

Notes on Data Structures and Programming Techniques (CPSC 223, Spring 2022)

James Aspnes

2022-05-02T09:36:05-0400

- 1 Course administration
 - 1.1 Overview
 - 1.1.1 License
 - 1.1.2 Resources
 - 1.1.3 Documentation
 - 1.1.4 Questions and comments
 - 1.2 Lecture schedule
 - 1.3 Syllabus
 - 1.3.1 On-line course information
 - 1.3.2 Meeting times
 - 1.3.3 Course synopsis
 - 1.3.4 Prerequisites
 - 1.3.5 Textbook
 - 1.3.6 Course requirements
 - 1.3.7 Staff
 - 1.3.7.1 Instructor
 - 1.3.7.2 Undergraduate Learning Assistants
 - 1.3.8 Use of outside help
 - 1.3.9 Clarifications for homework assignments
 - 1.3.10 Late assignments
 - 1.4 Introduction
 - 1.4.1 Why should you learn to program in C?
 - 1.4.2 Why should you learn about data structures and programming techniques?
 - 2 The Zoo
 - 2.1 Getting an account
 - 2.2 Getting into the room
 - 2.3 Remote use
 - 2.3.1 Terminal access
 - 2.3.2 GUI access
 - 2.4 Developing on your own machine
 - 2.4.1 Linux
 - 2.4.2 OSX
 - 2.4.3 Windows
-

- 3 The Linux programming environment
 - 3.1 The shell
 - 3.1.1 Getting a shell prompt in the Zoo
 - 3.1.2 The Unix filesystem
 - 3.1.3 Unix command-line programs
 - 3.1.4 Stopping and interrupting programs
 - 3.1.5 Running your own programs
 - 3.1.6 Redirecting input and output
 - 3.2 Text editors
 - 3.2.1 Writing C programs with Emacs
 - 3.2.1.1 My favorite Emacs commands
 - 3.2.2 Using Vi instead of Emacs
 - 3.2.2.1 My favorite Vim commands
 - 3.2.2.1.1 Normal mode
 - 3.2.2.1.2 Insert mode
 - 3.2.2.2 Settings
 - 3.3 Compilation tools
 - 3.3.1 The GNU C compiler `gcc`
 - 3.3.2 Make
 - 3.3.2.1 Make gotchas
 - 3.4 Debugging tools
 - 3.4.1 Debugging in general
 - 3.4.2 Assertions
 - 3.4.3 The GNU debugger `gdb`
 - 3.4.3.1 My favorite `gdb` commands
 - 3.4.3.2 Debugging strategies
 - 3.4.3.3 Common applications of `gdb`
 - 3.4.3.3.1 Watching your program run
 - 3.4.3.3.2 Dealing with failed assertions
 - 3.4.3.3.3 Dealing with segmentation faults
 - 3.4.3.3.4 Dealing with infinite loops
 - 3.4.3.3.5 Mysterious variable changes
 - 3.4.4 Valgrind
 - 3.4.4.1 Compilation flags
 - 3.4.4.2 Automated testing
 - 3.4.4.3 Examples of some common valgrind errors
 - 3.4.4.3.1 Uninitialized values
 - 3.4.4.3.2 Bytes definitely lost
 - 3.4.4.3.3 Invalid write or read operations
 - 3.4.5 Not recommended: debugging output
 - 3.5 Performance tuning
 - 3.5.1 Timing under Linux
 - 3.5.2 Profiling with `valgrind`
 - 3.5.3 Profiling with `gprof`
 - 3.5.3.1 Effect of optimization during compilation

- 3.6 Version control
 - 3.6.1 Setting up Git
 - 3.6.2 Editing files
 - 3.6.3 Renaming files
 - 3.6.4 Adding and removing files
 - 3.6.5 Recovering files from the repository
 - 3.6.6 Undoing bad commits
 - 3.6.7 Looking at old versions
 - 3.6.8 More information about Git
- 3.7 Submitting assignments
- 4 The C programming language
 - 4.1 Structure of a C program
 - 4.2 Numeric data types
 - 4.2.1 Integer types in C
 - 4.2.1.1 Basic integer types
 - 4.2.1.2 Overflow and the C standards
 - 4.2.1.3 C99 fixed-width types
 - 4.2.2 `size_t` and `ptrdiff_t`
 - 4.2.2.1 Integer constants
 - 4.2.2.1.1 Naming constants
 - 4.2.2.2 Integer operators
 - 4.2.2.2.1 Arithmetic operators
 - 4.2.2.2.2 Bitwise operators
 - 4.2.2.2.3 Logical operators
 - 4.2.2.2.4 Relational operators
 - 4.2.2.3 Converting to and from strings
 - 4.2.3 Floating-point types
 - 4.2.3.1 Floating point basics
 - 4.2.3.2 Floating-point constants
 - 4.2.3.3 Operators
 - 4.2.3.4 Conversion to and from integer types
 - 4.2.3.5 The IEEE-754 floating-point standard
 - 4.2.3.6 Round-off error
 - 4.2.3.7 Reading and writing floating-point numbers
 - 4.2.3.8 Non-finite numbers in C
 - 4.2.3.9 The math library
 - 4.3 Operator precedence
 - 4.4 Programming style
 - 4.5 Variables
 - 4.5.1 Memory
 - 4.5.2 Variables as names
 - 4.5.2.1 Variable declarations
 - 4.5.2.2 Variable names
 - 4.5.3 Using variables
 - 4.5.4 Initialization
 - 4.5.5 Storage class qualifiers

- 4.6 Input and output
 - 4.6.1 Character streams
 - 4.6.2 Reading and writing single characters
 - 4.6.3 Formatted I/O
 - 4.6.4 Rolling your own I/O routines
 - 4.6.5 Recursive descent parsing
 - 4.6.6 File I/O
- 4.7 Statements and control structures
 - 4.7.1 Simple statements
 - 4.7.2 Compound statements
 - 4.7.2.1 Conditionals
 - 4.7.2.2 Loops
 - 4.7.2.2.1 The while loop
 - 4.7.2.2.2 The do..while loop
 - 4.7.2.2.3 The for loop
 - 4.7.2.2.4 Loops with break, continue, and goto
 - 4.7.2.3 Choosing where to put a loop exit
- 4.8 Functions
 - 4.8.1 Function definitions
 - 4.8.2 When to write a function
 - 4.8.3 Calling a function
 - 4.8.4 The return statement
 - 4.8.5 Function declarations and modules
 - 4.8.6 Static functions
 - 4.8.7 Local variables
 - 4.8.8 Mechanics of function calls
- 4.9 Pointers
 - 4.9.1 Memory and addresses
 - 4.9.2 Pointer variables
 - 4.9.2.1 Declaring a pointer variable
 - 4.9.2.2 Assigning to pointer variables
 - 4.9.2.3 Using a pointer
 - 4.9.2.4 Printing pointers
 - 4.9.3 The null pointer
 - 4.9.4 Pointers and functions
 - 4.9.5 Pointer arithmetic and arrays
 - 4.9.5.1 Arrays
 - 4.9.5.2 Arrays and functions
 - 4.9.5.3 Multidimensional arrays
 - 4.9.5.3.1 Using built-in C arrays
 - 4.9.5.3.2 Using an array of pointers to rows
 - 4.9.5.3.3 Using a one-dimensional array
 - 4.9.5.4 Variable-length arrays
 - 4.9.5.4.1 Why you shouldn't use variable-length arrays
 - 4.9.6 Pointers to void
 - 4.9.6.1 Alignment

- 4.10 Strings
 - 4.10.1 C strings
 - 4.10.2 String constants
 - 4.10.2.1 String encodings
 - 4.10.3 String buffers
 - 4.10.3.1 String buffers and the perils of `gets`
 - 4.10.4 Operations on strings
 - 4.10.5 Finding the length of a string
 - 4.10.5.1 The `strlen` tarpit
 - 4.10.6 Comparing strings
 - 4.10.7 Formatted output to strings
 - 4.10.8 Dynamic allocation of strings
 - 4.10.9 Command-line arguments
 - 4.11 Structured data types
 - 4.11.1 Structs
 - 4.11.1.1 Operations on structs
 - 4.11.1.2 Layout in memory
 - 4.11.1.3 Bit fields
 - 4.11.2 Unions
 - 4.11.3 Enums
 - 4.11.3.1 Specifying particular values
 - 4.11.3.2 What most people do
 - 4.11.3.3 Using `enum` with `union`
 - 4.12 Type aliases using `typedef`
 - 4.12.1 Opaque structs
 - 4.13 Macros
 - 4.13.1 Macros with arguments
 - 4.13.1.1 Multiple arguments
 - 4.13.1.2 Perils of repeating arguments
 - 4.13.1.3 Variable-length argument lists
 - 4.13.1.4 Macros vs. inline functions
 - 4.13.2 Macros that include other macros
 - 4.13.3 More specialized macros
 - 4.13.3.1 Multiple expressions in a macro
 - 4.13.3.2 Non-syntactic macros
 - 4.13.3.3 Multiple statements in one macro
 - 4.13.3.4 String expansion
 - 4.13.3.5 Big macros
 - 4.13.4 Conditional compilation
 - 4.13.5 Defining macros on the command line
 - 4.13.6 The `#if` directive
 - 4.13.7 Debugging macro expansions
 - 4.13.8 Can a macro call a preprocessor command?
 - 5 Data structures and programming techniques
 - 5.1 Asymptotic notation
 - 5.1.1 Two sorting algorithms
 - 5.1.2 Big-O to the rescue
-

- 5 Data structures and programming techniques
 - 5.1 Asymptotic notation
 - 5.1.1 Two sorting algorithms
 - 5.1.2 Big-O to the rescue
 - 5.1.3 Asymptotic cost of programs
 - 5.1.4 Other variants of asymptotic notation
 - 5.2 Dynamic arrays
 - 5.3 Recursion
 - 5.3.1 Example of recursion in C
 - 5.3.2 Common problems with recursion
 - 5.3.2.1 Omitting the base case
 - 5.3.2.2 Blowing out the stack
 - 5.3.2.3 Failure to make progress
 - 5.3.3 Tail-recursion and iteration
 - 5.3.3.1 Binary search: recursive and iterative versions
 - 5.3.4 Mergesort: a recursive sorting algorithm
 - 5.3.5 Asymptotic complexity of recursive functions
 - 5.4 Linked lists
 - 5.4.1 Stacks
 - 5.4.1.1 Building a stack out of an array
 - 5.4.2 Queues
 - 5.4.3 Looping over a linked list
 - 5.4.4 Looping over a linked list backwards
 - 5.4.5 Deques and doubly-linked lists
 - 5.4.5.1 Alternate implementation using a ring buffer
 - 5.4.6 Circular linked lists
 - 5.4.7 What linked lists are and are not good for
 - 5.4.8 Further reading
 - 5.5 Abstract data types
 - 5.5.1 Boxed vs unboxed types
 - 5.5.1.1 A danger with unboxed types
 - 5.5.2 A sequence type
 - 5.5.2.1 Interface
 - 5.5.2.2 Implementation
 - 5.5.2.3 Compiling and linking
 - 5.5.3 Designing abstract data types
 - 5.5.3.1 Parnas's Principle
 - 5.5.3.2 When to build an abstract data type
 - 5.6 Hash tables
 - 5.6.1 Dictionary data types
 - 5.6.2 Basics of hashing
 - 5.6.3 Resolving collisions
 - 5.6.3.1 Chaining
 - 5.6.3.2 Open addressing

- 5.6 Hash tables
 - 5.6.1 Dictionary data types
 - 5.6.2 Basics of hashing
 - 5.6.3 Resolving collisions
 - 5.6.3.1 Chaining
 - 5.6.3.2 Open addressing
 - 5.6.4 Choosing a hash function
 - 5.6.4.1 Division method
 - 5.6.4.2 Multiplication method
 - 5.6.4.3 Universal hashing
 - 5.6.5 Maintaining a constant load factor
 - 5.6.6 Examples
 - 5.6.6.1 A low-overhead hash table using open addressing
 - 5.6.6.2 A string to string dictionary using chaining
- 5.7 Generic containers
 - 5.7.1 Generic dictionary: interface
 - 5.7.2 Generic dictionary: implementation
- 5.8 Object-oriented programming
- 5.9 Trees
 - 5.9.1 Tree basics
 - 5.9.2 Tree-structured data
 - 5.9.3 Binary trees
 - 5.9.4 The canonical binary tree algorithm
 - 5.9.5 Nodes vs leaves
 - 5.9.6 Special classes of binary trees
- 5.10 Heaps
 - 5.10.1 Priority queues
 - 5.10.2 Expensive implementations of priority queues
 - 5.10.3 Structure of a heap
 - 5.10.4 Packed heaps
 - 5.10.5 Bottom-up heapification
 - 5.10.6 Heapsort
 - 5.10.7 More information
- 5.11 Binary search trees
 - 5.11.1 Searching for a node
 - 5.11.2 Inserting a new node
 - 5.11.3 Deleting a node
 - 5.11.4 Costs
- 5.12 Augmented trees
 - 5.12.1 Applications
- 5.13 Balanced trees
 - 5.13.1 Tree rotations
 - 5.13.2 Treaps
 - 5.13.3 AVL trees
 - 5.13.3.1 Sample implementation
 - 5.13.4 2–3 trees

- 5.13 Balanced trees
 - 5.13.1 Tree rotations
 - 5.13.2 Treaps
 - 5.13.3 AVL trees
 - 5.13.3.1 Sample implementation
 - 5.13.4 2–3 trees
 - 5.13.5 Red-black trees
 - 5.13.6 B-trees
 - 5.13.7 Splay trees
 - 5.13.7.1 How splaying works
 - 5.13.7.2 Analysis
 - 5.13.7.3 Other operations
 - 5.13.7.4 Top-down splaying
 - 5.13.7.5 An implementation
 - 5.13.7.6 More information
 - 5.13.8 Scapegoat trees
 - 5.13.9 Skip lists
 - 5.13.10 Implementations
 - 5.14 Graphs
 - 5.14.1 Basic definitions
 - 5.14.2 Why graphs are useful
 - 5.14.3 Operations on graphs
 - 5.14.4 Representations of graphs
 - 5.14.4.1 Adjacency matrices
 - 5.14.4.2 Adjacency lists
 - 5.14.4.2.1 An implementation
 - 5.14.4.3 Implicit representations
 - 5.14.5 Searching for paths in a graph
 - 5.14.5.1 Implementation of depth-first and breadth-first search
 - 5.14.5.2 Combined implementation of depth-first and breadth-first search
 - 5.14.5.3 Other variations on the basic algorithm
 - 5.15 Dynamic programming
 - 5.15.1 Memoization
 - 5.15.2 Dynamic programming
 - 5.15.2.1 More examples
 - 5.15.2.1.1 Longest increasing subsequence
 - 5.15.2.1.2 All-pairs shortest paths
 - 5.15.2.1.3 Longest common subsequence
 - 5.16 Randomization
 - 5.16.1 Generating random values in C
 - 5.16.1.1 The `rand` function from the standard library
 - 5.16.1.1.1 Supplying a seed with `srand`
 - 5.16.1.2 Better pseudorandom number generators
 - 5.16.1.3 Random numbers without the `pseudo`
-

- 5.16.1 Generating random values in C
 - 5.16.1.1 The rand function from the standard library
 - 5.16.1.1.1 Supplying a seed with srand
 - 5.16.1.2 Better pseudorandom number generators
 - 5.16.1.3 Random numbers without the pseudo
 - 5.16.1.4 Range issues
- 5.16.2 Randomized algorithms
 - 5.16.2.1 Randomized search
 - 5.16.2.2 Quickselect and quicksort
- 5.16.3 Randomized data structures
 - 5.16.3.1 Skip lists
 - 5.16.3.2 Universal hash families
- 5.17 String processing
 - 5.17.1 Radix search
 - 5.17.1.1 Tries
 - 5.17.1.1.1 Searching a trie
 - 5.17.1.1.2 Inserting a new element into a trie
 - 5.17.1.1.3 Implementation
 - 5.17.1.2 Patricia trees
 - 5.17.1.3 Ternary search trees
 - 5.17.1.4 More information
 - 5.17.2 Radix sort
 - 5.17.2.1 Bucket sort
 - 5.17.2.2 Classic LSB radix sort
 - 5.17.2.3 MSB radix sort
 - 5.17.2.3.1 Issues with recursion depth
 - 5.17.2.3.2 Implementing the buckets
 - 5.17.2.3.3 Further optimization
 - 5.17.2.3.4 Sample implementation
- 6 Other topics
 - 6.1 More applications of function pointers
 - 6.1.1 Iterators
 - 6.1.1.1 Option 1: Function that returns a sequence
 - 6.1.1.2 Option 2: Iterator with first/done/next operations
 - 6.1.1.3 Option 3: Iterator with function argument
 - 6.1.2 Closures
 - 6.1.3 Objects
 - 6.2 Suffix arrays
 - 6.2.1 Why do we want to do this?
 - 6.2.2 String search algorithms
 - 6.2.3 Suffix trees and suffix arrays
 - 6.2.3.1 Building a suffix array
 - 6.2.3.2 Searching a suffix array
 - 6.3 Burrows-Wheeler transform

- 6 Other topics
 - 6.1 More applications of function pointers
 - 6.1.1 Iterators
 - 6.1.1.1 Option 1: Function that returns a sequence
 - 6.1.1.2 Option 2: Iterator with first/done/next operations
 - 6.1.1.3 Option 3: Iterator with function argument
 - 6.1.2 Closures
 - 6.1.3 Objects
 - 6.2 Suffix arrays
 - 6.2.1 Why do we want to do this?
 - 6.2.2 String search algorithms
 - 6.2.3 Suffix trees and suffix arrays
 - 6.2.3.1 Building a suffix array
 - 6.2.3.2 Searching a suffix array
 - 6.3 Burrows-Wheeler transform
 - 6.3.1 Suffix arrays and the Burrows-Wheeler transform
 - 6.3.2 Sample implementation
 - 6.4 C++
 - 6.4.1 Hello world
 - 6.4.2 References
 - 6.4.3 Function overloading
 - 6.4.4 Classes
 - 6.4.5 Operator overloading
 - 6.4.6 Templates
 - 6.4.7 Exceptions
 - 6.4.8 Storage allocation
 - 6.4.8.1 Storage allocation inside objects
 - 6.4.9 Standard library
 - 6.4.10 Things we haven't talked about
 - 6.5 Testing during development
 - 6.5.1 Unit tests
 - 6.5.1.1 What to put in the test code
 - 6.5.1.2 Example
 - 6.5.2 Test harnesses
 - 6.5.2.1 Module interface
 - 6.5.2.1.1 stack.h
 - 6.5.2.2 Test code
 - 6.5.2.2.1 test-stack.c
 - 6.5.2.3 Makefile
 - 6.5.2.3.1 Makefile
 - 6.5.3 Stub implementation
 - 6.5.3.1 stack.c
 - 6.5.4 Bounded-space implementation

- 6.6 Algorithm design techniques
 - 6.6.1 Basic principles of algorithm design
 - 6.6.2 Specific techniques
 - 6.6.3 Example: Finding the maximum
 - 6.6.4 Example: Sorting
 - 6.7 Bit manipulation
 - 6.8 Persistence
 - 6.8.1 A simple solution using text files
 - 6.8.2 Using a binary file
 - 6.8.3 A version that updates the file in place
 - 6.8.4 An even better version using mmap
 - 6.8.5 Concurrency and fault-tolerance issues: ACIDity
 - 7 What next?
 - 7.1 What's wrong with C
 - 7.2 What C++ fixes
 - 7.3 Other C-like languages
 - 7.4 Scripting languages
 - 8 Assignments
 - 8.1 Assignment 1, due 2022-02-03 at 17:00
 - 8.1.1 Bureaucratic part
 - 8.1.2 Hello, world: A program in honor of Georges Perec
 - 8.1.3 Submitting your assignment
 - 8.1.4 Sample solution
 - 8.2 Assignment 2, due 2022-02-10 at 17:00
 - 8.2.1 Recognizing lowercase letters
 - 8.2.2 Non-ASCII characters
 - 8.2.3 Testing your assignment
 - 8.2.4 Clarifications
 - 8.2.5 Submitting your assignment
 - 8.2.6 Sample solution
 - 8.3 Assignment 3, due 2022-02-17 at 17:00
 - 8.3.1 Testing your assignment
 - 8.3.2 Submitting your assignment
 - 8.3.3 Notes
 - 8.3.4 Clarifications
 - 8.3.5 Sample solution
 - 8.4 Assignment 4, due 2022-02-24 at 17:00
 - 8.4.1 Testing your assignment
 - 8.4.2 Submitting your assignment
 - 8.4.3 Notes
 - 8.4.4 Sample solution
 - 8.5 Assignment 5, due 2022-03-03 at 23:59
 - 8.5.1 Your task
 - 8.5.2 Submitting your assignment
 - 8.5.3 Sample solutions
 - 8.6 Assignment 6, due 2022-03-10 at 23:59
-