Lecture notes on pcds theory and problem solving and programming essay sample

Linguistics, Language

I would like to thank all who gave this opportunity to prepare and teach. First of all i would like to wish u all a very good luck for your B. Tech and whole life. As i said in class lectures to be at least be eloquent in one of programming languages, as it decides your fate, if u wish to go for software industry. we can say your B. Tech program starts with C language... PCDS Theory 1st unit to start up basics: We want to learn programming language because " programming is the part where rubber meets the road".. . it means it"s the main path where actual implementation of program starts. Requirements analysis-> design-> Implementation-> Testing-> Maintenance...these are simply Software Development Life Cycle Phases (SDLC)... Programming comes into existence after the design phase...i. e., Implementation My dear students this is the 1st unit of your academic syllabus of programming in c and data structures . 1st unit deals with the basic concepts instructions, software, programs, pseudo code, heuristics of algorithms, problem solving aspects, strategies and implementation of algorithms Note: most of things specified in brackets (.....) are just comments for understand ability and readability.

Please verify previous document that specifies syllabus copy and academic calendar. Analysis, Complexity and Efficiency of algorithms and few concepts of testing like verification... (testing is generally performed at every phase of life cycle at its end to check proper requirements and intension of finding errors in phase... programmer generally thinks how to make a program and tester thinks how to break a program) these concepts covers under further units of your syllabus as they are advanced concepts such as sorting looping related to final output...as your performance is calculated after the time

period of year or semester similarly analysis, efficiency are used to calculate performance of algorithm at the end of implementations. Now carry on with your 1st unit...apart from this notes please prepare precise notes of your own...sometimes u may feel text book is better than this notes as text wise unit is small but complex, but i elongated to the way u can understand and few info collected from other resources . CP Lab initial conditions to start: C program is generally stored with extension of *. c; c program is done in many platforms like windows and unix.

In windows u can have various compilers like turbo c (tc, tcc), dave cpp, and in unix gcc and many command line prompts are present. For future references to review or change ur past works u must save the programs by creating folder in any of drives. Ex: D: vicky sample. c (D: – sample drive that specifies drive in which program is to be saved, Vicky- sample folder where program to be saved and finally sample (filename) . c(extension to be saved)) (- back slash;/- forward slash; saving execution path- ; comments : // (single line), and /*.....*/ (multiline); – white space characters ,...; please specify the brackets circular, rectangular and flower brackets properly) SHRADDAVAN LABHATE GNANAM.. GNANAVAN LABATHE SHOWRYAM.. SHOURYAVAN LABATHE SARVAM..... YASHASWI BHAVA 🕹 -With Regards G. N. Vivekananda, M. Tech, Assistant Professor, Email Id:

UNIT I: INTRODUCTION TO COMPUTER PROBLEM SOLVING 1. 1 Introduction Computer problem solving(CPS) is an complex process that requires much thought, careful Planning, logical Precision, Persistence and attention to detail thus it‟s a challenging, exciting and satisfying experience with

considerable criteria for personal creativity and expression thus its demanding . If this CPS followed correctly chances of success are greatly amplified. Programs and Algorithms: Set of explicit and unambiguous (clear) instructions expressed in a certain programming language that indeed acts as vehicle for computer solution is called a program. Algorithm is the sequence of step by step instructions to solve the problem corresponding to a solution that is independent of any programming language (or) Algorithm consists of a set of explicit and unambiguous finite steps which, when carried out for a given set of initial conditions, produces output and terminate in a finite time.

To attain a solution to the problem, we have to supply with considerable input Program then takes he input and manipulates it according to instructions and produces output, that represents a solution to problem. Features of an efficient algorithm: Free of ambiguity Efficient in execution time Concise and compact Completeness Definiteness Finiteness Requirements for solving problems by computer: There are many algorithms to solve many problems. Let‟s consider to look up someone‟s telephone no. in telephone directory, we need to employ or design an algorithm as a first step. Tasks like this are generally performed without any thought to the complex underlying mechanism needed to effectively conduct the search, it surprises sometimes when we develop the computer algorithms that solution must be specified in logical precision. • • Requires interaction between “ programmer” and user Specifications include: – Input data • – – Type, accuracy, units, range, format, location, sequence

Special symbols to signal end of data Output data (results)

type, accuracy, units, range, format, location, “ headings”

We have to know how is output related to input Any special constraint

Example: “ find the phone no. of a person” Problems get revised often

These examples serve to emphasize the importance of influence of data organization on the performance of algorithms 1. 2 The Problem Solving Aspect Problem solving is a creative process, which needs systemization and mechanization that has no universal method, as there are no different strategies appear to work for different people. Problem Solving is the sequential process of analysing information related to a given situation and generating appropriate response options. There are 6 steps that you should follow in order to solve a problem: 1. Understand the Problem 2. Formulate a Model 3. Develop an Algorithm 4. Write the Program 5. Test the Program 6. Evaluate the Solution The main thing to be known here is “ what must be done, rather than how to do it” Problem Definition Phase: The first step in solving a problem is to understand problem clearly. Hence, the first phase is the problem definition phase.

That is, to extract the task from the problem statement. If the problem is not understood, then the solution will not be correct and it may result in wastage of time and effort. When we get to start a problem? There are many ways to solve the problem and also many solutions. The sooner you start coding your program the longer it is going to take. Consider a simple example of how the

input/process/output works on a simple problem: Example: Rule of thumb is generally used props or heuristics(A common sense rule (or set of rules) intended to increase the probability of solving some problem), to get a start with the problem. This general approach focusing on a particular problem can often give us foothold we need for making start on solution. Calculate the average grade for all students in a class. 1. Input: get all the grades … perhaps by typing them in via the keyboard or by reading them from a USB flash drive or hard disk. 2. Process: add them all up and compute the average grade. 3. Output: output the answer to either the monitor, to the printer, to the USB flash drive or hard disk … or a combination of any of these devices.

As you can see, the problem is easily solved by simply getting the input, computing something and producing the output. However, nothing less than a complete proof of correctness of our algorithm is entirely satisfactory. Similarities of Problems: This method is used to find out if a problem of this sort has been already solved and to adopt a similar method in solving the problem. The contribution of experience in the previous problem with help and enhance the method of problem for the current problem. Working backward from the solution: When we have a solution to the problem then we have to work backward to find the starting condition. Even a guess can take us to the starting of the problem. This is very important to systematize the investigation to avoid duplication of our effort.

The strategy of working backwards entails starting with the end results and reversing the steps you need to get those results, in order to figure out the

answer to the problem. There are at least two different types of problems which can best be solved by this strategy: (1) When the goal is singular and there are a variety of alternative routes to take. In this situation, the strategy of working backwards allows us to ascertain which of the alternative routes was optimal. An example of this is when you are trying to figure out the best route to take to get from your house to a store. You would first look at what neighbourhood the store is in and trace the optimal route backwards on a map to your home. (2) When end results are given or known in the problem and you’re asked for the initial conditions. An example of this is when we are trying to figure out how much money we started with at the beginning of the day, if we know how much money we have at the end of the day and all of the transactions we made during the day.

Fig: Working of Phases

Fig: The Interactions between Computer Problem-Solving Phases

General problem Solving Strategies: The following are general approaches. • Working backwards Reverse-engineering Once you know it can be done, it is much easier to do What are some examples? • Look for a related problem that has been solved before Java design patterns Sort a particular list such as: David, Alice, Carol and Bob to find a general sorting algorithm • Stepwise Refinement Break the problem into several sub-problems Solve each sub problem separately Produces a modular structure • K. I. S. S. = Keep It Simple Stupid! Apart from above, there are a no. of general and powerful computational strategies that are variously used

in various guises in computing science. The most widely strategies are listed below: a)Divide and conquer b)Binary doubling strategy c)Dynamic programming a)Divide and conquer method: The basic idea is to divide the problem into several sub problems beyond which cannot be further subdivided. Then solve the sub problems efficiently and join then together to get the solution for the main problem. When we consider the binary search algorithm , applying this strategy to an ordered data set results in an algorithm that needs to make only log2n rather than n comparisons to locate an item in n elements.

When this is used in sorting algorithms, no. of comparisons can be reduced from the order of n2 steps to nlog2n steps. b)Binary doubling strategy: The reverse of binary doubling strategy, i. e. combining small problems in to one is known as binary doubling strategy. This strategy is used to avoid the generation of intermediate results. With this doubling strategy we express the next term n to be computed in terms of the current term which is usually a function of n/2 in order to avoid the need to generate intermediate terms, c)Dynamic programming used: Dynamic programming is used when the problem is to be solved in a sequence of intermediate steps. It is particularly relevant for many optimizations problems, i. e. frequently encountered in Operations research. This method relies on idea that a good solution to a large problem can sometimes be built up from good or optimal solutions to smaller problems. Many approaches like greedy search, back tracking and bound and – bound evaluations follow this approach. There are still more

strategies but usually associated with more advanced algorithms, we will not further proceed further by using them.

1. 3 Top Down Design After defining a problem to be solved we vague for an idea of how to solve it. People as problem solvers can only focus on time, logic or instructions. Top-down design or step wise refinement is a strategy that can be applied to find a solution to a problem from a vague outline to precisely define the algorithm and program implementation by stepwise refinement. Its used to build solutions to problem in stepwise fashion. Breaking a problem into sub problems involves following steps: In top-down model, an overview of the system is formulated, without going into detail for any part of it. Each part of the system is then refined in more details. Each new part may then be refined again, defining it in yet more details until the entire specification is detailed enough to validate the model. This design model can also be applied while developing algorithm. Refinement is applied until we reach to the stage where the subtasks can be directly carried out. For the ease of use and understandability it successively refines by Breaking the problem/tasks into a set of sub tasks/sub problems called modules (divide and conquer)

Fig: An view of top-down design

Fig: Subdividing the party planning

This process continues for as many levels as it takes to expand every task to the smallest details A step that needs to be expanded is an abstract step. A step that is specified to the finest detail is a concrete step

Top-down design in detail consists follows: 🚌 Definition 🚌 Breaking a problem in to sub problems 🚌 Choice of a suitable data structure 🚌 Constructions of loops 🚌 Establishing initial conditions for loops 🚌 Finding the iterative construct 🚌 Terminations of loops Choice of a suitable data structure: One of the most important decisions we have to make in formulating computer solutions to problems is the choice of appropriate data structures. In appropriate choice of data structure often leads to clumsy, inefficient and difficult implementations. Appropriate choice leads to simple, transparent and efficient implementation. The key to effectively solving many problems comes down to making appropriate choices about associated data structures. As Data structures and algorithms are usually linked to one another, thus not easy to formulate general rules that says choice of data structure is appropriate. Unfortunately with regards to data structures each problem must be considered on its merits.

The sort of things we must however be aware of in setting up data structures are such questions as: 1) How can intermediate results be arranged to allow fast access to information that will reduce the amount of computation required at a later stage? 2) Can the data structure be easily searched? 3) Can the data structure be easily updated? 4) Does the data structure provide a way of recovering an earlier state in the computation? 5) Does the data structure involve the excessive use of storage? 6) Is it possible to impose some data structure on a problem that is not initially apparent? 7) Can the problem be formulated in terms of one of the common data structures (e. g. array, set, queue, stack, tree, graph, list)? These are general considerations

but they give the flavour of sort of things that we can proceed with development of an algorithm. Construction of loops: We are led to a series of constructs, loops, structures that are conditionally executed from the general statements. These structures with input, output statements, computable expressions, and assignments, make up heart of program implementations.

To construct a loop 3 conditions must be considered: 1. Initial conditions that must be applied before the loop begins to execute 2. Invariant relation that must apply after each iteration of the loop 3. Conditions under which the iterative process must terminate 1. To establish initial conditions for loop, effective strategy is to set the loop variables to values that they would have to assume in order to solve the smallest problem associated with loop. Number of iterations n that must be made by a loop are in the range 01) . thus these are used to solve problem for n>= 1. i:= 0; s:= 0—initialization conditions for loop and solution to summing problems when n= 0 while i= 1 s:= s+a[i] end the above steps gives solution to summation problem for n> 0 The other consideration for constructing loops is considered with setting up of termination conditions. Termination of loops: Generally it occurs in many ways . in general they are dictated by nature of problem. Simplest termination occurs when known in advance how many iterations needed to be made. In this we can use termination facilities provided by certain programming language.