# CS11001/CS11002 Programming and Data Structures (PDS)

(Theory: 3-1-0)

## Syllabus

1. Familiarization of a computer and the environment and execution of sample programs
2. Expression evaluation
3. Conditionals and branching
4. Iteration
5. Functions
6. Recursion
7. Arrays
8. Structures
9. Linked lists
10. Data structures

## Time Schedule

- Sections 11,12
- Class Room: F116
- Time Schedule:
  - Wednesday 11:30-12:25
  - Thursday 10:30-11:25
  - Friday 08:30-09:25
- Class Teacher: Debdeep Mukhopadhyay
- Teaching Assistant (TA):
  - Aniket Nayak
  - Soma Saha

## References

1. Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, Prentice Hall of India.
2. E. Balaguruswamy, *Programming in ANSI C*, Tata McGraw-Hill.
3. **Byron Gottfried, *Schaum's Outline of Programming with C*, McGraw-Hill.**
4. **Seymour Lipschutz, *Data Structures*, Schaum's Outlines Series, Tata McGraw-Hill.**
5. Ellis Horowitz, Satraj Sahni and Susan Anderson-Freed, *Fundamentals of Data Structures in C*, W. H. Freeman and Company.
6. R. G. Dromey, *How to Solve it by Computer*, Prentice-Hall of India.

## A Reference url

- http://cse.iitkgp.ac.in/pds/semester/notes/

## Marks distribution

- Theory :

|  |  |
|---|---|
| Class test 1 | : 10 |
| Mid Semester Exam | : 30 |
| Class test 2 | : 10 |
| End Semester Exam | : 50 |

## Software

In Labs we use:
- Linux work stations
- GNU C Compiler
- EMACS editor

Download GCC and EMACS for windows:
http://cse.iitkgp.ac.in/pds/software/

## Lets Start!

## Digital Computer

- A **computer** is a machine that can perform computation. It is difficult to give a precise definition of computation. Intuitively, a computation involves the following three components:
  - **Input:** The user gives a set of input data.
  - **Processing:** The input data is processed by a well-defined and finite sequence of steps.
  - **Output:** Some data available from the processing step are output to the user.

## Types of Problems

- **Functional Problems:** A set of arguments a1,a2,...,an constitute the input. Some function f(a1,a2,...,an) of the arguments is calculated and output to the user.
- **Decisional Problems:** These form a special class of functional problems whose outputs are "yes" and "no" (or "true" and "false", or "1" and "0", etc).

## Types of Problems

- **Search Problems:** Given an input object, one tries to locate some particular configuration pertaining to the object and outputs the located configuration, or "failure" if no configuration can be located.
- **Optimization Problems:** Given an object, a configuration and a criterion for goodness, one finds and reports the configuration pertaining to the object, that is best with respect to the goodness criterion. If no such configuration is found, "failure" is to be reported.

## Examples 1: Polynomial root finding

- **Category:** Functional problem
- **Input:** A polynomial with real coefficients
  **Output:** One (or all) real roots of the input polynomial
  **Processing:** Usually, one involves a numerical method (like the Newton-Raphson method) for computing the real roots of a polynomial.

## Example 2: **Matrix inversion**

- **Category:** Functional problem
- **Input:** A square matrix with rational entries
  **Output:** The inverse of the input matrix if it is invertible, or "failure"
  **Processing:** Gaussian elimination is a widely used method for matrix inversion. Other techniques may also be conceived of.

## Example 3: **Primality testing**

- **Category:** Decision problem
- **Input:** A positive integer
  **Output:** The decision whether the input integer is prime or not
  **Processing:** For checking the primality of n, it is an obvious strategy to divide n by integers between 2 and square root of n. If a divisor of n is found, n is declared "composite" ("no"), else n is declared "prime" ("yes").
  - *This obvious strategy is, however, very slow. More practical primality testing algorithms are available. The first known (theoretically) fast algorithm is due to three Indians (Agarwal, Kayal and Saxena) from IIT Kanpur*.

## Problem 5: **Traveling salesman problem (TSP)**

- **Category:** Optimization problem
- **Input:** A set of cities, the cost of traveling between each pair of cities, and the criterion of cost minimization
  **Output:** A route through all the cities with each city visited only once and with the total cost of travel as small as possible
- **Processing:** Since the total number of feasible routes for n cities is n!, a finite quantity, checking all routes to find the minimum is definitely a strategy to solve the TSP. However, n! grows very rapidly with n, and this brute-force search is impractical. We do not know efficient solutions for the TSP. One may, however, plan to remain happy with a suboptimal solution in which the total cost is not the smallest possible, but close to it.

## Problem 6: Weather prediction

- **Category:** Functional problem
- **Input:** Records of weather for previous days and years. Possibly also data from satellites.
  **Output:** Expected weather of Kharagpur for tomorrow
  **Processing:** One statistically processes and analyzes the available data and makes an educated extrapolating guess for tomorrow's weather.

## Problem 7: Web browsing

- **Category:** Functional problem
- **Input:** A URL (abbreviation for "Uniform Resource Locator" which is colloquially termed as "Internet site")
  **Output:** Display (audio and visual) of the file at the given URL
  **Processing:** Depending on the type of the file at the URL, one or more specific programs are run and the desired output is generated. For example, a web browser can render an HTML page, images in some formats etc. For displaying a movie, a separate software (or its plug-in) need be employed.

## Problem 8: Chess : Can I win?

- **Category:** Search problem
- **Input:** A configuration of the standard 8x8 chess board and the player ("white" or "black") who is going to move next
  **Output:** A winning move for the next player, if existent, or "failure"
  **Processing:** In general, finding a winning chess move from a given state is a very difficult problem. The trouble is that one may have to explore an infinite number of possibilities. Even when the total possibilities are finite in number, that number is so big that one cannot expect to complete exploration of all of these possibilities in a reasonable time. A more practical strategy is to investigate all possible board sequences involving a small number of moves starting from the given configuration and to identify the best sequence under some criterion and finally prescribe the first move in the best sequence.

## Computers and Programs

- A computer is a device that can solve these and similar problems. A *digital* computer accepts, processes and outputs data in digitized forms (as opposed to analog forms).
- Programming is a process of writing instructions in a language that can be understood by the computer so that a desired task can be performed.

## A Computer understands only 0s and 1s

- A **high-level language** helps you make your communication with computers more abstract and simpler and also widely machine-independent.
- You then require computer programs that convert your high-level description to the assembly-level descriptions of individual machines, one program for each kind of CPU. Such a translator is called a **compiler**.

## Three Steps

**Step 1: Write the program in a high-level language**

- You should use a text editor to key in your program. In the laboratory we instruct you to use the **emacs** editor. You should also *save* your program in a (named) file.

## Step 2: **Compile your program**

- You need a compiler to do that. In the lab you should use the C compiler **cc** (a.k.a. **gcc**).

  ```
  cc myprog.c
  ```

  If your program compiles successfully, a file named a.out (an abbreviation of "assembler output") is created. This file stores the machine instructions that can be understood by the particular computer where you invoked the compiler.

- If compilation fails, you should check your source code.
  - The reason of the failure is that you made one or more mistakes in specifying your idea.
  - Compilers are very stubborn about the syntax of your code.
  - Even a single mistake may let the compiler churn out many angry messages.

## Step 3: **Run the machine executable file**

- by typing
- ./a.out (and then hitting the return/enter button) at the command prompt.

## First Program

```
#include <stdio.h>
 main (){
      printf("Hello, world!\n");
 }
```

## Second Program

```
#include <stdio.h>
   main (){
       int n;
        scanf("%d",&n);
        printf("%d\n",n);
     }
```