CENG 707
Data Structures and Algorithms

Tolga Can

Department of Computer Engineering

Middle East Technical University

Fall 2011

CENG 707

Instructor: Tolga Can

Office: B-109

Email: tcan@ceng.metu.edu.tr

Lecture Hours: Thursday 18:00-21:00

Course Web Page:

http://www.ceng.metu.edu.tr/~tcan/ceng707_f1112

Teaching Assistant: To be announced

Course Description

Course Objectives: To introduce abstract concepts for data organization and manipulation, to show how these concepts are useful in problem solving.

Prerequisite: C programming

Textbook: Mark Allen Weiss, *Data Structures and Algorithm Analysis in C++ (3rd ed.)*, Addison Wesley, 2006.

References

- Data Structures and Algorithm Analysis in C, by Mark Allen Weiss, Addison-Wesley, 1997.
- Problem Solving and Program Design in C, 3rd edition Hanly and Koffman, Addison-Wesley.
- Data Structures, Algorithms & Software Principles in C, by T.A. Standish, Addison Wesley, 1995 .

Course Outline

1. Introduction: C and C++ Review
2. Algorithm analysis
3. Recursion
4. Searching and Sorting
5. Stacks
6. Queues
7. Linked Lists
8. Trees
9. Hash Tables
10. Graphs

Grading

- Midterm Exam 1 20%
- Midterm Exam 2 20%
- Final Exam 30%
- Programming Assignments 30%

Policies

- Policy on missed midterm:
  - no make-up exam
- Lateness policy:
  - Late assignments are penalized 10% per day.
- All assignments and programs are to be your own work. No group projects or assignments are allowed.

Introduction

- Computers are devices that mainly do the following:

  They carry out *algorithms* to process information.

- To computer scientists, the algorithm is the central unifying concept of computing, the mode of thought that is the core of the computing perspective.

ALGORITHM

- A set of logical steps to accomplish a task.
- A “recipe of action”.
- A way of describing *behavior*.

Correct Algorithm

- It must correctly solve the problem for any valid input data.
- For the same input data, it must always give the same answer.
- Invalid input data should produce an error message or some other indication that the algorithm cannot correctly solve the problem. It should not produce an answer when given incorrect data since the user will think that the answer is valid.

Abstraction

Idea: Define/implement the general idea, isolate the details.

- The steps in the algorithm should be grouped into related modules or blocks.
- You may use one module inside another module.
- You may refer to other algorithms by name instead of including all of their steps in the current algorithm.

Levels of Abstraction

- Well-designed algorithms will be organized in terms of abstraction.
- The simple instructions that make up each major logical step are hidden inside ***modules***.
- By hiding the details inside appropriate modules, we can understand the main ideas without being distracted.

Computational Abstractions

- **Problem**: Calculating a letter grade for the course, based on a student's various numerical scores (exam, quiz, and hw) and on the weights assigned to each

- **Inputs**: Student's name, hw average, quiz average, exam score, their respective weights

- **Output**: Letter grade for the student

Algorithm: Calculate Grade

1. Get data(student_name, hw_avg, quiz_avg, exam_score)
2. num_grade = Calc_Avg(hw_avg, quiz_avg, exam_score)
3. letter_grade = Calc_Ltr_Grade(num_grade)
4. Output_Grade(student_name,letter_grade)

Software Development

1. Problem Understanding
   - Read the problem carefully and try to understand what is required for its solution.
2. Analysis
   - Identify problem inputs and outputs.
3. Design (Top-down design)
   - Map out the modular structure of your algorithm. Give a descriptive identifier for each module.
   - Refine your modular design. Does each module do only one logical task? Subdivide any module that does not, as many times as necessary.
   - Define the interface of each module before writing any code.
   - Begin the bottom-up work of constructing each module.

4. Implementation
   - Implement the algorithm as a (C/C++) program.
   - Convert steps of the algorithm into programming language statements.
5. Testing and Verification
   - Test the completed program, and verify that it works as expected.
   - Use different test cases (not one) including critical test cases.

Exercise

- Write an algorithm to find the minimum of three numbers.

1. Problem Understanding
2. Analysis: What are the inputs and outputs?
3. Design: How to solve the problem.
4. Implementation: in C
5. Verification and Testing.

Algorithm 1

1. Read three numbers into a, b and c.
2. Keep a variable to hold the minimum value.
3. if (a < b) then min ← a

   else min ← b
4. if (c < min) then min ← c
5. Print min.

Algorithm 2

1. Read three numbers into a, b and c.
2. if (a < b) and (a < c) then print a
   else if (b < c) then print b
   else print c.

Algorithm Components

1. Data structures to hold data.
2. Data manipulation instructions to change data values.
3. Conditional expressions to make decisions
4. Control structures to act on decisions.
5. Modules to make the abstraction manageable by abstraction.