CPE 112 - Computer Methods in Engineering

Chapter 1
Course Basics

- Assignment:
  - Read Chapter 1,
  - Read pages 164-170 on Functional Decomposition ➔ Important
- At the end of each chapter, study Exam Preparation, Quick Check and programming warm-up Exercises and Case study follow up questions.

- All Class communications will be handled via ANGEL

ANGEL Course Management System

- ANGEL
  - Account username MAY be different from your engineering account username. Password will be different as well.
  - Your ANGEL username is the same as your uah email username (username@uah.edu or username@email.uah.edu). Password is the same as for accessing your UAH email account. https://angel.uah.edu/default.asp

- Example login and program submission
Remote Access

- Remote access of the Linux server (blackhawk) is possible. See the handout best suited for your machine type

- To edit a file type the command: gedit filename

- Editors to use are gedit, vim (vi) or emacs. However, there are limitations on these editors when accessing the Linux servers remotely.

INTRODUCTION to C++

- Programming styles are very individualistic
- Need to develop your own style (within reason) and stay with it
- Debugging is an art form of sorts. Several tools are available to help with finding where errors occur.

- Comment Statements
  - // at the beginning of a line means the entire line is a comment
  - // in the middle of a line means the rest of a line is a comment
  - /* text */ all text between the two markers is a comment. The text can be multiple lines
    // this is a comment line
    and this is not
    /* this is a comment
    and so is this
    */
Overview of Computer Programming

- Computer Programming is telling computers what to do in a language that they understand (instructions/coding).
- Writing and maintaining a program consists of 3 (4) phases
  0 **Problem Definition** - Purpose for writing the program
  1 **Problem Solving** (All done by hand) – most important step
    - **Analysis and specification** - Understand the problem and solution
    - **General Solution** - An Algorithm. Logical steps solving a problem
    - **Verify** - Determine if the solution solves the problem.
  2 **Implementation**
    - **Concrete solution** - convert algorithm into a programming language
    - **Test** - verify that the computer results are those expected. Fix all syntax and logic errors that are discovered
  3 **Maintenance**
    - **Use** (execute/run) the program
    - **Maintain** - Modify the program to meet changing requirements

Overview of Computer Programming

- **Algorithm** - A step by step procedure for solving a problem in a finite amount of time.
- **Car Starting and Weekly Wage algorithms** - page 4 of book
- **Algorithm example (Baking a cake)**
  Obtain all ingredients - SubAlgorithm
  Pre-heat oven to 350 degrees
  Mix cake batter components together - SubAlgorithm
  Pour mix into pan
  Cook mix for a fixed amount of time
  Cool cake for 2 hours
  Make frosting - SubAlgorithm
  Frost Cake
  Eat
What is Computer Programming?

- Algorithms are translated (coded)
  - Into different programming languages (C++, Pascal, Fortran, …)
  - by different people in different ways
- A Programming Language is a simplified version of English that adheres to a strict set of rules
- Documentation is a very important part of programming
  - External - Specifications, development history, design document
  - Internal – Comments
- Implementation - coding and testing of an algorithm. Do not jump immediately to the implementation phase by avoiding the Problem-solving phase
- Think first and code later

Computer Terms and Definitions

- **Machine Language** - binary coded (1’s and 0’s) instructions used by the computer
- **Assembly Language** - Low-level programming language using mnemonics to represent machine language instructions (ADD, SUB)
- **Assembler** - Program to translate assembly language to machine language
- **High Level Language** - Highly structured language close to English that can be translated to many different machine languages by compilers
- **Compiler** - Program that translates a high-level language to machine language
  - Source Program - Program written in a high-level language
  - Object Program - Machine language (executable) version of the source program.
Control Structures

- Four ways of structuring statements (instructions) in most programming languages. These control structures allow for the expression of algorithms as programs. Most programs will contain 2 or more of these structures
  - **Sequentially** - instructions executed one after the other
  - **Selection** (if, switch) - conditional control - pick one of multiple choices
  - **Loop** (while, for, etc)- perform same instructions repetitively until a condition is met
  - **Subprogram** (function)- pass control over to another set of instructions/code. Allows for breaking up a large program into smaller units/programs

Ethics and Responsibilities in the Computing Profession

- **Software Piracy** - Copying of software without permission of its creator
- **Privacy of Data** - Do not take advantage of special access to confidential data
- **Use of computer resources** - Do not use computer resources without permission (i.e. hacking, viruses)
- **Software Engineering** - Programmers have a responsibility for developing software that is free from errors
Problem Solving Techniques

- **Ask Questions** - Make sure you understand the problem
- **Look for things that are familiar** - re-use functions
- **Solve by analogy** - Solving a similar problem may help with the current problem
- **Means-Ends Analysis** - Define the ends and then analyze the way(s) (means) of getting from one end to the other
- **Divide and Conquer** - Break large problems up into smaller ones
- **Building-Block Approach** - Combine solutions of smaller problems to solve larger problems. Combination of look for things that are familiar and divide and conquer approaches
- **Merge Solutions** – Combine existing solutions on a step-by-step basis.
- **Mental Blocks** - Avoid by writing down the problem and start looking at individual parts to the problem.

Problem-Solving a Case Study

- **Problem:** Need to write a program that can be used to determine whether a year is a leap year
  - Need clear instructions
  - Leap years are divisible by 4, but not by 100
  - Leap years are divisible by 400

- **Use divide and conquer to solve - three obvious first steps:**
  - Obtain the data
  - Compute the results
  - Output the results
Case Study (continued)

Algorithm for Program
• Obtain a four-digit year – Sub-Algorithm
• Test the year to see if it is a leap year – Sub-Algorithm
• Write out if it is a leap year or not

Sub-Algorithms

• Obtain Data
  • Prompt for a year
  • Read in the value entered

• Determine if year is a leap year – This algorithm is implemented as a function in the example
  • Is the year divisible by 4?
    • No – then not a leap year
    • Yes – is the year divisible by 100
      • No – then it is a leap year
      • Yes – is the year divisible by 400
        • No – then it is not a leap year
        • Yes – then it is a leap year
Sub-Algorithms (continued)

• Output data
  • Output if the year is a leap year or not

• Actual C++ code is listed in the book pages 33-36.

Sample Header Format

    // ****************************************
    // ProgramTitle: Sample.cpp
    // Name: Ron Bowman
    // Course Section: CPE-112-01 (section 1 of the course)
    // Lab Section: 4
    // Due Date: 01/12/05
    // program description: What does the program do?
    // Inputs: four-digit year
    // Outputs: if year is a leap year
    // Assumptions: four-digit year is entered
    // Functions called: IsLeapYear
    // ****************************************
Chapter Summary

- Computers are dumb, they must be told what to do. And they do exactly what they are told - even if it is incorrect!
- Computers are tools used to solve problems
- Think about strategies for solving problems before you start writing algorithms and before writing code (repeat)

- Study Quick check and Exam preparation Exercises in Ch. 1
- Read Chapter 2

Algorithm Example

- Write an Algorithm for driving from engineering building parking lot to the nearest airport.
  Leave school grounds
    go south in parking lot and exit to the west
    turn left onto road (Lakeside).
    turn right at Y in road
    go to stop light and turn left onto Sparkman Drive
  Drive to airport
    From Sparkman Drive, turn right onto 565 West
    Take airport exit #8
    Follow signs to departing flight drop-off
Functional Decomposition – Pages 164-170

• Work from the abstract – A list of the major steps in a solution, to the particular – algorithmic steps that can be translated directly into C++ code.

• Take major steps and break them down into smaller size pieces which become sub-problems.
  • Sub-problems may be reduced further into smaller sub-problems
  • Creates a hierarchical or tree structure to the problem

• See figure 4-4 on page 167

• Concrete Step – a step that has enough implementation details that can be coded directly into C++

• Abstract Step – a step for which some implementation details remain unspecified – further sub-dividing is necessary

• Module – A self contained collection of steps that solves a problem or subproblem

Functional Decomposition Continued

• Properly written final modules contain concrete steps only

• If a module contains concrete and abstract steps, the abstract steps require additional sub-problem analysis in a new module
  • Modules may contain concrete and abstract steps
  • Final modules along a tree branch contain only concrete steps

• Look at figure 4-3 on page 165 for an example of a top module with 3 abstract steps.
Functional Decomposition Example

- Write a F.D. to read an invoice number, quantity ordered and unit price. Use this information to compute the total price. The following information should be output with identifying phrases: the invoice number, quantity, unit price and total price
- Calculate product price (Program name – top module - Level 0)
  - Input data ← abstract (step in top module - Level 0)
    - Prompt for invoice number, read invoice number ← concrete(Level 1)
    - Prompt for quantity ordered, read quantity ordered ← concrete(Level 1)
    - Prompt for unit price, read unit price ← concrete(Level 1)
  - Compute total price: total = item_price*quantity; ← concrete(Level 0)
  - Output information ← abstract (step in top module - Level 0)
    - Output invoice number ← concrete(Level 1)
    - Output quantity ← concrete(Level 1)
    - Output unit price and total price ← concrete(Level 1)

Functional Decomposition Example

- Calculate product price (Level 0)
  - Input data (abstract)
  - Compute total price: total = item_price*quantity;
  - Output information (abstract)

- Input Data (Level 1)
  - Prompt for invoice number, read invoice number
  - Prompt for quantity ordered, read quantity ordered
  - Prompt for unit price, read unit price

- Output information (Level 1)
  - Output invoice number
  - Output quantity
  - Output unit price and total price
Sample Program

```cpp
#include <iostream>
using namespace std;
int Square( int );
int Cube( int );
int main()
{
    cout << "The square of 27 is " << Square(27) << endl;
    cout << "and the cube of 27 is " << Cube(27) << endl;
    return 0;
}
int Square( int n )
{
    return n * n;
}
int Cube( int n)
{
    return n * n * n;
}
```

Sample Program Analysis – Part 1

Statements:  
```cpp
#include <iostream>
using namespace std;
```

Output:  
<no visible output>

Comments:  
Statements tell the preprocessor and compiler where to find required information on the built-in functions used by the program
Sample Program Analysis – Part 2

Statement:  
\[
\begin{align*}
\text{int Square}( \text{int} ); \\
\text{int Cube}( \text{int} );
\end{align*}
\]

Output:  
<no visible output>

Comments:  
Tell the compiler that two user defined functions \text{Square} and \text{Cube} will be used that each take one integer argument and return an integer value upon completion.

Sample Program Analysis – Part 3

Statement:  
\[
\text{cout} \ll \text{“The square of 27 is “} \ll \text{Square}(27) \ll \text{endl;}
\]

Output:  
The square of 27 is 729

Comments:  
During execution, the function \text{Square} is invoked.  
- \text{Square} returns the integer value 729  
- \text{cout} prints the string “The square of 27 is “, followed by the integer 729, followed by the new line character.
Sample Program Analysis – Part 4

Statement:
  \texttt{cout} << "and the cube of 27 is " << \texttt{Cube(27)} << \texttt{endl};

Output: and the cube of 27 is 19683

Comments: During execution, the function \texttt{Cube} is invoked
  - \texttt{Cube} returns the integer value 19683
  - \texttt{cout} prints the string “and the cube of 27 is “,
    followed by the integer 19683,
    followed by the newline character.

Sample Program Analysis – Part 5

Statement: \texttt{return 0;}

Output: <no visible output>

Comments: \texttt{return} sends the integer value 0 back to the
  calling function, in this case the operating
  system, telling it that this function (main)
  executed successfully
Sample Program Analysis – Part 6

Statement:    int Square( int n )
             {
                return n * n;
             }

Output:      <no visible output>

Comments:    Function definition for Square. This definition shows that the function has
             one integer parameter and returns an integer value that is the square of the integer
             parameter

Sample Program Analysis – Part 7

Statement:    int Cube( int n)
             {
                return n * n * n;
             }

Output:      <no visible output>

Comments:    Function definition for Cube. This definition shows that the function has
             one integer parameter and returns an integer value that is the cube of the integer
             parameter
Examples of Defective Software

• Ariane 5 (June 4, 1996)
  – Loss of attitude/guidance information after liftoff
  – Software specification and design errors
  – Defect: Untrapped Numeric Overflow
    • Conversion of a 64-bit floating point value to a 16-bit signed integer value
  – Result:
    Loss of rocket and cargo valued at $500 million

Source: //www.ima.umn.edu/~arnold/disasters/ariane.html

Examples of Defective Software

• Mars Climate Observer (September 23, 1999)
  – Erroneous steering commands lead to loss of spacecraft and failure of mission costing $125 million (Mixing English and metric units)

• Mars Polar Lander (December 7, 1999)
  – Defective software shuts off engines prematurely leading to loss of spacecraft

• Common Defects:
  – Software specification/design errors
  – Project Management: “Faster, Cheaper, Better”

Examples of Defective Software

- **Therac-25** (June 1985 - January 1987)
  - Computerized radiation therapy machine
  - Defective control software
  - Relied on software, not hardware safety interlocks
  - Six known massive radiation overdoses
  - Results: radiation burns and patient deaths