C Programming
for Scientists and Engineers

Chapter 1 Notes
Key Concepts

- Modularity
- Top-down design
- Programs = data + operations on data
- Algorithms
- Features of a program
- Types of data
- Structure charts, pseudo-code, flowcharts
- Structured Programming
Top-down design

- Provides an approach to solving complex problems
- Break into sub-problems and iterate
Modular Programming

• Breaking down a complex problem into easily manageable functional units
• Then break each module into sub-modules
• Continue (iterate) until sub-modules are easy to realize
• Based on the concept of "divide and conquer"
• Modules should be loosely coupled - changes to one module do not force changes to other modules
• Modules should be tightly specified - perform a very well defined function
Modular Programming

• Modules can be functions (in C)
• Note functions can call other functions
• Have to pass data back and forth between modules
• Data is passed into one function and then other data passed back
• Modules can be written, compiled, debugged, and tested independently. Then they are linked together to form executable code.
• Modules can be input, computational, output
• Modules can be reused from one program to the next
• Functions can be reused within the program (called from different places)
• C has a rich library of pre-written modules; e.g., input values, compare "strings", output values
• Modules (functions) can be used from other people's "library"
Algorithms

• Program development involves the use of algorithms
• Definition: a procedure consisting of a finite number of precisely defined steps for solving a problem
• Each step must be unambiguous - clear and precise when written in a computer language, can be executed by a computer
• Order of steps is critical
• Algorithm must terminate (or not as in the case of embedded systems)
• Choose algorithm based on efficiency, accuracy, reliability, and *clarity*
  – Efficiency w.r.t. computational time, response time, storage requirements
  – Accuracy - degree required is specified by user (compute to X digits)
  – Reliability - consistently produces correct answers from valid data and rejects invalid data
  – Clarity - written in easy to understand style. Makes code easier to write, modify, understand, debug and maintain
  – Must sometimes compromise on these (EXCEPT NOT CLARITY)
• Algorithms for simple problems can be straightforward, but for complex problems can be very difficult
From Algorithm to Program

• A program is a sequence of executable, unambiguous instructions written in a computer language

• Computer understands instructions of many types:
  – Input/output: to/from keyboard, I/O port, disk, network, monitor
  – Data move: from one memory location to another (assignment operator: a = b. not stating that “a” is “b”)
  – Arithmetic: + - * / % (precedence rules apply)
  – Logical &&, ||, ! (explained later)
  – Control:
    • if (brightness > 150) {speed = speed+5;} else {speed = speed-5;}
    • Test involves comparison; result is true or false
Data
Collection of related information to be processed

• Can be:
  – Numeric: Integer (signed or unsigned – no negatives) or "floating point"
  – Character: single character ('a') or strings ("abcde")
  – Logical: either true or false. C uses integers for this: 0 = false, anything else is true
    • C++ has boolean type called "bool"
• Constants:
  – Integer: 1, 42, 1000, -7, 0
  – Floating point: 1., -2.3, 3.14159, 0.314159E-1
  – Characters: 'a', 'A', '0', ';', '+', '!
  – Strings: "abc", "I like programming", "1.7", "7 + 4", "a", "+", ""
• Variables:
  – Contain values of various types
  – C is strongly typed: each variable is given a type;
    • e.g., a, A, count, tempval, i, x, x0
    • Different types used for different things e.g., count is an integer, speed is a floating point, name is a string
• Structures:
  – Collections of data into one entity
  – E.g., Employee name, employee ID, date of hire, salary,
• Input data should be validated
• Output data should be formatted for easy understanding and use (e.g., tabular)
• Many values are purely internal
Procedure for Problem Analysis

• First write a precise problem specification, algorithm, input & output
  – Include initial conditions and boundary conditions
  – Include equations or formulas to be used
• You must precisely understand the problem before programming
• Solution Design Methodology: At the design stage, the algorithm can be represented by
  – Structure charts
  – Pseudo-code and/or
  – Flowcharts, or similar
Structure charts

- Reflect the relationship of the modules
- Expand where necessary
- Executed from left to right
Pseudo-code

• A semi-formal way to express an algorithm
• Conventions:
  – Initialize summations and counters and other variables
  – Assignment of values is shown by an arrow pointing left
  – Decisions are written with a condition followed by indented action optionally followed by one or more "Else" with indented actions ended with "Endif" (or similar)
  – Repetition is written with a condition (at the beginning or end) and indented action ended with "Endfor" (or similar)
  – Words can optionally be substituted for arithmetic
  – If algorithm terminates indicate that clearly
Flow Chart Symbols

- Start/stop: oval or rounded corners
- Process box containing a statement: rectangular box
- Function box: rectangular box with double left/right edges
  - function details described elsewhere
- Input/output: parallelogram
- Decision box: diamond
- Connectors: small circle
  - with letter & arrow in or out for jumps
- Author also adds connectors: small circles where arrows intersect
- Flow lines: arrows
Flow Charts

• Flow is from top to bottom
• Each box (except "Start") has single input at the top
• All but decision (and "Stop") have a single arrow out at bottom
• Steps in a flowchart can be described by language or mathematical notation
• Possibilities you should know
  – Sequence
  – Repetition
    • Decision at the top
    • Decision at the bottom
  – Selection
    • if/then
    • if/then/else
Structured Programming

• Only use the control structures:
  – Sequence - a series of steps executed in order, no skipping
  – Selection - optionally take one action, or choose among two or more actions
    • No more than one action will be taken
    • Action is executed only once
    • Might have, e.g., a sequence or repetition embedded as one of the actions
    • Might have another selection as one or more of the actions
  – Repetition - one (zero) or more instructions executed multiple times
    • while - test first and do action; if true and loop back, else go on
      – The action might not be done at all
    • do while - do action first then test and repeat if true, else go on
      – The action must be done at least once
      – Usually the test is based on the results of the action

• Structures can be "stacked" or "nested".
  – Stacked is really just nested in a sequence (execute one after another)
Program Processing

Note fix errors (and warnings!) at each step

• Write "source code" and documentation
  – Write and test one module at a time
  – Add comments as you go along
    • Document the purpose of a module, inputs and outputs, formula used and anything else that clarifies the operation of the module
    – Carefully read and test by hand before testing on computer
• Compile code - code checked for errors and converted to "object code"
  – Sometimes you must iterate: compile, fix errors, repeat
• Object code is then "linked" with other modules (library functions, etc.)
  – Fix errors and recompile
• Executable code is then run - verify proper operation
  – Fix errors and repeat
Testing and documentation

• Testing - Errors can be of many types:
  – Data specification, problem specification, program design, implementation, typing, logic, data input.
  – Errors are detected at different stages of the process. Earlier = cheaper
  – Terminology: program bugs and "debugging"
  – **More good stuff in the book on this section...

• Program documentation is important
  – System documentation - functional descriptions, intro manuals, ref materials, installation manuals, user manuals, etc.
  – Program documentation – document all phases of program development
    • Statement of the problem, Glossary of input/output variables
    • Description of each module, Error messages produced and their cause
    • Security measures to be incorporated, test data to use when modifying
  – Some documentation is part of the source code.
  – Documentation changes as program evolves