DRAFT

64 Steps to a Programmer's Mind

This book is designed for people who already understand computer science basics, but want to improve their thinking and problem solving skills. This book will not teach computer science concepts, but will provide practice problems to better your ability to use them in real life situations. The problems in this book should be solvable in any modern programming language.

The best way to use this book is to write a solution to each problem using your favorite language, then compare your solution with the example solutions. But dont peek at the answers! I can tell you from experience that solving a problem yourself will teach you a LOT more than memorizing the solution. Once you've seen the solution, you won't be able to “unsee” it, if you get stuck, think about the problem until you come up with a solution yourself, even if it takes days of thinking. Once you've finished all the problems in this book, you can use it to practice other programming languages.

When you are doing these problems, be aware of programming best practices. Remember to give your variables meaningful names, break the problems down into appropriately sized functions, and so on....

If your language gives you the choice of making a GUI application or a console application, do all the problems up to the GUI section as console applications. Then when you get to the GUI section, re-make the previous problems into GUI applications.

Part 1 – Conditionals

1. Make a function that takes 3 integer arguments and returns the highest value of the 3 numbers.

2. Modify the function in (1) to return the number with the highest **absolute** value (e.g. -300 > 60).

3. Create a function that returns true if a package is light enough and small enough to be safely shipped. For the package to be safe, the overall volume must be less than 1 cubic meter, and it must weigh less than 20kg. The function should take the arguments: width (in centimeters), height (in centimeters), length (in centimeters), and weight (in kilograms).

4. Create a “tax calculator” function that takes a person's annual salary (integer) and returns the amount of taxes they need to pay. If their salary is equal to or over 100,000, they must pay 10% of their total earnings. If their salary is between 50,000 and 100,000, they must pay 5%. If their salary is 50,000 or less, they must pay 2%.

5. Create a function that tests login information. It will take 2 string arguments, the username and password. It must return true/false whether the login information is correct. There are three valid user accounts: Theresa (Password: “apple1”), Joan (Password: “passw0rd!”), and Bill (Password: “superblade”). For example, if “Theresa” and “apple1” are given as parameters, the function will return true. If the username “Theresa” is given with the password “superblade”, it must return false.

Part 2 – Looping and Recursion

6. Create a program that takes an integer value from the user and outputs all the numbers from 0 to that number. Make a “while” loop solution and a “for” loop solution.

DRAFT

7. Create a program that takes an integer from the user and counts by 2 from 0 to that number. Make one “while” loop solution and one “for” loop solution.

8. Create a program that takes 2 integers from the user and adds up all the integers between those numbers. e.g. If 5 and 12 are given, the program will calculate 5+6+7+8+9+10+11+12.

9. Using recursion (loops are not allowed), make a function that returns the factorial of a number. Then, remake the same function using an iterative solution (uses loops).

10. Create a “bank” application that asks the user (over and over) whether they want to withdraw money from their account, deposit money, view their account balance, or quit the application. If they want to deposit, the program will ask the user how much they are depositing, and add it to their balance. If they are withdrawing, the amount to withdraw is subtracted from their balance. If they try to withdraw more money than they have in their account, it will show an error message and go back to asking them if they want to deposit or withdraw.

11. Create a program that outputs all prime numbers that are lower than 1 million. A prime number is a number that can only be divided by 1 and itself.

Part 3 – String Parsing and Manipulation

12. Create a program that asks the user for their name, and outputs their initials in upper case. e.g. “Nancy Smith” turns into “NS” and “Charles alex Carmichael” turns into “CAC”.

13. Create a string replace program that asks the user for a string, then asks them what they want to replace, and what value to replace it with. The program will then replace all of the values in the string and show the result, along with a count of how many occurrences were replaced. e.g. If the user gives “Hello Jello” and wants to replace “ll” with “abc”, the program will output: “Replaced 'll' with 'abc' 2 times: Heabco Jeabco”.

14. Create a program that takes a string from the user and outputs the ASCII values of each character in the string. e.g. “Apple” becomes “65 112 112 108 101”.

15. Create the reverse of (13), a program that takes a string of space separated ASCII values and outputs the corresponding string. e.g. “65 112 112 108 101” becomes “Apple”.

16. Create a program that asks for a string, and says whether the string is a palindrome or not. A palindrome is a phrase that is spelled the same backwards and forwards. e.g. If the user enters “A dog! A panic in a pagoda!” the program will say “That is a palindrome!”.

Part 4 – Arrays and Other Data Structures

17. Create a program that keeps asking a user for positive integers until they enter '-1'. The program should fill an array with these numbers. When the user enters '-1', the program will output the numbers in reverse order.

DRAFT

18. Create a program that obtains an array of numbers from the user by asking the user how many numbers he wants to put in, then prompting for each number. When the user is done entering numbers, the program should output the sum of every number, the average, the highest number, the lowest number, and all of the numbers sorted in ascending order (write a simple sorting algorithm yourself).

19. Create a function that takes a two dimensional array of integers and returns the sum of every number in the array. e.g. if the array { {1,2} , {2,3}, {3, 4} } is given, the function will return 15.

20. Create a to-do list program that asks the user for a list of tasks, then shows them to the user in the order they were entered, one at a time, giving the user the options “skip” or “task completed” for each task. If the user chooses “skip”, that task won't be shown until all the other tasks have been shown once (e.g. moving it to the “end of the line”). When the user completes a task, it is removed from the list and is never shown again. For this problem you may use your language's “queue” data structure, or make your own.

21. Create a program that takes a sentence from the user, and applies a substitution cipher to that string and outputs the result. You can assume that the input string only contains alphabetical characters. A substitution cipher encrypts data by mappnig each character to another, for example, “A” becomes “R” and “B” becomes “H”, and so on. The mapping using in your application should be randomly generated when the program starts. Remember to make sure that no two letters encrypt to the same value, or else decryption would be impossible!

22. Create a program that prompts the user for an algebraic expression, then solves this equation and outputs the correct value. In an expression, “^” denotes exponentiation and “*” is multiplication. An example expression would be: “((2+3)^(-15+4^2) * 32 + 3/2 - 3)” which should solve to 158.5. Be sure to follow the correct order of operations.

Part 5 – Files

23. Create a program that asks the user for the path to a text file. The program will then count the number of lines, words, and characters in the file. If the file doesn't exist, or an error occurs while reading the file, it must be caught by the program so that the user can be asked for a different filepath.

24. Create a program that replaces all occurrences of a string with another in a text file. The program should ask the user for the path to the file, the string to be replaced, and the string to replace it with.

25. Create a program that calculates the average temperature over each year using a data file provided by the user. The file format will look like this:

AUG2012-45
JAN2010-20
FEB2011-32

The number after the dash is the average temperature of the specific month described by the text before the dash. Remember that the data file can be in the wrong order, AUG2012 may come before JAN2012 and DEC2005. If the data file is missing some months of a year, instead of showing the average temperature for that year, an error message will be shown in its place.

DRAFT

The output of the program should be similar to the following:

Average Temperatures By Year:
2001: 53
2002: 59
2003: 48
2004: ERROR: Missing data for January, March, December.
2005: 38

26. Create a one time pad encryption program that asks the user for the path to a key file, the path to the file to be encrypted/decrypted, the path to the output file, and an offset. To encrypt/decrypt the file, the program will read a byte at a time from the key file (starting at the offset) and from the plaintext/ciphertext file (starting at the beginning of the file). The program will XOR these bytes together, and write the result to the output file. When the program finishes, it will output the position of the next byte in the keyfile to be used as the offset for the next encryption. If the file to be encrypted/decrypted is larger than the remaining bytes in the key file, show an error.

27. Create an interpreter for a simple math scripting language. The program should take a file containing the script as a command line argument, execute the script, and show the result. If there is an error parsing the script, the program should tell the user what line contains the error. The scripting language is defined as follows:

The script has access to four signed integer variables, A, B, C and D. Each line contains one instruction. Everything after a single quote on a line is ignored (comments). The instructions are:

MOV	Destination, Source	Moves the contents of the “Source” variable (or value) into the “Destination” varaible. Example: MOV A, 5 //stores 5 in A MOV A, B //copies the value of B to A
ADD	Destination, Add	Adds the value or variable in the “Add” position to the “Destination” variable. Example: ADD A, 10 //adds 10 to A ADD A, B //adds B to A
SUB	Destination, Sub	Subtracts “Sub” from the “Destination” variable. Example: SUB A, 10 //subtracts 10 from A
MUL	Destination, Mul	Multiplies “Mul” with “Destination”

DRAFT

		Example: MUL A, 20 //multiplies A by 20
PRINT	Output	Prints the contents of the "Output" variable. Example: MOV A, 10 PRINT A will output "10".

Example script:
MOV A,1
MOV B, 2
MOV C, 3
MOV D, 4
ADD A, B
ADD A, C
ADD A, D 'after this, A should contain 10
MUL A, B '10 * 2 = 20
SUB A,5 '15
PRINT A 'should output 15

Part 6 – Object Oriented Programming
-account class, takes username and password as a constructor, has method .ValidateLogin
-bank account (inherit account)
-chromakey

Part 7 – Graphical User Interfaces
-modify all other programs to use GUI
-create an image viewer
-that game made for phill, black hole one, with a button
-listbox, can sort & save order snapshots, revert to old order snapshot
-listview (grid) application
-creating multiple textboxes at runtime and getting their values (custom list)
-user interface that expands and contracts as the window grows/shrinks
-multiple document interfaces
Part 8 – Processes and Threading
Part 9 – Network Communication
-encrypted chat program
-stateless file server (encrypted communication)
-superfast UDP file transfer program
-p2p chat program (with all the crypto)

Part 10 – Putting it Together
-php exploit scanner
-project tracking system

DRAFT

-firedreamerx

-SORTING ALGORITHMS IN ARRAY!!!
-command line argumentsf