23MCAE234	FUNCTIONAL PROGRAMMING	L	T	P	J	S	C	Year of Introduction
		3	1			3	4	2023

Preamble: This course introduces a functional programming approach in problem solving. Salient features of functional programming like recursion, pattern matching, higher order functions are discussed.
Lists and their features, new types such as Recursive types, Enumerated types, Composite and Abstract types along with their applications are being discussed with high importance Haskell is introduced to give a practical flavour to the course

Prerequisite: Discrete mathematics

Course Outcomes: After the completion of the course the student will be able to

CO 1	Understand the principles of functional programming (Module 1)
CO 2	Write purely functional programs, using recursion, pattern matching, and higher- order functions ((Module 2) .
CO 3	Design immutable data structures like lists. (Module 3)
CO 4	Understand generic types for functional programs (Module 4)
CO 5	Write programs using Haskell (Module 5)

CO - PO MAPPING

CO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO 1	2	2	2			2			2			
CO 2	2	2	2			2			2			
CO 3	2	2	2			2			2			
CO 4	2	2	2			2			2			
CO 5	2	2	2		2	2	2		2			2

Assessment Pattern

Bloom's Category	Continuous Assessment Tools			End Semester Examination
	Test1	Test 2	Other tools	
Remember	✔	✔	✔	✔
Understand	✔	✔	✔	✔
Apply	✔	✔	✔	✔
Analyse			✔	
Evaluate				
Create				

Mark Distribution of CIA

Course Structure	Attendance	Theory [L- T]	Total Marks

[L-T-P-J]		Assignment	Test-1	Test-2	
3-1-0-0	5	15	10	10	**40**

Total Mark distribution

Total Marks	CIA (Marks)	ESE (Marks)	ESE Duration
100	40	60	3 hours

End Semester Examination [ESE]: Pattern

PATTERN	PART A	PART B	ESE Marks
PATTERN 1	10 Questions, each question carries 2 marks Marks: (2x10 =20 marks)	2 questions will be given from each module, out of which 1 question should be answered. Each question can have a maximum of 2 sub divisions. Each question carries 8 marks. Marks: (5x8 = 40 marks) Time: 3 hours	60
	Total Marks: 20	Total Marks: [5x8 = 40 marks]	

SYLLABUS

MODULE I : Review of recursion

Tail recursion -recursive program design- Functional Programming: Introduction, λ calculus, λ expressions, Identity function, Self application function, Function application function, Notation for naming functions and application reduction, Functions from functions, Argument selection and argument pairing functions,Free and bound variables,Name clashes and α conversion, Simplification through eta reduction, Conditions, Booleans and Integers, Recursion and Arithmetic, Expressions and values, Basic Data Types , Names and values in programming- Data structures in functional languages - Names and values in imperative and functional languages- Execution order in imperative and functional languages- Repetition in imperative and functional languages- Functions as values.

(Note : Recursion is a very important technique in functional programming, hence high importance needs to be given to make students understand the essentials of recursive thinking and program design, Basic Lambda (λ) calculus needs to be taught.)

MODULE II : Functions

Functions and definitions, Functional composition, Operators, Inverse functions, Strict and non-strict functions, Type Inference.

(Note : Basic ways of defining functions, how to infer the types of variables and function needs to be taught)

MODULE III : Lists

List notation, List comprehensions, Operations on lists, Map and filter, List patterns

Recursion and Induction: Over natural numbers, Over lists. Operations on lists

(Note : Mathematical Induction based Proofs needs to be taught from the reference text book.)

MODULE IV : New Types

Enumerated types , Composite types , Recursive types , Abstract types , Trees: Binary trees , Binary search trees

(Note : Various definitions of properties of these new types, their property proofs etc needs to be taught.)

MODULE V : Programming with Haskell

Introduction to Haskell, Defining functions: guards, pattern matching and recursion, Lists, strings and tuples, Types and polymorphism, Higher order functions on lists: map, filter, list comprehension, User defined data types:lists, queues, trees

(Note : Students need to be taught how to program using Haskell in this module.)

Text books

1. Richard S. Bird, Philip Wadler, “Introduction to Functional Hall, 1988 Programming”,Prentice (Module 1,2,3,4)
2. Greg Michaelson, “An introduction to functional programming through lambda calculus”, Dover Publications, 2011 (Module 1)
3. Miran Lipovaca “Learn You a Haskell for Great Good!: A Beginner's Guide”, No Starch Press, 1st Edition (15 March 2011) (Module 5)

Reference books

1. Simon Peyton Jones , “The Implementation of Functional Languages” , Prentice Hall.
2. Benjamin C. Pierce, " Types and Programming Languages", MIT Press, 2002
3. https://www.haskell.org/
4. http://learnyouahaskell.com

COURSE CONTENTS AND LECTURE SCHEDULE		
No.		No. of Hours
MODULE 1		
1	Introduction to Algorithm Analysis : Algorithm and its properti Review of recursion -Tail recursion -recursive program design- Functional Programming: Introduction, λ calculus, λ expressions, Identity function, Self application function, Function application function, Notation for naming functions and application reduction, Functions from functions, Argument selection and argument pairing functions, Free and bound variables, Name clashes and α conversion, Simplification through eta reduction, Conditions, Booleans and Integers, Recursion and Arithmetic, Expressions and values, Basic Data Types , Names and values in programming- Data structures in functional languages - Names and values in imperative and functional languages- Execution order in imperative and functional languages- Repetition in imperative and functional languages- Functions as values.	10
MODULE II		
2	Functions: Functions and definitions, Functional composition, Operators, Inverse functions, Strict and non-strict functions, Type Inference.	8
MODULE III		
3	Lists: List notation, List comprehensions, Operations on lists, Map and filter, List patterns, Recursion and Induction: Over natural numbers, Over lists. Operations on lists	10
MODULE IV		
4	New Types : Enumerated types , Composite types , Recursive types, Abstract types , Trees: Binary trees , Binary search trees	10
MODULE V		
5	Programming with Haskell: Introduction to Haskell, Defining functions: guards, pattern matching and recursion, Lists, strings and tuples, Types and polymorphism, Higher order functions on lists: map, filter, list comprehension, User defined data types:lists, queues, trees	10

CO Assessment Questions	
1	a) Design a recursive function to add two numbers. b) Design a tail recursive function to find the nth Fibonacci number. c) Explain the basic differences between imperative style programming and functional style programming d) Analyse each of the following lambda expressions to clarify its structure. If the expression is a function, identify the bound variable and the body expression, and then analyse the body expression. If the expression is an application, identify the function and argument expressions, and then analyse the function and argument expressions: i) λa.(a λb.(b a)) ii) λx.λy.λz.((z x) (z y)) iii) (λf.λg.(λh.(g h) f) λp.λq.p) iv) λfee.λfi.λfo.λfum.(fum (fo (fi fee)) v) (λp.(λq.p λx.(x p)) λi.λj.(j i))
2	a) Explain with the help of examples the various forms of function definitions. b) Explain functional composition with the help of examples c) Deduce the type of the following expression: (.) f g x = f (g x) where . -> Functional Composition
3	a) Predict the output of the following along with detailed explanation on how did you arrive at the answer: A. [(a,b) | a <- [1 . . 8] ; even a; b <- [a + 3. . 4] ; odd b] B. ["Party" | k <- [1 .. 5]] C. [' * ' | i <- [1 .. 3] ; j <- [1, 2]]
	b) Explain any three list operations along with function definitions and examples Note: Questions can be asked to solve problems using list comprehensions, to prove properties on list operations and functions on natural numbers using Mathematical Induction
4	a) Define Natural numbers as a Recursive Type and explain how this definition enumerates all Natural numbers. b) Find the equivalent decimal representation of this value: Succ (Succ (Succ (Succ (Succ (Succ Zero))))) c) Define Fibonacci numbers using Pattern matching. Natural numbers should be represented as a Recursive type. Note: Questions can be asked to prove properties on Binary Trees

	and Binary Search Trees using Structural Induction (Variant of Mathematical Induction
5	a) Duplicate only even numbers among the elements of a list using a Haskell function and explain. You need to do this in two ways; 1. Recursion 2. List Comprehension Example : λ> dupli [1, 2, 3] ANS: [2,2]