

The course syllabus of Data Structures and Algorithms (SSD5)

Overviews

As students work their way through this course, they will learn how to program in C++, including how to evaluate, select, and use libraries that implement a variety of algorithms and data structures as well as familiarize themselves with some of the key principles for designing algorithms and data structures. Specifically, after successfully completing this course, students will know how to write C++ programs using templates, classes and objects, pointers and references, and C++ input and output. They will know how to write programs using binary search trees, and pointer and array representations of graphs. In addition, they will learn to use C++ and Standard Template Library (or STL) documentation and reference tools; they will learn about the time and space requirements of various algorithms and data structures, which will help them make sound programming choices. Students will also learn how to implement the design principles of divide-and-conquer, backtracking, and dynamic programming.

Outcomes

The purpose of SSD5 is for students to

1. Learn to program in C++
2. Learn to use the STL (Standard Template Library)
3. Learn to evaluate, select, and use libraries implementing algorithms and data structures
4. Learn key principles of algorithm and data structure design

Students successfully completing SSD5 will be able to

I. Produce

1. C++ programs using, classes, objects, templates, pointers, references and I/O
2. Programs using binary trees and associated algorithms, pointer and array representations of graphs, and hashing algorithms
3. Designs of programming solutions independent of programming languages
4. Classifications of program segments into logarithmic, linear, polynomial, and exponential algorithms

II. Use

1. C++ Standard Template Library facilities in writing large programs including sequential containers, trees, hash tables, stacks, and queues
2. Descriptions of the time and space requirements of algorithms and data structures to make appropriate design decisions

III. Knowledgeably Discuss

1. The notion of asymptotic analysis of algorithms in terms of growth rates
2. The concepts of search, divide-and-conquer, and memorization as algorithm design principles
3. The concept of templates in terms of generic programming

IV. Hold Positions as C++ Programmer

The students successfully completing the course will be able to

1. Participate in project design teams
2. Contribute to various phases of software development from requirements through implementation
3. Design special purpose libraries that implement particular feature sets, for example, business

- logic for a medical records application that maintains patient profiles
4. Trouble-shoot programs and implement fixes for software with performance problems
 5. Port difficult-to-maintain legacy code to smaller, efficient, extensible code

Syllabus

Chapter 1 Introduction (3 hours)

1. Why we need Data Structure?
2. Data Structure Philosophy
3. Concepts and Notations
 - a) Data and Data Structure
 - b) Abstract Data Type and Data Type
4. Algorithms and Programs
5. Algorithm Efficiency and Analysis
6. Summary

Chapter 2 C++ reviews (3 hours)

1. C++ Introduced
 - a) C++ Background
 - b) Compiling and Running a C++ Program
2. Basic C++ Programming
 - a) Specifying Classes
 - b) Input and Output
 - c) The Preprocessor
 - d) A Side-By-Side Example
3. Memory Management
 - a) Pointers
 - b) Parameter Passing Mechanisms
 - c) Dynamic Memory Management
4. Mechanisms for Code Reuse and Abstraction
 - a) Inheritance
 - b) Polymorphism
 - c) Templates
 - d) Exception Handling
5. Summary

Chapter 3 Linear List (Sequential List) (3 hours)

1. Arrays and ADTs of Array
 - a) 1D array
 - b) ADT of array
 - c) 2D and high dimensional arrays
2. Sequential List
 - a) ADT of sequential list
 - b) Applications
3. Polynomial ADT and Representation

- a) Polynomial ADT
- b) Sequential Representation of PolyN
- c) Addition of Polynomial
- 4. Standard Template Library
 - a) STL Overview
 - b) Containers
 - c) Iterators
 - d) Algorithms
- 5. Summary

Chapter 4 Linked List and STL (3 hours)

- 1. Singly Linked List
 - a) Structure of singly linked list
 - b) ListNode Class and List Class definition
 - c) Implementation of functions
 - d) Iterator Class
- 2. Circular List
- 3. Doubly Linked List
- 4. Linked representation of Polynomial
- 5. STL Lists
 - a) STL Lists
 - b) STL Iterator
- 6. Summary

Chapter 5 Stack and Queue (6 hours)

- 1. Stack Specifications
- 2. ADT Stacks and Their Implementations
- 3. Application of Stack
 - a) Application 1: Bracket Matching
 - b) Application 2: A Desk Calculator
 - c) Application 3: Infix Expression to Postfix
- 4. Queues
- 5. ADT Queues and Linear Implementation
- 6. Circular Implementations of Queues
- 7. Linked Queue Implementation
- 8. Demonstration and Testing
 - a) Application 1: Fibonacci Array
 - b) Application 2: Yangvi Triangle
 - c) Application 3: Airport Simulation
- 9. Summary

Chapter 6 Recursion (3 hours)

- 1. Introduce to Recursion
- 2. Principle of Recursion

3. Recursion and Iteration
4. The Tower of Hanoi
5. The Maze
6. Eight Queens Puzzle
7. Summary

Chapter 7 Sparse matrix and General List (3 hours)

1. Compact storage of special matrix
2. Sparse Matrix and ADT
 - a) Triple list
 - b) Cross linked list and Addition
3. Transposition and Multiplication of sparse matrix
4. General list and ADT
5. Representation and recursive algorithm
6. Summary

Chapter 8 Tree and Binary Trees (6 hours)

1. Tree Definition and Some Concepts
2. Binary Tree
3. Binary Tree Traversal
4. Binary Tree Reconstruction
5. Threaded Binary Tree
6. Tree Representation and Forest
7. Huffman Tree and Coding
8. Summary

Chapter 9 Graph and Algorithms (6 hours)

1. Graph Definition and Concepts
2. Graph ADT and Storage Implementation
3. Graph Traversal and Connectivity
4. Mini Spanning Tree
5. Shortest Path
6. Topological Sorting and Critical Path
7. Summary

Chapter 10 Searching and Dictionary (6 hours)

1. Concepts of searching
2. Static Searching Table
 - a) Sequential searching table
 - b) Dichotomy searching in sorted table
 - c) Indexed sequential searching table
3. Dynamic Searching Table
 - a) Binary searching tree
 - b) Balanced binary searching tree (AVL tree)

- c) Splay Tree
 - d) M-branch balanced searching trees (B-Tree and B+-Tree)
 - e) Trie Tree
4. Hash Searching Table
 5. Summary

Chapter 11 Sorting (6 hours)

1. Introduction of Sorting
2. Internal Sorting
 - a) Insertion Sorting
 - b) Exchange and Quick Sorting
 - c) Selection Sorting
 - d) Merging Sorting
 - e) Radix Sorting
3. External Sorting
4. Summary

Schedule

1. Lecture 01: Introduction
2. Lecture 02: C++ reviews
3. Lecture 03: Linear Structure and STL overview
4. Lecture 04: Linked list and STL implementation
5. Lecture 05: Stack and Queue (STL implementation)
6. Lecture 6: Recursion (Principle and application)
7. Lecture 07: Sparse Matrix & General List
8. Lecture 08: Tree and Binary tree (1)
9. Lecture 09: Tree and Binary tree (2)
10. Lecture 10: Graph and algorithms (1)
11. Lecture 11: Graph and algorithms (2)
12. Lecture 12: Searching and dictionary (1)
13. Lecture 13: Searching and dictionary (2)
14. Lecture 14: Sorting (1)
15. Lecture 15: Sorting (2)
16. Lecture 16: Conclusion and review

Textbook

Mark Allen Weiss, Data Structures and Problem Solving Using C++, Second Edition, published by Addison Wesley Longman, 2000.

Herbert Schildt, C++: The Complete Reference, Fourth Edition, published by McGraw-Hill/Osborne, 2003.

Exercises and practical assignments

Recommended Exercise1 (rex1): Homework List

Recommended Exercise2 (rex2): Mangling Memory

Recommended Exercise3 (rex3): Safe Array

Recommended Exercise4 (rex4): Parking Lot Simulation

Optional exercise 1 (oex1): Enhanced Linked List: Find and Remove

Optional exercise 2 (oex2): Printer Simulation: FIFO

Optional exercise 3 (oex3): Assessing Infection

Optional exercise 4 (oex4): Spellchecking

Optional exercise 5 (oex5) Stable vs. Non-Stable Sorting

Optional exercise 6 (oex6): Calculating "The Sum of Its Parts"

Optional exercise 7 (oex7): Europe by Rail

Exercise1 (ex1): Building the Core Classes

Exercise2 (ex2): Posting Advertisements

Exercise3 (ex3): Viewing Advertisements by Category

Exercise4 (ex4): Sorting and Keyword Filtering

Exercise5 (ex5): Maintaining Bids

Exercise6 (ex6): Refactoring

Grading

The tentative grading break-down (subject to change) is as follows:

1. 20% : exercises and practical assignments
2. 20% : exam 1 (multiple choice and programming practical)
3. 20% : exam 2 (multiple choice and programming practical)
4. 20% : exam 3 (multiple choice and programming practical)
5. 20% : final exam