352 Final Exam Study Guide

Ultimately, the topics for the final exam includes anything from class, videos, the readings, the PAs, and any other material covered before the exam. This list is just provided as a guide.

Topics

- General UNIX, Linux, bash knowledge
- How to use various bash commands (ls, cd, pwd, cut, grep, cal, echo, printf, sed, etc)
- Standard input, output, error
- Piping and redirection
- GCC and some of the command line arguments used in class (-Wall, -Werror, etc)
- C programming
    - This includes everything covered about C in this course. Language basics, types, functions, I/O, pointers, memory management, l-values and r-values, etc, etc
- Standard library functions
    - You don't have to know every standard library function in the whole library, but you should be familiar with ones used in class and homework assignments. For example, ones from **`stdio.h`** such as printf, fprintf, fopen, fclose, etc.
- Basics about C program layout and the stack
- How to use GDB
- General UNIX file system knowledge
- Navigating a file system via a terminal such as bash
- UNIX files, inodes, inode structure
- File permissions
- Dynamic memory allocation
- Stack and heap
- malloc, calloc, free, etc
- Managing memory
- Valgrind
- Data structures in C (linked lists, trees, graphs, dynamically-sized arrays, etc)
- Make
- Processes, process control, top command, kill command
- Regular expressions, grep, sed
- Shell scripting
- The steps of C compilation
- C preprocessor
- Binary IO and bit manipulation
- Object files, executables, ELF, linking
- Function pointers
- The kernel and system calls

Problems

I am not going to be including solutions for these problems. In order to check if your solutions are correct you can try:

1. Analyzing your solutions yourself after writing
2. Comparing with other students
3. For coding problems: Run and test the code!

For some of the coding questions, you can try implementing two versions - one with the standard library functions allowed (which could make it shorter) and another version with no standard library. You should not rely on only this study guide for final exam preparation.

Problem 1

In this problem, you should write a C function named **`bubble_sort`**. This function should have one parameter of type **`StringLLNode*`**. **`StringLLNode`** represents a node for a linked list of strings, which is defined as:

```
typedef struct StringLLNode {
  char* string;
  struct StringLLNode* next;
} StringLLNode;
```

You can expect that the node that gets passed into the function represents the root of a linked list. The function should perform a bubble-sort on the linked list to get the elements into alphabetical order, based on the string contents in each node. Initially, try implementing this *with the standard* library functions. For example, you could use the **`strcmp`** function for comparing if a string is greater than the other. After implementing this, try accomplishing it *without* any help from standard library functions for additional practice.

‘

Problem 2

In this problem, you should write a C function named **`build_binary_tree`**. This function should have one parameter of type **`char*`** that represents the name of a file to read from. You can expect that the file will have exactly one word per line, and no word will be longer than 25 characters. The function is responsible for constructing a binary tree from these words, and should return the root node of the constructed tree, which should be a **`StringTreeNode*`**. You can expect that a **`StringTreeNode`** is defined as:

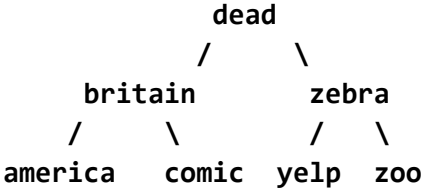
```
typedef struct StringTreeNode {
  char* string;
  struct StringTreeNode* left;
  struct StringTreeNode* right;
} StringTreeNode;
```

The function should read through the strings from the file in order, and add each to the tree. Words that come earlier in alphabetical order should go to the left, and words that go later in alphabetical order should go to the right.

For example, if the input file contained the words:

```
dead
zebra
yelp
britain
america
zoo
comic
```

The resulting tree should be:



Problem 3

In this problem, you should write a C function named **`convert`**. This function should have one parameter of type **`char*`**. You can expect that this parameter will be a string containing only upper and lower case letters. The function should convert any upper-case letters to The character **`'0'`** and lower-case letters to **`'1'`** in the string that was passed in. The function should not return anything (a **`void`** function).

Problem 4

In this problem, you should write a C function named **`count_ones`**. This function should have one parameter of type **`unsigned long`**. The function should count how many 1 digits exist in the binary representation of that **`unsigned long`** number in memory and return the count as an **`int`**. For example, say that on a test system, an **`unsigned long`** was 64 bits, and the **unsigned long** with the value 25 was passed in. The underlying binary of this would be:

`0000000000000000000000000000000000000000000000000000000000011001`

The function should return 3 in this case.

Problem 5

In this problem, you should write a C function named **`max_cross`**. This function should have two parameters: The first of type **`int**`** which you can expect to be a pointer to an array of pointers, which each point to the beginning of an int array. (In this question, I'll refer to this as a 2D array, though technically it is not one). You can expect the second parameter to be an **int** representing the number of rows and columns (you can assume that the 2D array will have an equal number of rows / columns, and every row will have the same number of elements). This function should iterate through every element in the 2D array. For each element, it should sum up all the values in that particular row and column. The function should return the highest of these sums. For example, say that this 3x3 2D array was passed in:

```
10, 20, 30
10, 20, 50
50, 15, 40
```

The function should return 185 in this case. This is because the max row/column sum combination is:

```
10, 20, 30
10, 20, 50
50, 15, 40
```

Problem 6

In this problem, you should write a C function named **`binary_tree_to_binary_file`**. This function should have two parameters. The first should be a **`NumTreeNode*`** representing the root of a binary tree with **`unit32_t`** values in each node. The second parameter should ba a **`char*`** representing the name of a file to write to. A **`NumTreeNode`** is defined as:

```
typedef struct NumTreeNode {
  uint32_t value;
  struct NumTreeNode* left;
  struct NumTreeNode* right;
} NumTreeNode;
```

The function should iterate through the tree via in-order traversal, and write each of the **`value`** values to the file specified by the second parameter. It should write each number in its binary representation, one after the other.

Problem 7

In this problem, you should write a C function named **`reverse_words`**. This function should have one parameter of type **`char*`** which you can expect will be a string with 0 or more words separated by spaces. The function should reverse each individual word within the string. For example, this code:

```
char words[] = "cheetah elephant cat";
reverse_words(words);
printf("%s\n", words);
```

Should print:

```
hateehc tnahpele tac
```

Problem 8

In this problem, you should write a C function named **`merge_numbers`**. This function should have two parameters of type **`char*`** which you can expect will both be paths to text files containing an equal number of integer numbers values 0 - one billion, one number per line. This function should create a file named **`numbers.bin`**. The function should interleave the numbers from the two text files into this binary file, and write each in its 32 bit binary representation.

Problem 9

ELF is the standard executable file format for UNIX systems. ELF files contain a number of sections. What is the purpose of the following three sections? Provide a 1-2 sentence description for each.

- **`.text`**
- **`.data`**
- **`.rodata`**

Problem 10

Write a bash command to print out a text representation of all of the **`pop`** assembly instructions from an ELF file named **`runme`**.

Problem 11

In this problem, you should write a short bash script. The script should list all files that end in **.o** from the **/lib** directory on a UNIX system, only if the file name (not including the extension) begins and ends with a vowel.

Problem 12

Pretend that we have a simple game program called statistics whose source code has the following structure and characteristics:

```
- statistics
   |- statistics.c
   |- coremath.c
   |- coremath.h
   |- calc.c
   |- calc.h
   +- makefile
```

Expect that statistics.c contains the main function. coremath.c contains a collection of function implementations whose prototypes are in coremath.h and calc.c contains a collection of function implementations whose prototypes are in calc.h. statistics.c only includes calc.h and a few standard library header files. calc.c includes only coremath.h. coremath.h does not contain any other includes. When built, the resulting game executable should be named statistics. Write what should go in the makefile that has one rule per output file produced.

Problem 13

In this problem, you should write a bash script to complete the following task.

The bash script should have one command line argument, that being a path to some directory on the computer. The script should find every file that ends in .c or .h within that directory, and count how many lines are in each. It should print out the name of each file with the number of lines in the files.

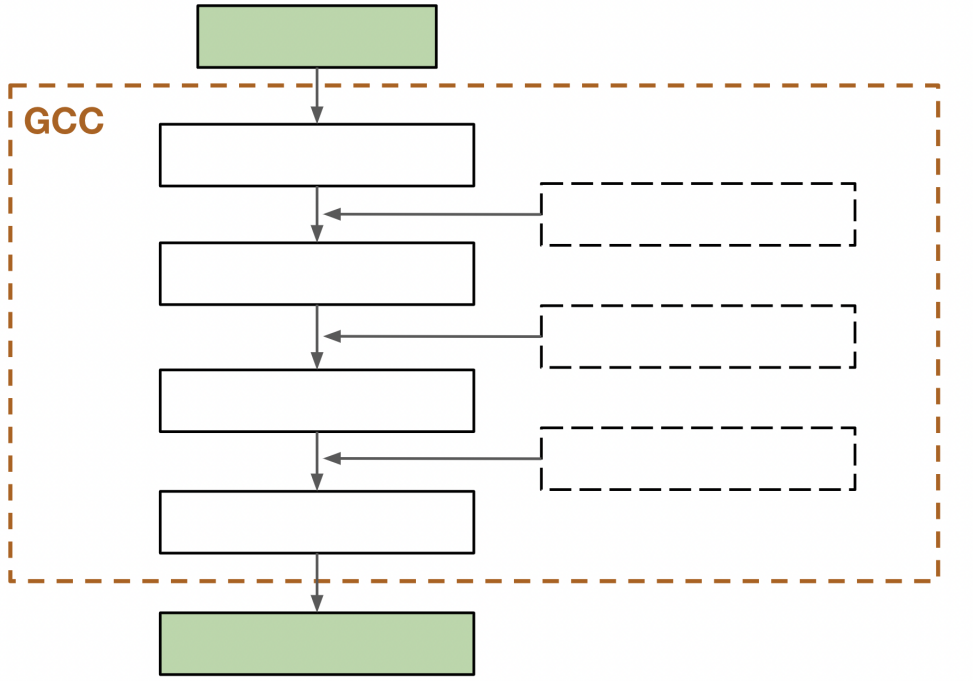
Problem 14

In this problem, you should write a bash script to complete the following task.

The bash script should have one line argument, which you can expect will be the name of a text file. The script should determine how many lines of the file begin with the letter **`a, b, c . . ., y, z`**. For each, it should print out the starting letter, accompanied by how many lines start with that.

Problem 15

Fill in the blanks of the diagram showing the steps of compilation with GCC:



Problem 16

When we run a program with valgrind, it can sometimes display this type of problem

```
definitely lost: X bytes in Y blocks
```

and it can also display this type of problem:

```
Conditional jump or move depends on uninitialised value(s)
```

What do each of these mean? What is the difference between the two? Give an example of each.

Problem 17

Match the name of the command to the best-fitting description:

sed	displays the current user
ls	show running processes
elfdump	search for and optionally replace elements in text
objdump	show processes and what resources those processes are consuming
whoami	search through text
grep	display files
du	show information about ELF executables, including disassembly
ps	show information about the sections in an ELF file
top	display sizes of files

Problem 18

What is the difference between the **fork** and **exec** system calls? Why are these two system calls often used together? Give a clear example.

Problem 19

Recall that behind-the-scenes on a UNIX system, a file is represented by an i-Node. For this problem, assume the following:

- A data block is 160 bytes.
- A data block pointer is 8 bytes.
- An i-Node will generally contain some amount of meta-data at the beginning, such as file permissions, last modified time, etc.
- The i-Node will have 5 direct pointers to data blocks.
- The i-Node will have 1 pointer to a data block that contains pointers to data blocks. (1 level of indirection)
- The i-Node will have 1 pointer to a data block that contains pointers to other data blocks, with pointers to data blocks. (2 levels of indirection)

Say we want to save an ASCII text file that contains 3,000 letters. How many total data block pointers will need to be used to store this? Show your work.

Problem 20

How many bytes of heap memory does the below program allocate in total? Show your work.

```
#include <stdint.h>
#include <stdlib.h>

int main() {
  uint8_t num = 0;
  uint8_t num2 = 1;
  while (num < num2) {
    num = num+1;
    num2 = num2+1;
  }
  for (int i = 0; i < num; i++) {
    char * x = malloc(sizeof(int32_t));
  }
  while (num > 0) {
    num = num / 2;
    char * z = calloc(2, sizeof(int16_t));
  }
  return 0;
}
```

Problem 21

If we have a struct named Student as defined below, what is the minimum number of bytes that would be allocated when calling malloc(sizeof(Student)); on lectura.

```
typedef struct Student {
    char* name;
    uint8_t id;
    float gpa;
} Student;
```

Problem 22

Given the following main function, write out the function that thing points to.

```
int main(void) {
    int (*thing)(int) = &doubler;
    printf("%d\n", (*thing)(1));
    printf("%d\n", (*thing)(5));
    printf("%d\n", (*thing)(10));
    return 0;
}
```

This outputs:
```
2
10
20
```

Problem 23

Assume you're given a program named testprog which accepts input from standard in. You can also assume that there's a text file named input.txt which contains a sample set of inputs.

What is the command to run testprog, feeding it the inputs from input.txt on stdin, and checking for memory leaks all in one command?

Problem 24

Write a function that has a char * as an argument which represents the path to a file on the system and uses a system call to print out the modification time of that file. You will likely want to include time.h and sys/stat.h to do this.

Problem 25

Assume we want to anonymize some data in a csv file. Write a command using sed which replaces all occurrences of street addresses in a file with the string ####.

Street addresses will always take the form: "1234 E Streetname"
Where the the number can be any length
The direction will always be a single uppercase letter from {N, S, E, W}
And the street name will be a single word without spaces of unknown length.

Problem 26

There are at least two problems with the following piece of code. Identify the problems and provide a fix for each problem.

```c
int main(void) {
    char *name;
    for (int i = 0; i < 10; i++) {
        name = calloc(1000, sizeof(char));
        printf("Enter your name: ");
        scanf("%s", &name);
        printf("Your name is: %s\n", name);
    }
    free(name);
}
```

Problem 27

True or False?

The C language automatically initializes variables to safe values.
Valgrind will show you more detailed error locations when you compile a library with -g
Everytime you run your program it will have the same process ID
When you call fork() the new process is a child of the caller.

Created by: Benjamin Dicken
with additional problems by Michael Stark