I. **OVERVIEW**

The following information will appear in the 2010 - 2011 catalog.

**CMPSC 261 Problem Solving and Programming 2**

*Prerequisite:* Satisfactory completion of CMPSC 205.

Introduction to data structures implemented using object-oriented design. Includes more advanced features of high-level languages such as C++ or Java. Continued emphasis on good programming methodologies and problem solving techniques and analysis. Emphasis on algorithm efficiency, recursive algorithms, and linked lists, stacks, queues, and trees.

Field trips might be required.  (A-F or P/NP - Student choice) Lecture /Lab

**Transfer:** (CSU, UC) **General Education:** (MJC-GE: D2)

II. **LEARNING CONTEXT**

Given the following learning context, the student who satisfactorily completes this course should be able to achieve the goals specified in Section III, Desired Learning:

A. **COURSE CONTENT**

1. **Required Content:**

   a. Review of Structured Programming Principles
      i. Program control structures
      ii. Logical expression
      iii. Standard data types
      iv. String processing
      v. Structured data types
      vi. I/O control

   b. Review of Modularity and Program Design
      i. Stepwise refinement
      ii. Loop invariance
      iii. Debugging and testing

   c. Modularity in a Programming Language
      i. Subprograms
      ii. Parameter passing mechanisms
      iii. Run-time stack
      iv. Scope rules
d. Recursion and Recursive Algorithms
   i. Recursion versus repetition
   ii. Recursive subprograms and the run-time stack

e. Simple Data Structures
   i. Stacks, queues, and ordered lists in static structures
   ii. Pointers and dynamic data structures
   iii. Stacks, queues, and ordered lists in dynamic structures
   iv. Tree structures

f. Sorting and Searching
   i. Measures of efficiency
   ii. Internal sorting algorithms
   iii. Table organization and searching

2. Required Lab Content:

a. Review of Structured Programming Principles
   i. Program control structures
   ii. Logical expression
   iii. Standard data types
   iv. String processing
   v. Structured data types
   vi. I/O control

b. Review of Modularity and Program Design
   i. Stepwise refinement
   ii. Loop invariance
   iii. Debugging and testing

c. Modularity in a Programming Language
   i. Subprograms
   ii. Parameter passing mechanisms
   iii. Run-time stack
iv. Scope rules

d. Recursion and Recursive Algorithms
   i. Recursion versus repetition
   ii. Recursive subprograms and the run-time stack

e. Simple Data Structures
   i. Stacks, queues, and ordered lists in static structures
   ii. Pointers and dynamic data structures
   iii. Stacks, queues, and ordered lists in dynamic structures
   iv. Tree structures

f. Sorting and Searching
   i. Measures of efficiency
   ii. Internal sorting algorithms
   iii. Table organization and searching

B. ENROLLMENT RESTRICTIONS

1. Prerequisites
   Satisfactory completion of CMPSC 205.

2. Requisite Skills
   Before entering the course, the student will be able to:
   a. Computer Science background equivalent to ACM CS 1 course.

C. HOURS AND UNITS

<table>
<thead>
<tr>
<th>INST METHOD</th>
<th>TERM HOURS</th>
<th>UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lect</td>
<td>54</td>
<td>3.00</td>
</tr>
<tr>
<td>Lab</td>
<td>54</td>
<td>1.00</td>
</tr>
<tr>
<td>Disc</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

D. METHODS OF INSTRUCTION (TYPICAL)
   Instructors of the course might conduct the course using the following method:
   1. Lecture and Discussion Methods
   2. Classroom Demonstrations
   3. Technology Presentations
   4. Question and Answer Sessions
5. Lab Demonstrations
6. Practical Hands-on Exercises
7. Independent Study through readings

E. ASSIGNMENTS (TYPICAL)

1. EVIDENCE OF APPROPRIATE WORKLOAD FOR COURSE UNITS
   Time spent on coursework in addition to hours of instruction (lecture hours)

   a. (Weekly) Categorize and analyze assigned topic readings.
   b. (Weekly) Carefully analyze and follow detailed instructions for completion of assignments.
   c. (Weekly) Quizzing on problem solving and programming concepts and terminology.
   d. (Weekly) Lab activities utilizing computing technology to apply software engineering and programming concepts.
   e. (Weekly) Design algorithmic solutions based on programming design and software engineering concepts.
   f. (Per term) Several exams at strategic points during the term.

2. EVIDENCE OF CRITICAL THINKING
   Assignments require the appropriate level of critical thinking

   a. Assignment Question: Suppose that queue is implemented as an array with a special reserved slot discussed in class. Also, suppose that the size of the array implementing queue is 100. If the value of queueFront is 50, what is the position of the first queue element?
   b. Assignment Question: Both the merge sort and quick sort algorithms sort a list by partitioning it. Explain how the merge sort algorithm differs from the quick sort algorithm in partitioning the list.
   c. Lab Project: Write and test a recursive method to print a (single) linked list backward. Add your method to the interface LinkedListADT and provide its definition in the class LinkedListClass.
   d. Lab Project: Write a method, singleParent, that returns the number of nodes in a binary tree that have only one child. Add this method to the class BinaryTree and create a program to test this method.
   e. Example Quiz/Exam Questions
      i. Answer true or false: Every recursive definition must have one or more base cases.
      ii. Answer true or false: In a linked list, the nodes are always inserted either in the beginning or at the end because a linked list is not a random access data structure.
      iii. Answer true or false: The inorder traversal of a binary tree always outputs the data in ascending order.
      iv. Evaluate the following postfix expression: 12 25 1 / / * 8 7 + - =
      v. There are 14 different binary trees that have four nodes. Draw all of them.

F. TEXTS AND OTHER READINGS (TYPICAL)

III. **DESIRED LEARNING**

A. **COURSE GOAL**
   
   As a result of satisfactory completion of this course, the student should be prepared to:
   
   evaluate the computation complexity of an algorithm using mathematical formulation and Big-O notation; evaluate and construct static and dynamic lists, stacks, and queues implemented with an object-oriented programming language; construct and evaluate dynamic implementations of binary search trees; and evaluate and construct recursive algorithms and explain how the run-time stack make this possible.

B. **STUDENT LEARNING GOALS**

   Mastery of the following learning goals will enable the student to achieve the overall course goal.

   1. **Required Learning Goals**
      
      Upon satisfactory completion of this course, the student will be able to:
      
      a. Select and create compound data types such as arrays, strings, record structures, and enumerated types needed to represent the data requirements of complex problems using a high-level language such as C++ or Java.
      
      b. Evaluate the need for user-defined data types within a complex problem and construct such data structures using a high-level language such as C++ or Java.
      
      c. Design and implement advanced algorithmic solutions to moderately complex problems.
      
      d. Evaluate and measure the complexity of an algorithmic process such as sorting or searching a data set.
      
      e. Evaluate the computational complexity of an algorithm using mathematical formulation and Big-O notation.
      
      f. Discuss process preconditions, post conditions, and invariance.
      
      g. Employ parameter-passing mechanisms to ensure data integrity and reduce side effects.
      
      h. Explain scope rules.
      
      i. Explain and implement recursive algorithms and explain how the run-time stack makes this possible.
      
      j. Compare and contrast recursive versus iterative processes.
      
      k. Design and implement static lists, stacks, and queues using a high-level language such as C++ or Java.
      
      l. Design and implement dynamic lists, stack, and queues using a high-level language such as C++ or Java.
      
      m. Design and implement dynamic binary search trees using a high-level language such as C++ or Java.
      
      n. Compare and contrast procedural versus object-oriented problem solving.

   2. **Lab Learning Goals**
      
      Upon satisfactory completion of the lab portion of this course, the student will be able to:
      
      a. Analyze and implement compound data types such as arrays, strings, classes, and enumerated types needed to represent the data requirements of complex problems using a high-level language such as C++ or Java.
b. Examine the need for user-defined data types within a complex problem and construct such data structures using a high-level language such as C++ or Java.

c. Design and implement advanced algorithmic solutions to moderately complex problems.

d. Evaluate the complexity of an algorithmic process such as sorting or searching a data set.

e. Evaluate the computational complexity of an algorithm using mathematical formulation and Big-O notation.

f. Formulate process preconditions, post conditions, and invariance.

g. Employ parameter-passing mechanisms to ensure data integrity and reduce side effects.

h. Explain scope rules as applied to methods, classes, and complex programs.

i. Explain and implement recursive algorithms and demonstrate how the run-time stack makes this possible.

j. Evaluate run-time overhead incurred using recursive versus iterative processes.

k. Design and implement static lists, stacks, and queues using a high-level language such as C++ or Java.

l. Design and implement dynamic lists, stack, and queues using a high-level language such as C++ or Java.

m. Design and implement dynamic binary search trees using a high-level language such as C++ or Java.

IV. METHODS OF ASSESSMENT (TYPICAL)

A. FORMATIVE ASSESSMENT

1. Assignments
2. Quizzes
3. Lab Activities
4. Exams

B. SUMMATIVE ASSESSMENT

1. Assignments
2. Quizzes
3. Lab Activities
4. Exams