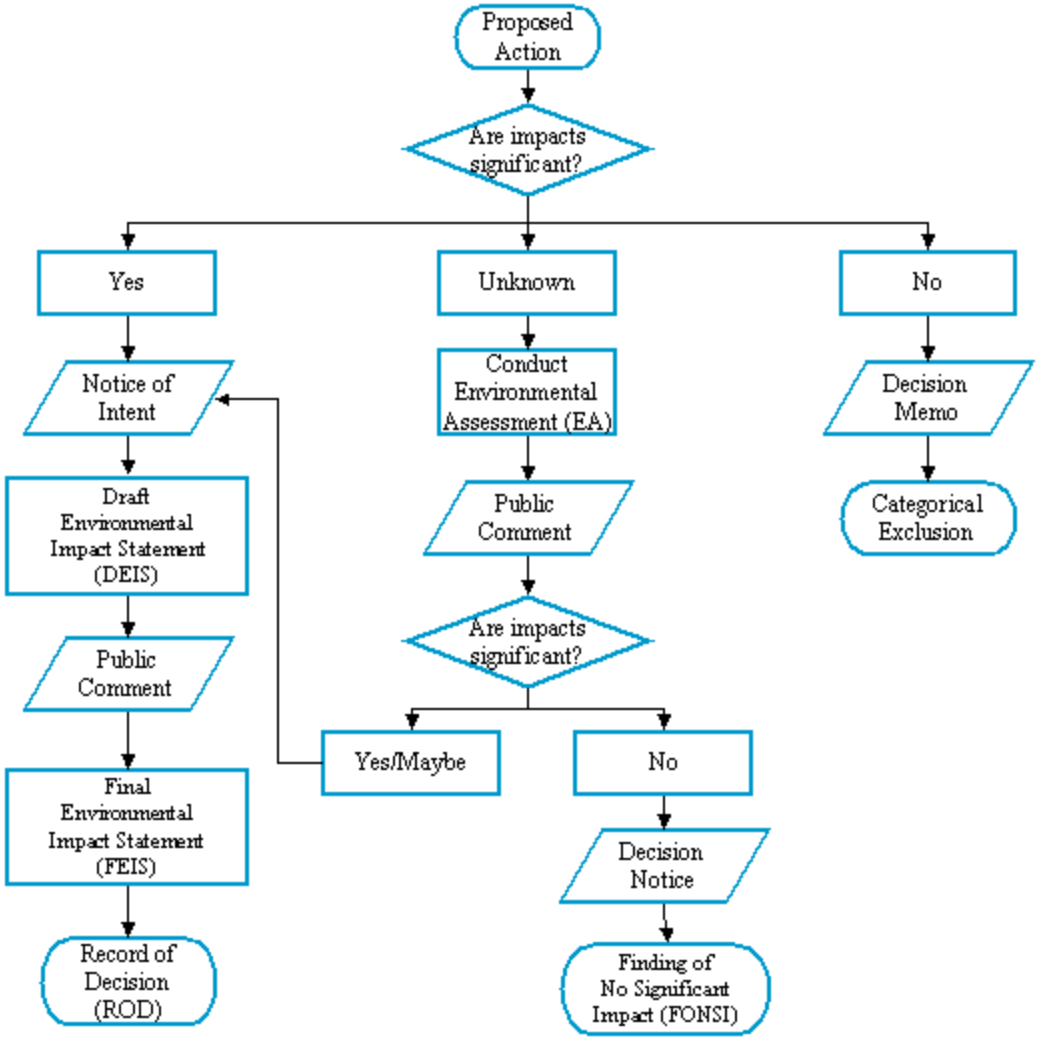
- The simplest one is by using **Flow chart** and **pseudo-code** technique

Examples on
Designing Algorithm Using Flow chart

- Design a flow char to determine the maximum of a given three numbers X, Y, and Z
- Design a flow char to determine the square root of a given number X
- Design a flow char to determine weather a given number is prime or not
- Design a flow char to determine the N^{th} prime number

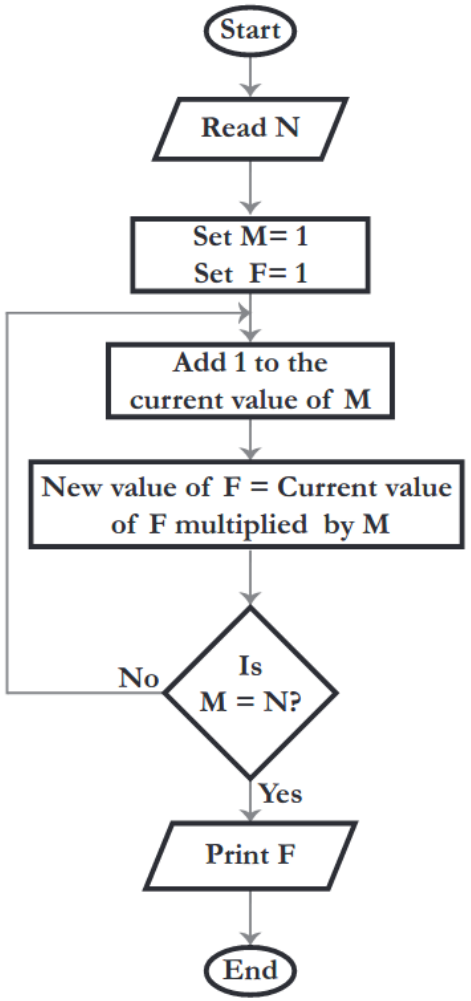
Algorithm Design





Designing Algorithm Using Flow chart

Find factorial of a given number N



Finding factorial of 10

Start

N = 10

M = 1
F = 1

F = 1 * 1 = 1; M < 10; M = 1 +1 = 2
F = 1 * 2 = 2; M < 10; M = 2 +1 = 3
F = 2 * 3 = 6; M<10; M = 3 + 1 = 4
F = 6 * 4 = 24; M<10; M = 4 +1 = 5

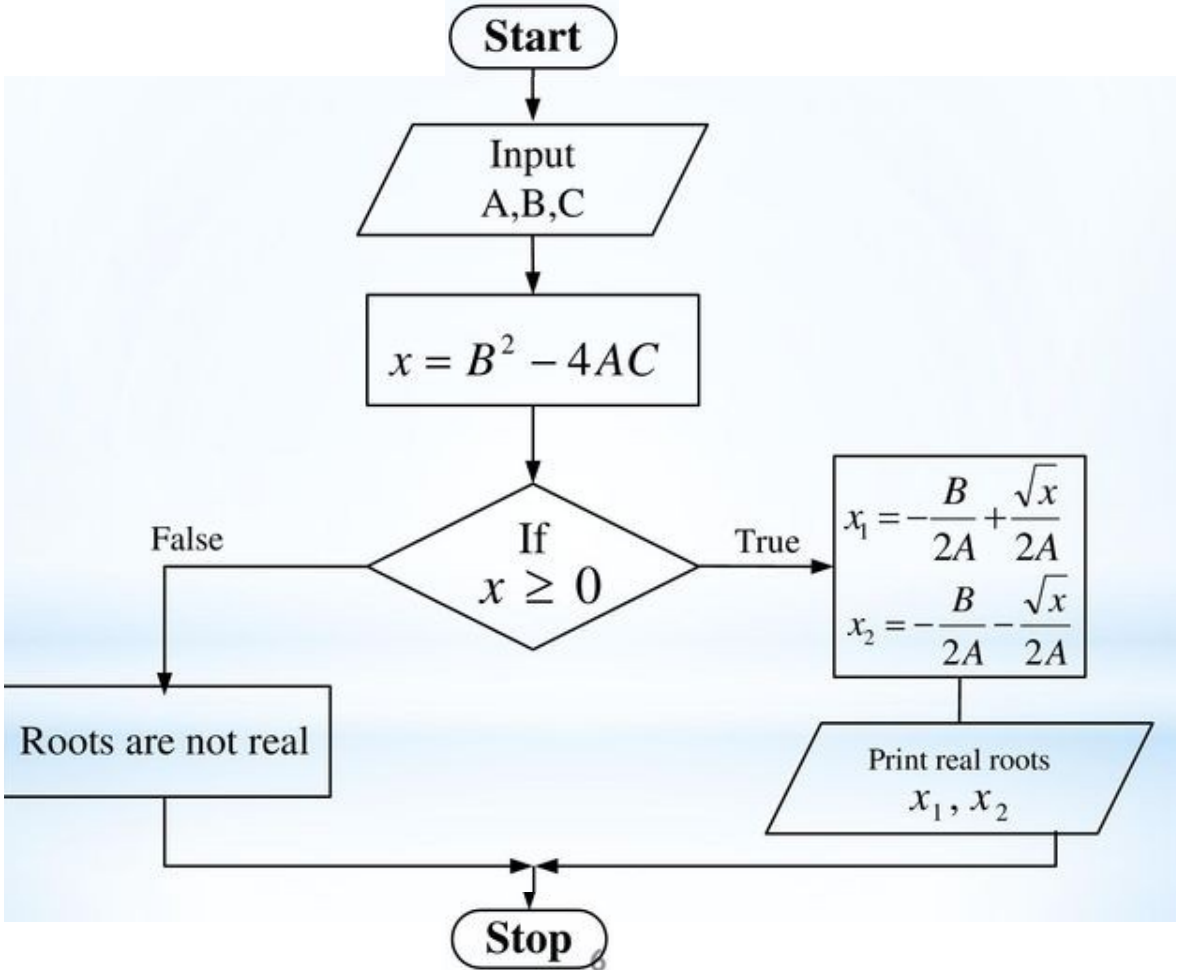
.....
M<10; M = 9 + 1 = 10
F = 362880 * 10 = 3628800; M = 10

Factorial of 10 = 3628800

End

Designing Algorithm Using Flow chart

Finding a real roots of quadratic equation



Assignment Two
Designing Algorithm Using Flow chart

- Develop a pseudo-code and corresponding flowchart for the following problem
  1. A program that compute the inverse of a given NxM matrix
  2. A program that compute molecular weight of a given chemical equation
  3. A program that balance a given molecular equation
  4. A program that compute the total molecular weight of a reactant in a balanced chemical equation

Fortran programming language

History

- Fortran (FORmula TRANslation) was the first high-level programming language.
- It was devised by John Bachus in 1953.
- The first compiler was produced in 1957.
- Fortran is highly standardized, making it extremely portable (able to run under a wide range of computers and operating systems).

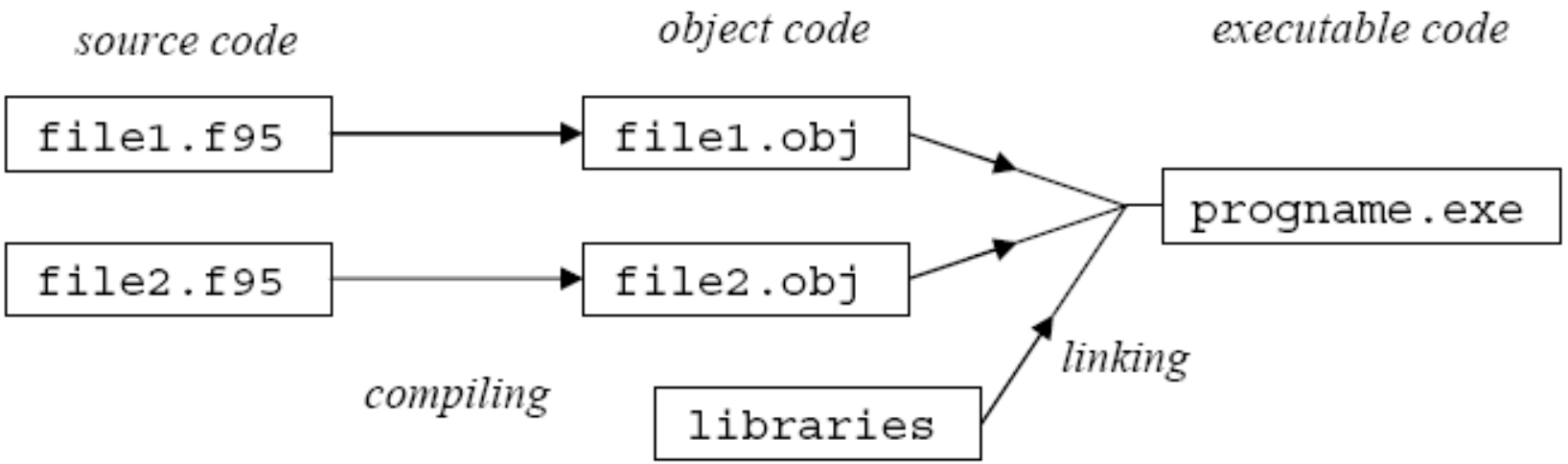
Fortran programming language

- It is an evolving language, passing through a sequence of international standards:
  - ***Fortran I – is released in 1957***
  - ***Fortran II – is released in 1958***
  - ***Fortran 66 (Fortran IV) – the original ANSI standard (accepted 1972)***
  - ***Fortran 77*** – ANSI X3.9-1978
    - introduced structured programming
  - ***Fortran 90*** – ISO/IEC 1539:1991 – major revision, making Fortran into a modern computer language
    - modularity, use of virtual memory
  - ***Fortran 95*** – ISO/IEC 1539-1: 1997
    - improvements for parallel processing
  - ***Fortran 2003*** – ISO/IEC 1539-1:2004(E)
    - adds object-oriented support and interoperability with C

Fortran programming language

- Many Fortran compilers exist
- ***Silverfrost FTN95***
  - http://www.silverfrost.com/
  - This is essentially an upgraded version of the compiler and is available free for personal use only. It can also be downloaded from cnet (http://www.download.com/).
- **Gnu Fortran**
  - http://gcc.gnu.org/wiki/GFortran
- **G95**
  - http://www.g95.org/
- The particular compiler which we shall use is FTN95, originally provided by Salford Software1, which is a Fortran 95 compiler.
- Like many Fortran implementations it comes with additional library routines for producing, for example, graphical output or Windows application

Process of Creating executable code



What's good about FORTRAN?

- FORTRAN is very natural language for expressing numerical computations

- FORTRAN handles arrays really very well (extremely important for numerical computing)
  - Think of arrays as enumerated lists of data until we learn about them in a few weeks or as a vector, matrix or complex structure)
- It has natural syntax for handling subsets or parts of a an array of data

What's good about FORTRAN?

- Produces extremely fast code.
  - The language has a very restricted set of operations that allows the compiler to find, in many cases, an optimal order in which to execute those operations
  - C++ code can sometimes be made to run as fast as FORTRAN but it can take some work on the part of the programmer
  - Usually the language recommended by most vendors of supercomputing systems
- Highly standardized language
  - Makes porting code to a new computer easy

What's bad about FORTRAN?

- Not flexible for input/output of data compared to C++
  - Fixed in FORTRAN 2003
- Not good for system level programming
  - Can be done in some cases but it is not easy
  - Don't try to write an editor in FORTRAN!
- Expressing modern programming concepts can be difficult in earlier versions of FORTRAN
  - FORTRAN 90 lessens this problem
  - FORTRAN 2003 fully resolves this problem