C-ID Descriptor
Introduction to Programming Concepts and Methodologies for Engineers

Descriptor Details

- **Descriptor Title**: Introduction to Programming Concepts and Methodologies for Engineers
- **C-ID Number**: 120
- **Units**: 4
- **Date of Last Revision**: 10/12/2017 11:44:08 PM GMT+0000

General Description

Introduces (1) the basics of software development using a high level language utilizing programming and (2) the interface of software with the physical world (e.g., the use of sensors).

Prerequisites

Pre-Calculus (C-ID MATH 155)

Corequisites

None

Advisories

Basic knowledge of computer usage.

Content

I. Programming Fundamentals

A. Fundamental programming constructs

Minimum coverage time: 9 hours

Topics

1. Basic syntax and semantics of a higher-level language
2. Variables, types, expressions, and assignment
3. Simple I/O
4. Conditional and iterative control structures
5. Functions and parameter passing
6. Structured decomposition

Learning Outcomes

1. Analyze and explain the behavior of simple programs involving the fundamental programming constructs covered by this unit;
2. Modify and expand short programs that use standard conditional and iterative control structures and functions;
3. Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures, and the definition of functions;
4. Choose appropriate conditional and iteration constructs for a given programming task;
5. Apply the techniques of structured (functional) decomposition to break a program into smaller pieces; and
6. Describe the mechanics of parameter passing.

B. Algorithms and problem-solving

Minimum coverage time: 6 hours

Topics

1. Problem-solving strategies
2. The role of algorithms in the problem-solving process
3. Implementation strategies for algorithms

4. Debugging strategies
5. The concept and properties of algorithms

Learning outcomes

1. Discuss the importance of algorithms in the problem-solving process;
2. Identify the necessary properties of good algorithms;
3. Create algorithms for solving simple problems;
4. Use pseudocode or a programming language to implement, test, and debug algorithms for solving simple problems; and
5. Describe strategies that are useful in debugging.

II. Programming Languages

A. Overview of programming languages

Minimum coverage time: 2 hours

Topics

1. History of programming languages
2. Brief survey of programming paradigms
3. Procedural languages
4. Object-oriented languages

Learning outcomes

1. Summarize the evolution of programming languages illustrating how this history has led to the paradigms available today; and
2. Identify at least one distinguishing characteristic for each of the programming paradigms covered in this unit.

B. Declarations and types

Minimum coverage time: 3 hours

Topics

1. The conception of types as a set of values together with a set of operations
2. Declaration models (binding, visibility, scope, and lifetime)
3. Overview of type-checking

Learning outcomes

1. Explain the value of declaration models, especially with respect to programming-in-the-large;
2. Identify and describe the properties of a variable such as its associated address, value, scope, persistence, and size;
3. Discuss type incompatibility;
4. Demonstrate different forms of binding, visibility, scoping, and lifetime management; and
5. Defend the importance of types and type-checking in providing abstraction and safety.

III. Interface with the Physical World

Minimum coverage time: 10 hours

1. Introduction to basic software development tools
2. Introduction to basic hardware interface development tools
3. Modifying software provided to the students
4. Demonstrating designs and implementations associated with the course topics
5. Demonstrating success at implementing software that manipulates or responds to physical phenomena
6. Demonstrating the use of systematic, basic QA procedures to evaluate design and implementation quality.

Learning outcomes

1. Be able to use basic development tools and IDEs for simple software development interfaced with the physical world.
2. Be able to understand software developed by others, be able to modify such software, and be able to validate the quality of the modifications.
3. Be able to demonstrate the interplay between software and the physical world.
4. Be able to follow simple software QA procedures.

Topics fulfilling these tasks and outcomes could include OOPS and other programming elements.

Lab Activities

No information provided

Objectives

At the conclusion of this course, the student should be able to:

1. Design, implement, test, and debug a program that uses each of the following fundamental programming constructs: basic computation, simple I/O, standard conditional and iterative structures, the definition of functions, and the use of interfaces to the physical world;
2. Use pseudocode or a programming language to implement, test, and debug algorithms for solving simple problems
3. Summarize the evolution of programming languages illustrating how this history has led to the paradigms available today
4. Demonstrate different forms of binding, visibility, scoping, and lifetime management
5. Demonstrate how one may combine software and hardware components in order to respond to physical phenomena and manipulate the physical world.

Evaluation Methods

May include any or all of:

Exams
Quizzes
Programming Projects
Discussions
Class Presentations
Laboratory assignments

Textbooks

Savitch, Walter: Problem Solving with C++ Latest Edition