Course Title and Code	CS181- Programming II

I. Course Identification and General Information:

Course Title	Programming II	Course Code	CSC 181	Pre-requisite	CSC111
Department	Computer Science	Course Level	3	Credit Hours	3(2+1)

II. Course Description/Topics: The following course topics will be covered.

- Problem-solving, algorithmic design, Introduction to Programming, Structured Program Development, Program Control, Functions, recursion, Arrays, pointers, structures, unions, files. Assignments in algorithm design and translation of algorithms in high level language like C language.
- Effect-free programming
- ○ Function calls have no side effects, facilitating compositional reasoning
- ○ Variables are immutable, preventing unexpected changes to program data by other code
- ○ Data can be freely aliased or copied without introducing unintended effects from mutation
- Processing structured data (e.g., trees) via functions with cases for each data variant
- ○ Associated language constructs such as discriminated unions and pattern-matching over them
- ○ Functions defined over compound data in terms of functions applied to the constituent pieces
- First-class functions (taking, returning, and storing functions)
- Function closures (functions using variables in the enclosing lexical environment)
- ○ Basic meaning and definition -- creating closures at run-time by capturing the environment
- ○ Canonical idioms: call-backs, arguments to iterators, reusable code via function arguments
- ○ Using a closure to encapsulate data in its environment
- ○ Currying and partial application
- Defining higher-order operations on aggregates, especially map, reduce/fold, and filter.
- **PL/Basic Types**
- A type as a set of values together with a set of operations
- o Primitive types (e.g., numbers, Booleans)
- o Compound types built from other types (e.g., records, unions, arrays, lists, functions, references)
- Association of types to variables, arguments, results, and fields
- Type safety and errors caused by using values inconsistently with their intended types
- Goals and limitations of static typing
- o Eliminating some classes of errors without running the program
- o Undecidability means static analysis must conservatively approximate program behavior
- Generic types (parametric polymorphism)
- o Definition
- o Use for generic libraries such as collections
- o Comparison with ad hoc polymorphism (overloading) and subtype polymorphism
- • Complementary benefits of static and dynamic typing
- o Errors early vs. errors late/avoided
- o Enforce invariants during code development and code maintenance vs. postpone typing decisions while prototyping and conveniently allow flexible coding patterns such as heterogeneous collections
- o Avoid misuse of code vs. allow more code reuse
- o Detect incomplete programs vs. allow incomplete programs to run

III. Course Outcomes: Summary of the main learning outcomes for students enrolled in the course.

- Compare and contrast (1) the procedural/functional approach—defining a function for each operation with the function body providing a case for each data variant—and (2) Understand both as defining a matrix of operations and variants.
- Write basic algorithms that avoid assigning to mutable state or considering reference equality. [Usage]
- Write useful functions that take and return other functions. [Usage]

- Correctly reason about variables and lexical scope in a program using function closures. [Usage]
- Define and use iterators and other operations on aggregates, including operations that take functions as arguments.

IV. Required Text:

- C How to Program, ISBN-10: 0-13-612356-2, Year: 2010, Author(s): Paul Deitel, Harvey M. Deitel

V. References: