SSD5: Data Structures and Algorithms

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.

This course requires the following texts:

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

  Note: Weiss (2000) will serve as the course's textbook and will supplement the course's online materials.

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

  Note: Schildt (2003) will serve as a general reference tool, *not* as a course textbook. Although you are not required to learn its material as well as that of the course textbook, gaining familiarity with such a tool is still an important part of learning to program in any computer language.

Reading assignments are included at the start of each course section. Alternatively, you can refer to Appendix A. Readings for a consolidated list of readings.

Hardware/Software Requirements

- A C++ compiler. The examples and assessment handouts successfully compile under Microsoft Visual C++ 6.0 (Service Pack 5), Microsoft Visual C++ .NET, and the GNU C++ compiler, version 2.95.3-5. The GNU C++ compiler can be obtained free of charge through the

  Cygwin

  environment.

  Note to users of Microsoft Visual C++ 6.0:

  Certain versions of Microsoft Visual C++ 6.0 incorrectly report an error during the compilation of a class, when a "using namespace std" directive is placed before a friend operator declaration. This bug was corrected in Visual Studio 6.0 Service Pack 3. For more information about Visual Studio service packs, please see the following articles in the Microsoft Knowledge Base:

  194022 INFO: Visual Studio 6.0 Service Packs, What, Where, Why

  194295 HOWTO: Tell That Visual Studio 6.0 Service Packs Are Installed

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

Appendix A. Readings

- Mark Allen Weiss, Data Structures and Problem Solving Using C++, published by Addison Wesley Longman, 2000.
- Herbert Schildt, C++: The Complete Reference, Fourth Edition, published by McGraw-Hill/Osborne, 2003.

Section	Weiss	Schildt
1.1: C++ Introduced	None	Chapter 11
1.2: Data Structures and Algorithms	None	None
1.3: Basic C++ Programming	Chapter 2	Chapters 12, 19 - 21
1.4: Memory Management	Chapter 1	Chapters 13 - 15
1.5: Mechanisms for Code Reuse and Abstraction	Chapters 3, 4	Chapters 16 - 18
2.1: Using the Standard `string` Class	None	Chapter 36 (Reference)
2.2: The STL and Basic Containers	Sections 1.2, 2.6	Chapter 36 (Reference)
2.3: Linked Lists	Section 7.6, Chapter 17	Chapters 33 - 37 (Reference)
2.4: Queues	Section 7.2.3, Chapter 16	Chapters 33 - 37 (Reference)
2.5: Stacks	Sections 7.2.1 - 7.2.2, Chapter 16	Chapters 33 - 37 (Reference)
3.1: The Basic Concept of Recursion	Sections 8.1 - 8.3	Chapters 33 - 37 (Reference)
3.2: Problem Solving with Recursion	Sections 8.5, 8.7	Chapters 33 - 37 (Reference)
4.1: Sorting and Searching	Chapters 9, 20	Chapters 33 - 37 (Reference)
4.2: Complexity	Chapter 6	Chapters 33 - 37 (Reference)
5.1: Trees	Chapter 18, Sections 7.7 - 7.9, 19.1	Chapters 33 - 37 (Reference)
5.2: Graphs	Chapter 15	Chapters 33 - 37 (Reference)

Appendix B. Coding Conventions

Throughout this course, unless otherwise specified by your instructor, use the conventions described in this page.

Use the following prototype for function `main()`.

```
1 int main(int argc, char* argv[]);
```

Listing 1 Prototype for `main`

Note that return type `int` is required for function `main`.

Function `main` should terminate specifically by executing one of the following two statements.

```
// indicates abnormal termination
return EXIT_FAILURE;

// ...

// indicates normal termination
return EXIT_SUCCESS;
```

Listing 2 Returning from function `main`

The constants `EXIT_SUCCESS` and `EXIT_FAILURE` are defined in library cstdlib.

Use `void` as the formal parameter list of any function that takes no arguments.

Use file extension `.cpp` for C++ source files. Use file extension `.h` for C++ header files.

Use object `cout` for normal program output. Use object `cerr` for error messages. Both of these objects are found in library iostream.

So that code is self-documenting, choose identifiers carefully. Provide documentation only where tricks or obscure code is used, or to delimit large code segments. The amount of documentation should not rival the amount of code.