CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test - I

Sub:	Programming Languages							Code:	10CS666
Date:	30/ 03 / 2017	Duration:	90 mins	Max Marks:	50	Sem:	VI	Branch:	CSE

Answer Any FIVE FULL Questions

		Marks	OBE	
			CO	RBT
1	What is data-type ? What are the advantages of data-type? Give some examples of data-type.	[10]	CO1	L1
ans	Data-type is a tag or string that maps to some abstract data-structure and corresponding functionality. Advocates of static typing argue that the advantages of static typing include earlier detection of programming mistakes (e.g. preventing adding an integer to a boolean), better documentation in the form of type signatures (e.g. incorporating number and types of arguments when resolving names), more opportunities for compiler optimizations (e.g. replacing virtual calls by direct calls when the exact type of the receiver is known statically), increased runtime efficiency (e.g. not all values need to carry a dynamic type), and a better design time developer experience (e.g. knowing the type of the receiver, the IDE can present a drop-down menu of all applicable members). Static typing fanatics try to make us believe that “well-typed programs cannot go wrong”. While this certainly sounds impressive, it is a rather vacuous statement. Static type checking is a compile-time abstraction of the runtime behavior of your program, and hence it is necessarily only partially sound and incomplete. This means that programs can still go wrong because of properties that are not tracked by the type-checker, and that there are programs that while they cannot go wrong cannot be type-checked. The impulse for making static typing less partial and more complete causes type systems to become overly complicated and exotic as witnessed by concepts such as “phantom types” [11] and “wobbly types” [10]. This is like trying to run a marathon with a ball and chain tied to your leg and triumphantly shouting that you nearly made it even though you bailed out after the first mile. Advocates of dynamically typed languages argue that static typing is too rigid, and that the softness of dynamically languages makes them ideally suited for prototyping systems with changing or unknown requirements, or that interact with other systems that change unpredictably (data and application integration). Of course, dynamically typed languages are indispensable for dealing with truly dynamic program behavior such as method interception, dynamic loading, mobile code, runtime reflection, etc. In the mother of all papers on scripting [16], John Ousterhout argues that statically typed systems programming languages make code less reusable, more verbose, not more safe, and less expressive than			

	dynamically typed scripting languages. This argument is parroted literally by many proponents of dynamically typed scripting languages. We argue that this is a fallacy and falls into the same category as arguing that the essence of declarative programming is eliminating assignment. Or as John Hughes says [8], it is a logical impossibility to make a language more powerful by omitting features. Defending the fact that delaying all type-checking to runtime is a good thing, is playing ostrich tactics with the fact that errors should be caught as early in the development process as possible. Examples: my_number (from class), basic data-types etc.			
2 (a)	Define and explain scope and lifetime of names.	[5]	CO1	L1
ans	The scope of a variable defines the section of the code in which the variable is visible. As a general rule, variables that are defined within a block are not accessible outside that block. The lifetime of a variable refers to how long the variable exists before it is destroyed. Destroying variables refers to deallocating the memory that was allotted to the variables when declaring it.			
(b)	Briefly describe the types of scopes. What type of scoping does C support ?	[5]	CO1	L1
ans	Static and dynamic (at compile time v/s at run time) C supports static scoping. Partial marks for local/global scope, but not the correct answer.			
3	Describe normal-order evaluation with an example (in any language). How is it different from associative-order evaluation (with respect to substitution).	[10]	CO1	L2
ans	In normal-order, evaluation of the inputs are only done if their value is required. In associative-order (or applicative order), all input values are evaluated at the beginning itself – 7 marks Simple example to demonstrate the difference: (define (try a b) (if (= a 0) 1 b)) (try 0 (/ 1 0)) will give error if associative order (it will try to evaluate 0/1, even though the value will not be required).			
4	Briefly describe currying of a procedure with an example. What are its advantages ?	[10]	CO1	L1
ans	Currying: converting a procedure that takes multiple inputs at once to a series of procedures that each take 1 input and output the next procedure – 3 marks example: curried addition of 3 numbers – 3 marks (define add (lambda (a b) (+ a b))) on currying: (define cadd (lambda a (lambda b (+ a b)))) Advantages: partial application, etc.			

5 (a)	What is functional programming ? List the constraints enforced in functional programming.	[5]	CO1	L1
ans	In computer science, functional programming is a programming paradigm—a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data. It is a declarative programming paradigm, which means programming is done with expressions[1] or declarations[2] instead of statements. In functional code, the output value of a function depends only on the arguments that are passed to the function, so calling a function f twice with the same value for an argument x will produce the same result f(x) each time; this is in contrast to procedures depending on a local or global state, which may produce different results at different times when called with the same arguments but a different program state. Eliminating side effects, i.e. changes in state that do not depend on the function inputs, can make it much easier to understand and predict the behavior of a program, which is one of the key motivations for the development of functional programming. Constraints: non-mutability and stateless.			
(b)	List the advantages and disadvantages of functional programming.	[5]	CO1	L1
ans	Advantages: • The style of functional programming is to describe what you want, rather than how to get it. ie: instead of creating a for-loop with an iterator variable and marching through an array doing something to each cell, you'd say the equivalent of "this label refers to a version of this array where this function has been done on all the elements." • Functional programming moves more basic programming ideas into the compiler, ideas such as list comprehensions and caching. • The biggest benefit of Functional programming is brevity, because code can be more concise. A functional program doesn't create an iterator variable to be the center of a loop, so this and other kinds of overhead are eliminated from your code. • The other major benefit is concurrency, which is easier to do with functional programming because the compiler is taking care of most of the operations which used to require manually setting up state variables (like the iterator in a loop). • Since a function can only output a value at most, testing debugging, etc become very simple. Disadvantages: • Functional programming languages are typically less efficient in their use of CPU and memory than imperative languages such as C and Pascal.[59] This is related to the fact that some mutable data structures like arrays have a very straightforward implementation using present hardware (which is a highly evolved Turing machine). Flat arrays may be accessed very efficiently with deeply pipelined CPUs, prefetched efficiently through caches (with no complex pointer chasing), or handled with SIMD instructions. It is also not easy to create their equally efficient general-purpose immutable counterparts. For purely functional languages, the worst-case slowdown is logarithmic in the number of			

memory cells used, because mutable memory can be represented by a purely functional data structure with logarithmic access time (such as a balanced tree).[60] However, such slowdowns are not universal.

- Modularity is harder. Simulating states is also non-trivial.

6 (a)	Write a procedure to take 2 lists as input and output the list-merge of them.	[5]	CO2	L3
ans	(define (list-merge l1 l2) (if (null? l1) l2 (cons (first l1) (list-merge (rest l1) l2))))			
(b)	Write a procedure to take a list as input and reverse it.	[5]	CO2	L3
ans	(define (list-rev l) (if (null? l) l (list-merge (list-rev (rest l)) (cons (first l) '()))))			
7	Write a procedure to implement list-linear-search. Take a list and element as input and output the first index of the element in the list. Make appropriate assumptions as required. Write signature and some test cases to demonstrate usage.	[10]	CO2	L3
ans	Signature: (list-search lst ele -1), this will output -1 if element ele is not there in the list lst, else outputs the pos of ele in lst. Example: (list-search `(1 3 2 4 5) 2 -1), this will output 2. (define list-search (lambda (lst ele pos) (if (null? lst) pos (if (= (first lst) ele) (+ pos 1) (list-search (rest lst) ele (+ pos 1))))))			
8	Evaluate the following expressions: (a) (* (- 34 27) 11) (b) (- 12 (- (- 7 11) 3)) (c) (/ 42 (* 24 5)) (d) (first (rest (cons 5 6))) (e) (if (null? 5) (if (num? 5) “yes” “no”) “not null”)	[2*5]	CO2	L2
ans	a) 77, note that this is prefix notation. b) 19			

	c) 7/20 d) error, first and rest take lists as input but cons returns a pair, not list (its 2nd input is not a list) e) not null			
9	Give inductive definitions for the following sets of data: (a) Set of lists of integers. Assume, definition for integer is provided. (b) S = { 3n+2 | n in N, the set of natural numbers}	[5+5]	CO1	L3
ans	a) `() is a list of integers if lst is a list of integers and ele is an integer, then (cons ele lst) is a list of integers. b) smallest value of n is 1 (natural numbers). Hence, the smallest element in S is 5. Given a valid number, the next number is 3 more than previous. Therefore 5 is in S if k is in S, then k+3 is in S.			
10	Define a representation of all the integers (negative and nonnega-tive) as diff-trees, where a diff-tree is a list defined by the grammar: Diff-tree :: = (one) | (diff Diff-tree Diff-tree) The list (one) represents 1. If t_1 represents n_1 and t_2 represents n_2 , then (diff t_1 t_2) is a representation of $n_1 - n_2$. So both (one) and (diff (one) (diff (one) (one))) are representations of 1; (diff (diff (one) (one)) (one)) is a representation of − 1. For this representation, give procedure to add two integers. The procedure must do the addition in constant time (i.e.. no recursion).	[10]	CO2	L3