CE 2207
Numerical Analysis and Computer Programming
Course outline:

Computer Programming:
- Introduction to Computer Programming
- Study of C language
- Computer Applications in Civil Engineering Problems

References:
- Programming in ANSI C
  ..........E. Balagurusamy
- The C Programming Language 2nd Edition
  ..........Brian W. Kernighan, Dennis M. Ritchie
LECTURE-01
Introduction to Computer Programming
What is Computer programming?

- **Computer programming** is a process that leads from an original formulation of a computing problem to executable computer programs.
- The purpose of programming is to find a sequence of instructions that will automate performing a specific task or solving a given problem.
- Important tasks in computer programming include testing, debugging, and maintaining the source code, implementation of the build system, and management of derived artifacts such as machine code of computer programs.
• Debugging

Debugging is a very important task in the software development process since having defects in a program can have significant consequences for its users. Some languages are more prone to some kinds of faults because their specification does not require compilers to perform as much checking as other languages.
• **Algorithm**
An algorithm is a well-defined procedure that allows a computer to solve a problem. Another way to describe an algorithm is a sequence of unambiguous instructions.

• **Flowchart**
A flowchart is a type of diagram that represents an algorithm, workflow or process, showing the steps as boxes of various kinds, and their order by connecting them with arrows. This diagrammatic representation illustrates a solution model to a given problem.
Languages are a means of communication. Normally people interact with each other through a language. On the same pattern, communication with computers is carried out through a language. This language is understood both by the user and the machine. Just as every language like English, has its own grammatical rules; every computer language is also bounded by rules known as syntax of that language. The user is bound by that syntax while communicating with the computer system.
Languages form an approximate spectrum from "low-level" to "high-level"; "low-level" languages are typically more machine-oriented and faster to execute, whereas "high-level" languages are more abstract and easier to use but execute less quickly. It is usually easier to code in "high-level" languages than in "low-level" ones.
Computer languages are broadly classified as follows:

Low Level Language: The term low level highlights the fact that it is closer to a language which the machine understands.

The low level languages are classified as:

Machine Language: This is the language (in the form of 0’s and 1’s, called binary numbers) understood directly by the computer. It is machine dependent. It is difficult to learn and even more difficult to write programs.
**Assembly Language:**
This is the language where the machine codes comprising of 0’s and 1’s are substituted by symbolic codes to improve their understanding. It is the first step to improve programming structure. It is simpler and less time consuming than machine level programming, it is easier to locate and correct errors in assembly language than in machine language programs. It is also machine dependent. Programmers must have knowledge of the machine on which the program will run.
**High Level Language:**

A high-level programming language is a programming language with strong abstraction from the details of the computer. In comparison to low-level programming languages, it may use natural language elements, be easier to use, or may automate (or even hide entirely) significant areas of computing systems (e.g. memory management), making the process of developing a program simpler and more understandable relative to a lower-level language.
Examples of High level languages:

**BASIC (Beginners All Purpose Symbolic Instruction Code):** It is widely used, easy to learn general purpose language. Mainly used in microcomputers in earlier days.

**COBOL (Common Business Oriented language):** A standardized language used for commercial applications.

**FORTRAN (Formula Translation):** Developed for solving mathematical and scientific problems. One of the most popular languages among scientific community.

**C:** Structured Programming Language used for all purpose such as scientific application, commercial application, developing games etc.

**C++:** Popular object oriented programming language, used for general purpose.
To make the machine understand the instructions provided by both high level and low level languages, programming language instructors are used. They transform the instruction prepared by programmers into a form which can be interpreted & executed by the computer.
Following are the various tools to achieve this purpose:

**Compiler:**
The software that reads a program written in high level language and translates it into an equivalent program in machine language is called as compiler. The program written by the programmer in high level language is called source program or source code and the program generated by the compiler after translation is called as object program.
**Interpreter:**

It also executes instructions written in a high level language. Both compiler & interpreter have the same goal i.e. to convert high level language into binary instructions, but their method of execution is different. The compiler converts the entire source code into machine level program, while the interpreter takes 1 statement, translates it, executes it & then again takes the next statement.
**Assembler:**
The software that reads a program written in assembly language and translates it into an equivalent program in machine language is called as assembler.

**Linker:**
A linker or link editor is a computer program that takes one or more object files generated by a compiler and combines them into a single executable file, library file, or another object file.
Integrated Development Environments (IDE)

The process of editing, compiling, running, and debugging programs is often managed by a single integrated application known as an Integrated Development Environment, or IDE. Under Windows, Microsoft Visual Studio is a good example of a popular IDE.
**History of C:**

- The C programming language is a structure oriented programming language, developed at Bell Laboratories in 1972 by Dennis Ritchie.
- C programming language features were derived from an earlier language called “B” (Basic Combined Programming Language – BCPL)
- C language was invented for implementing UNIX operating system.
History of C (Cont.):

• In 1978, Dennis Ritchie and Brian Kernighan published the first edition “The C Programming Language” and is commonly known as K&R C.

• In 1983, the American National Standards Institute (ANSI) established a committee to provide a modern, comprehensive definition of C. The resulting definition, the ANSI standard, or “ANSI C”, was completed late 1988.

• Many of C’s ideas & principles were derived from the earlier language B, thereby naming this new language “C”.
Taxonomy of C Language

1960 • Algol
1967 • BCPL
1970 • B
1972 • Traditional C
1978 • K & RC
1989 • ANSI C
1990 • ANSI / ISO C
1999 • C99
Basic Components of C:

i. Preprocessor Commands
ii. Functions
iii. Variables
iv. Statements & Expressions
v. Comments

C has now become a widely used professional language for various reasons:

• Easy to learn
• Structured language
• It produces efficient programs
• It can handle low-level activities
• It can be compiled on a variety of computer platforms
Why is C Popular

Ø It is reliable, simple and easy to use.
Ø C is a small, block-structured programming language.
Ø C is a portable language, which means that C programs written on one system can be run on other systems with little or no modification.
Ø C has one of the largest assortments of operators, such as those used for calculations and data comparisons.
Ø Although the programmer has more freedom with data storage, the languages do not check data type accuracy for the programmer.
Why should we use C?

C was initially used for system development work. Some examples of the use of C might be:

- Operating Systems
- Language Compilers
- Assemblers
- Text Editors
- Print Spoolers
- Network Drivers
- Modern Programs
- Databases
- Language Interpreters
Characteristics of C Language:
Ø Middle level language.
Ø Small size – has only 32 keywords
Ø Extensive use of function calls enables the end user to add their own functions to the C library.
Ø Supports loose typing – a character can be treated as an integer & vice versa.
Ø Structured language
Ø It can be compiled on a variety of computers.
Characteristics of C Language:
(Cont.):
Ø Low level (Bit Wise) programming readily available
Ø Pointer implementation - extensive use of pointers for memory, array, structures and functions.
Ø It has high-level constructs.
Ø It can handle low-level activities.
Ø It produces efficient programs.
Ø It can be compiled on a variety of computers.
Basic Structure of a C Program

Documentation section
Link Section
Definition Section
Global Declaration Section
main () Function Section
{
    Declaration part
    Executable part
}
Subprogram section
    Function 1
    ................
    ................
    ................
    Function n
**Documentation section**

- It consists of comment lines giving the name of the program, the author and other details.
- It starts with the character /* and ends with */.
- Comment lines are not an executable statement.

**Link section**

- The link section provides instruction to the compiler to link functions from the system library.
- Library functions are grouped category-wise and stored in different files known as header files.
- The preprocessor directive # include as follow:

  ```
  # include<file name>
  ```
**Definition Section**

- It defines all the symbolic constants.
- Symbolic constants are defined with `#define` instruction.
- `#define` is a pre-processor compiler directive not a statement. Therefore `#define` lines should not end with semicolon (;)
- Symbolic constants are mainly defined with uppercase letters
- `#define` instructions are mainly placed before the main() function
- For example `#define PI 3.1416`

**Global Declaration Section**

- The variables that are used in more than one function is called global variables
- This section declares all the user defined functions
- For example, definition of integer and real numbers are declared in this section.
**main() function**

- It is a part of every C program
- The parenthesis after `main` indicates that `main` is a program building block called a *function*
- C programs are composed of one or many functions like it but `main` is a must
- This sections contains two parts, declaration and executable part. The declaration parts declares all types of variables are defined in executable.
- All the statements in declaration and executable parts are ends with semicolon (;)

**Notes:**
- All commands in C must be lowercase (small letter).
- C has a free-form line structure. End of each statement must be marked with a semicolon (;). Multiple statements can be on the same line. *White space is* ignored. The C program starting point is identified by the word **main()**.
FILES USED IN A C PROGRAM

Source File- This file contains the source code of the program. The file extension of any c file is c. The file contains C source code that defines the main function & maybe other functions.

Header Files Header files contain definitions of functions and variables which can be incorporated into any C program by using the pre-processing #include statement.

This is done at the beginning of the C source file. For example, to use the function `printf()` in a program, the line `#include <stdio.h>` should be at the beginning of the source file.
All header files have the extension .h and generally reside in the /user/include subdirectory.

```c
#include <string.h>
#include <math.h>
#include <mylib.h>
```

The use of angle brackets <> informs the compiler to search the compiler’s include directories for the specified file. The use of the double quotes " " around the filename informs the compiler to start the search in the current directory for the specified file.
- **Object File** - An object file is a file containing object code, with an extension '.o', meaning re-locatable format machine code that is usually not directly executable. Object files are produced by an assembler, compiler, or other language translator, and used as input to the linker, which in turn typically generates an executable or library by combining parts of object files.

- **Executable File** - The binary executable file is generated by the linker. The linker links the various object files to produce a binary file that can be directly executed.
Compilation & Execution of a C Program

source code

Preprocessor

expanded code

Compiler

assembly code

Assembler

other object file

Libraries

object code

Linker

executable code
/*PROGRAM:02*/
/*DEVELOP A PROGRAM TO WRITE KUET*/
#include<stdio.h>
#include<conio.h>
main()
{
printf("OUTPUT:\n");
printf("KUET");
printf("\n Roll:\n Section: A\nGroup:");
getch();
}

OUTPUT:
KUET
Roll:
Section: A
Group:
THANKS TO
EVERYONE
See you in next class