CMPSC 122

Intermediate Programming

Required Course in Computer Science and Computer Engineering

Catalog Data: Intermediate Programming (3)
Object-oriented programming, recursion, fundamental data structures (including stacks, queues, linked lists, hash tables, trees, and graphs), the basics of algorithmic analysis, and an introduction to the principles of language translation. Prerequisite: CMPSC 121.


Course Objectives: CMPSC 122 introduces the fundamental concepts of program design, object-oriented programming and data structures. Topics covered include advanced C++ concepts such as classes, inheritance, virtual methods, public/private/static class members, constructors, destructors, operators, templates and Standard Template Library (STL), exceptions, heap & stack-based memory allocation, program and project organization, code debugging and variable watching. Advanced data structure topics include sequential containers (vectors and lists), stacks and queues, recursive algorithms, trees, sorting algorithms, and hash tables. This course uses C++ as the object-oriented computer language to prepare computer science and computer engineering majors to meet immediate demands in solving computational problems and produce high-quality and efficient code. Given a problem statement, the students will follow a structured problem solving methodology consisting of systematic steps that break the problem into inputs, outputs, and processes, and design, code, debug and test the software to produce an error-free source code that is well organized, readable and efficient.

Primary Course Outcomes: Upon completion of the course, students should possess the following skills:

- Class Programming: Given an object-based entity with specific functionality, students should be able to create an equivalent C++ class to represent it.
- Generic Programming: Given a concrete problem students should be able to derive a generic solution using templates.
- Memory Management: Students should be able to choose an appropriate allocation technique (stack or heap-based) and write code that manages memory efficiently while avoiding memory leaks.
- Coding Standard: Students should be able to follow a coding standard necessary for producing robust, efficient, well-structured and readable code.
- Data Types: Students should understand how fundamental data types such as vectors, linked-lists, stacks, queues and trees work, considerations for using them, and be able to employ the corresponding STL templates in their code when necessary. Students should be able to design their own templates corresponding to these data types.
- Testing and Debugging: Students should be able to test and debug the code they develop to ensure correctness and to repair runtime errors.

Relationship to Undergraduate Program Outcomes: CMPSC 122 is the second in a sequence of programming courses which collectively support the following program outcomes:

- Develop a modest (on the order of a thousand lines of code) software application, using appropriate data structures and algorithms.
- Analyze algorithms or computer code for correctness.

Required Topics: Problem analysis, decomposition, and solution synthesis. (~1.5 hrs)
Testing and debugging. (~1.5 hrs)
Source code and project organization, coding standard. (~1.5 hrs)
Equivalence of arrays and pointers, const pointers, references. (~1.5 hrs)
Stack and heap allocation, exceptions. (~1.5 hrs)
Introduction to classes and object-oriented design, public, private class members. (~1.5 hrs)
Constructors: default, init, copy; destructors. (~1.5 hrs)
Instance data vs. static data, virtual methods and inheritance. (~1.5 hrs)
Templates & operators (~3 hrs)
Vectors. (~3 hrs)
Linked Lists. (~1.5 hrs)
Queues and Dequeues. (~3 hrs)
Recursion. (~1.5 hrs)
Trees. (~3 hrs)
Sorting. (~1.5 hrs)
Hash tables. (~3 hrs)

Class Format: Two lecture/labs per week. Each lecture/lab is 75 minutes.

Professional Component: CMPSC122 is designed to aid in the professional development of engineers by developing skills in problem solving, critical thinking, algorithm design, and program implementation. Although C++ will be used to demonstrate these skills, many of the concepts can be applied to general engineering problems and writing programs in other languages as well.

Evaluation: 50 – 70 % proctored assessments (exams/labs)
30 – 50% unproctored assignments (programming projects, take-home labs, etc)

Suggested breakdown based on 100 points as follows:
30 Class / lab work and attendance
30 Homework (3 assignments worth 10 points each)
20 Tests (4 tests worth 5 points each)
20 Final / individual project
1 Extra points awarded for correct answers for questions asked in class

Programming Assignments: Tentative Schedule

<table>
<thead>
<tr>
<th>Goals</th>
<th>Assigned</th>
<th>Due</th>
<th>Worth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedural programming, arrays &amp; pointers</td>
<td>L2</td>
<td>L7</td>
<td>10</td>
</tr>
<tr>
<td>Object-oriented programming, classes</td>
<td>L7</td>
<td>L10</td>
<td>10</td>
</tr>
<tr>
<td>Advanced class programming, memory management</td>
<td>L10</td>
<td>L14</td>
<td>10</td>
</tr>
</tbody>
</table>

Author: Max Fomitchev
Last Revised: February 18, 2007