CMPT 125

Introduction to Computing Science and Programming II

A Course Overview

Jeffrey Leung

Table of Contents

Basics of C

- Extension: *.c*
- Header files:
    - Extension: *.h*
    - Each *.c* file with at least one declared function (outside of main()) should have a separate header file containing function declarations. This header file is called in whichever file(s) require the declared functions.

- Variables:
    - Strong type (variables must be declared before use)
    - Size in memory:

int/float:	4 bytes
long/double:	8 bytes
char:	1 byte

- Pointer: Variable containing the address of another variable
    - Pointers to a local function variable cannot be returned by a function, because the memory is deallocated when the function is removed from stack memory
    - *<variable> indicates a pointer towards an address
      &<variable> indicates an address
        - *<pointer> dereferences the pointer to access/change the variable
    - Arrays:
        - A sequence of pointers
        - Can only be of one single type
        - Can be passed directly to a function
        - Strings are arrays of characters with a null terminator at the end

- Structures:
    - Used for encapsulation of data and operations

Basics of C++

- Evolved out of C
- Differences with C:
    - Information hiding
    - Encapsulation of data and methods through object-oriented programming

- Extension: *.cpp*
- Header files:
    - Extension: *.hpp*

- Classes:
    - Similar to structures in C
    - Requires a constructor and a destructor
    - Methods and variables are private by default

- Templates:
    - Allow for unspecified typing

Programming

- **Compilation:** Translating an entire program into machine language at once
    - **Machine language:** Code which can be read by the CPU
        - Each instruction is represented by a number in binary
        - No variable names or subroutines
- **Assembly language:** Code using abstract operation codes
    - Higher-level than machine language; lower-level than C
    - **Assembler:** Translates mnemonics and labels into machine language

- Variables:
    - Composed of a type, a value, and an address
    - **Address:** Location of a variable in memory
    - **Array:** Mutable sequence of data
        - Series of pointers
        - Static length
        - Entries are in a contiguous space
        - **Base address:** Address of the first element in an array
        - Expanding an array requires a running time of O(N^2) (see *Chapter 2: Algorithm Performance*)
    - **Pointer:** Variable which contains the address of another variable
        - **Dereference:** Directly accessing/changing the variable located at the value at a pointer address
    - **Strong type:** Variables must be explicitly assigned a specific type
        - Type cannot be changed in some languages (e.g. C)
    - **Weak type:** Variables do not have to be explicitly assigned a specific type

- In some programming languages (e.g. Python), functions pass parameters by reference (i.e. modifying a parameter also modifies its corresponding argument).
  In some programming languages (e.g. C), functions pass parameters by value (i.e. modifying a parameter does not modify its corresponding argument), but passing a pointer will have the same effect as passing by reference.
  In some programming languages (e.g. Java), functions can pass parameters either by reference or by value.
- `\0` Null terminator (end of a string/character array)
  `NULL` Null pointer

- Bug-testing:
    - **Assertion:** Expectation of a characteristic of the code at that given point
        - **Loop invariant:** Assertion in a loop which holds true for every iteration, and must hold true each time for the code to run its intended purpose
        - Precondition: Assertion placed before a block of code
        - Postcondition: Assertion placed after a block of code

Algorithm Performance

- Can be determined by memory usage or by time

- Can be predicted through counting the number of elementary steps required by the program in the worst case – total steps (T) as a function of the input size (N)
    - *Example:*

```
int dup_chk( int a[], int length )                  Times run within its parent:
{
        int i = length;                             1
        while (i > 0)                               N + 1
        {
                i--;                                N
                int j = i – 1;                      N
                while (j  >=  0)                    j + 2 = i +1
                {
                        if ( a[i] == a[j] )         j  + 1 = i
                        {
                                return 1;           0
                        }
                        j--;                         j + 1 = i
                }
        }
        return 0;                                   1
}
```

- *Example: Given a 2-D array (N x N) of integers, find the 10x10 swatch with the largest sum.*
    - Brute-force approach: Run through all possible positions for the upper-left corner of the swatch ((N-10) x (N-10) possibilities) – O(n^2))

- **Big-O notation:**
  - Counts the number of elementary lines of code proportional to the size of the given input, takes only the dominant term, and removes its leading constant
    - Constant factors sometimes matter
      (e.g. $f_1(n) = 20n^2$ requires 10x more time to run than $f_2(n) = 2n^2$)
      and can be affected by:
      - CPU speed
      - Simultaneous tasks by the system
      - Memory characteristics
      - Program optimization
    - Leading constants are important when Big-O growth rates are the same
  - Frequency of the algorithm's 'barometer instructions' will be proportional to its Big-O running time (i.e. the count of the most frequent operations)
  - Based on the worst-case scenario

  - Calculating Big-O running time:
    - Multiply loops together (number of times run × operations run)
    - Substitute function calls with the running time of the function
    - L'Hopital's rule can be applied to most functions
    - Choose the maximum running time of an option from an if/else if/else statement
    - Equals sign: $n^a = O(n^b)$ if and only if $a \leq b$
    - When adding function times, choose the maximum

  - Common growth rates, from fastest to slowest:

Running time:	**Name:**	
$O(1)$	Constant time	Array indexing
$O(\log n)$	Logarithmic time	Usually in base 2; binary search
$O(n)$	Linear time	Linear search
$O(n \log n)$		Quicksort, mergesort
$O(n^2)$	Quadratic time	Selection sort, insertion sort
$O(n^k)$	Polynomial time	k is a constant
$O(k^n)$	Exponential time	

- Examples:

$5n$	$O(n)$
$\log_a n$	$O(\log_b n)$
$2n + 3n^2 + 4 \log n$	$O(n^2)$
$n + 1 + n \log n$	$O(n \log n)$

- Optimizing algorithms:
  - The slower the function grows, the faster the algorithm will run
  - An algorithm may be fundamentally inefficient
  - It is more important to reduce running time by a factor of n than of a constant

Sorting and Searching

- Sorting:
  - Best sorting algorithms run in O($n \log n$)
  - **Selection sort:** Swaps the least/greatest element with the first, second, etc. element
    - Runs in O(n^2)
  - **Insertion sort:** Sorts each element into the beginning section of the array, while shifting any elements in between the sort to the right
    - Runs in O(n^2)
  - **Quick sort:** Recursively sorts the elements in the array except the first depending on whether the elements are greater or less than the first element, switches the 'middle' and first elements, and moves to a half of the new array
    - Runs in O(n^2)
      - Most often runs in O(n log n)
    - Uses a 'pivot' element to separate the part of the array less than the first element, and the part of the array greater than the first element
    - qsort() is implemented in <stdlib.h>
  - **Merge sort:** Recursively halves an array into arrays of 2 elements, sorts, and then merges the arrays by taking the minimum of the leftmost elements
    - Runs in O(n log n)

- Searching:
  - **Linear search:** Moves through the array sequentially beginning from the first element until a match is found
    - Runs in O(n)
    - Elements do not have to be ordered
  - **Binary search:** Compares the middle element to the target and moves to the least section (if the target is less than the element) or the greatest section (if the target is greater than the element)
    - Runs in O($\log_2 n$)
    - Elements must be ordered
    - **Divide and conquer:** Slicing an array into pieces, and using previous knowledge to simplify/solve the problem

Abstract Data Types

- **Stack:** Ordered sequence of data where insert/remove operations work on the same end of the sequence
    - Last-in-first-out (LIFO) order
    - Push: Placing an item on the top of the of the stack
    - Pop: Removing an item from the top of the stack
    - Peek: Checking the value at the top of the stack
    - **Call stack:** Ordered data structure in memory which stores information about active functions of a program
        - Contains parameter return value, local variables, and return address
        - When a function is called (even main()), it is pushed to the stack; when a value (or null) is returned, the function is popped from the stack
    - Simulated by postfix notation (a mathematical expression where an operator is placed after two values)
        - E.g. 1 2 +
        - Numbers are placed on a stack; when an operator is reached, the last two numbers are pulled off the stack, evaluated, and placed back onto the stack

- **Queue:** Ordered sequence of data where insert/remove operations work on opposite ends of the sequence
    - First-in-first-out (FIFO) order
    - E.g. Lineup
    - Enqueue: Adding a value to an end of the list
    - Dequeue: Removing a value from the opposite end of the list as the enqueue operation

- **Heap:** Memory space outside of the stack which can be manually allocated
- **Interface:** Collection of data and behaviours connecting data types
    - Connects input (parameters) to output
    - E.g. Functions, invariants, constants, header files
    - Useful for reducing code repetition and for increasing code independence

- Software engineering principles:
    - **Modularity:** Separating components of a system
    - **Encapsulation:** Packaging related data and operations into a single component
        - Allows selective hiding of properties and implementation details
        - Makes code less likely to be changed in an adverse way

- **Linked lists:** Multiple values scattered throughout memory, linked by a pointer placed directly after each value which points to the next value
  - Benefits of a linked list vs. an array:
    - Adding/removing an element to/from the head, and adding an element to the tail
    - Adding an element in a known place
  - Recursive definition: Composed of its first node and a slightly smaller list

Graphs and Trees

- **Graph:** Diagram which depicts the relationships among a collection of items
  - Vertices: Items in a collection
    - Represented by junctions in a graph
  - Edges: Relations between items in a collection
    - Represented by lines in a graph
  - Connected: A graph where each pair of vertices has a path
  - Disconnected: A graph where at least one pair of vertices are not connected
  - **Depth:** Distance of a node from the root
    - **Height:** Maximum depth of all nodes
- **Tree:** Graph where any two vertices are connected by exactly one path
  - **Rooted tree:** Directed graph where one vertex (the root) has no incoming edges but all other vertices have exactly one incoming edge
    - E.g. Directories in a computer
    - Communicable period: Distance between each node
    - Infection rate: Rate of increase of nodes over time
  - *Post-order traversal:* Evaluating the children of a node before evaluating the node itself
- **M-ary tree:** Tree where each vertexes has *m* or less children
- Binary trees:
  - Recursive definition: T is a binary tree if T is an empty tree, or T has a root vertex with left and/or right binary subtrees.
  - *Full binary tree:* Binary tree where each vertex has either 0 or 2 children
    - *Expression tree:* Full binary tree representing an arithmetic calculation, where each vertex with children holds an operator, and each vertex without children holds a value
      - Uses the postfix notation
      - Evaluated from bottom to top
      - Uses post-order traversal
  - *Binary search tree:* Binary tree where each node contains a value sorted in its position by the rule that for any node, its left subtree contains values ≤ the node, and its right subtree contains values > the node
    - search() or insert:
      - Worst case: O(*n*)
      - Average case: O(*n log n*)
      - Best case: O(*log n*)

Formal/Regular Languages
Finite State Machines

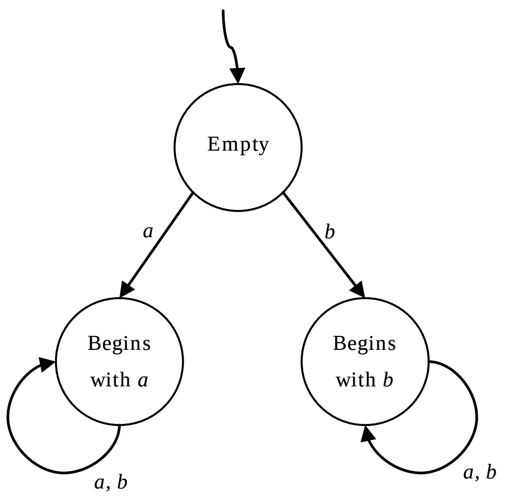
- **Formal language:** Set of valid syntax
  - Exact
  - Expressed mathematically/recursively
  - Input of length 0 is represented by λ (lambda) or ε (epsilon)
  - Grammar determines the rules of interpretation

- **Finite state machine:**
  - Σ denotes the universe of acceptable input

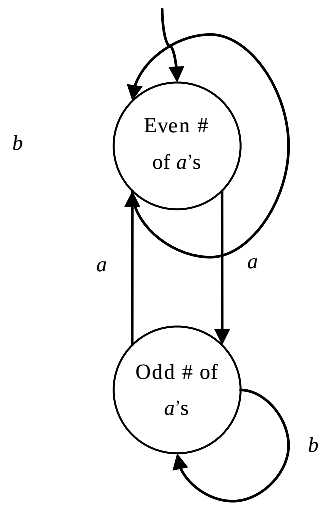
  - *Start state:* Beginning state
    - Often denoted by a curved arrow
  - *Final state:* Optimal state
    - Often denoted by two circles
  - *Dead state:* State from which it is impossible to move to a final state

  - Process:
    - One input at a time
    - Next state is determined through the current state and the next input
    - Accept if the last state is a final state; reject otherwise

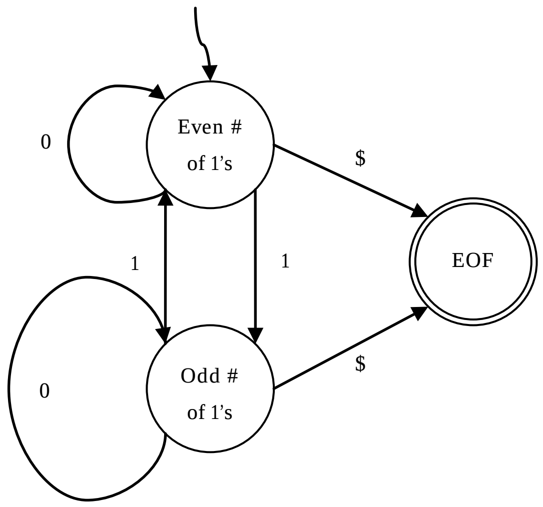
- Examples:
  - $\Sigma = \{ a, b \}$
    Condition: Begins with a/b



$\Sigma = \{ a, b \}$
Condition: Even/odd number of a's



  - $\Sigma = \{ 0, 1 \}$ EOF = \$



- *Transition table:* Visualization of the possible inputs from each state and their results

Current state:	**Input:**