CSC 352 : Study Guide for Exam 1

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 covered:

- General UNIX, Linux, bash knowledge
- How to use various bash commands (ls, cd, pwd, cut, grep, cal, echo, printf, sed, etc)
- Piping and redirection
- Strings, standard in/out/err
- Functions, return values and arguments and out-parameters
- Using command line arguments
- GCC and some of the command line arguments used in class (-Wall, -Werror, etc)
- Arrays - 1D, 2D and pointer math
- C programming
    - This includes everything covered about C in this course. Language basics, types, functions, I/O, pointers, 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, fgets, scanf, etc.
- General UNIX file system knowledge
- UNIX files, inodes, inode structure

Problems

Solutions for these problems are not included in this guide. 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 exam preparation.

Problem 1

Match the UNIX command with the description that best describes the commands purpose.

`cd`	Copy a file from one location to another on the same computer
`cut`	Create a new directory
`ls`	Search for text within a file or standard input
`scp`	Change the current working directory
`pwd`	Create a link to a file
`gcc`	Select columns from a file or standard input
`grep`	Copy files between different computers
`mkdir`	List files
`ln`	Show the current working directory
`cp`	Compile a C program

Problem 2

Answer each of the below questions.

- Difference between Unix and Linux?
- Name the command line arguments for gcc which makes the compiler interpret all warnings as errors.
- The C standard library has a function to flush the buffer of an open file, name it.
- Difference between ssh and scp?
- Define following commands- cd, ls, mkdir, grep, cat, cut, cal, find, sort, head, tail, tr, rev, touch, uniq.
- What is a pointer in C?
- What is a preprocessor?
- Total number of reserved keywords in C?
- What is the size of a pointer?
- How is a pointer represented in C?
- How to get the address of a value in C?
- What is garbage value?
- Define L-values and R-values?
- What is the difference between x++ and ++x?
- What is a MakeFile and its purpose?
- Define Command line arguments.
- What is the role of (break, run, next/step, bt, frame, print) key option for gdb?
- Make sure you understand and are confident about PA1 part A (bash commands)

Problem 3

How do you *print()* x and y in the program given below in C?

```
#include <stdio.h>
double x = 1.23;
float y = 1.23;
```

Problem 4

Give the expected output of the following program:

```
#include <stdio.h>
void main() {
  int p1 =100, p2 =200, p3, p4;
  p3=p1++;
  p1 =++p2;
  p4=pl+p2--;
  printf ("%d %d %d %d", p1,p2,p3,p4);
  return 0;
}
```

Problem 5

Give the output of the following code Snippet.

```
#include<stdio.h>
void main() {
  int * ptr;
  int x =500;
  ptr = &x;
  printf("%d %d %d %d %d", x, ptr, &x, &ptr, *(&ptr));
  return 0;
}
```

Problem 6

Assuming we have:

```
int a[] = {52, 1, 22, 80, 4, 35, 90, 44};
int *p=a, *q=&a[6];
```

1) What is the value of *(p+5)?
2) What is the value of *(q-3)?
3) Is the condition *q < *p true?
4) What are the values of the array *a* after executing the following statements?

```
      *p = *q – 1;
      p += 1;
      *p = *(q - 1);
      p = p + 2;
      printf("%d ",(*p)++);
```

Problem 7

The following code doesn't work correctly. Why you can't use == operator to compare two string variables in C? What happens if you do?

```
char name1[] = "John";
char name2[] = "John";
if (name1==name2)
  printf("Same!");
```

Problem 8

The code below is supposed to make a copy of a string where all lowercase letters are converted to uppercase, but it has a memory problem. Explain what the problem is and provide the fixed code.

```
char* toUpper(char* s) {
  char buf[100];
  for (int i=0 ; s[i]!='\0' ; i++)
    if (s[i]>='a' && s[i]<='z')
      buf[i] = s[i] - 32;
    else
      buf[i] = s[i];
  return buf;
}

int main() {
  char s[] ="this is an example text";
  char* upper = toUpper(s);
  printf("%s\n", upper);
  return 0;
}
```

Problem 9

The following piece of code is supposed to copy string *str* to *new_str*, but it doesn't work correctly. If you run it, the printf will cause a memory error. What is wrong? Provide the fixed code.

```
char str[10], new_str[10];
for (int i = 0; str[i] != '\0'; i++)
  new_str[i] = str[i];
printf("The new string = %s\n", new_str);
```

Problem 10

Write a function called **`is_palindrome.`** This function should have a single parameter, a char array (or char pointer) and should return an **`int`**. It should determine if the char array is spelled the same forward or backward. The function should return nonzero if the string is a palindrome or zero if it is not a palindrome. You can use the **`strlen(char*)`** function from the C standard library, which returns the length of the char array.

Problem 11

Write a function called **`switch_case`**. This function will have a single parameter, a char array (or char pointer) and should not return anything (**`void`**). You can expect the string to contain lowercase, uppercase and special characters. The function should switch the case of all uppercase characters to lowercase and all lowercase characters to uppercase.

Problem 12

Write a function called **`longest_substring`**. This function will have a single parameter, a char array (or char pointer) and should return an **`int`**. You can expect the string to contain only uppercase and lowercase characters. The function should determine the length of the longest substring of repeated characters found in the string (case sensitive). The function should return the length of the longest substring. If there are no repeated characters in the string, it should return **`1.`**

Problem 13

Write a function called **`odds_and_evens.`** This function shouldn't return anything and will have four parameters. The first three are pointers to integer arrays and the last is an integer. The first two are are pointers to integer arrays (**`int*`**) to hold the even and odd values. The last parameter is an integer representing the length of the third parameter. The function should copy every integer from the third parameter into the respective array that holds the even and odd values. You should terminate the first two integer arrays with (**`-1`**) as well.

Problem 14

Write a C function named **`RightMove`** which will have `int arr[], int len and int k` as its parameter which moves the first **length - k** elements in array arr k places to the right and wraps the last `k` elements in array `arr` around to the left end of the array.

Explanation: Assume that 0 <= k < length and that array arr has length elements.
For example, if arr[] = {11, 22, 33, 44, 55, 66, 77, 88},
the call `RightMove(arr, 8, 3)` changes arr to {66, 77, 88, 11, 22, 33, 44, 55}.

Problem 15

Write a C function named **`append`**, with prototype **`void append (char *s1, const char *s2)`**, that places string s2 at the end of string s1. For example, if s1 contains *"Tucson"* and s2 contains *"Arizona",* the function updates s1 to *"TucsonArizona"*. You are not allowed to use the library function strcat from string.h in your solution.

Problem 16

Write a function called **`toInt`** with the prototype **`int toInt(char* s)`** that gets a string (e.g "1234") and returns its corresponding integer (1234). You can assume that the given integer number is small enough to fit in the standard *int* type. Also, you can assume that the integer number is not negative.
Note: there are functions in the C standard library that convert a string to an int. You may not use them.

Problem 17

Write a function called **`toString`** with the prototype **`void toString(int num, char* s)`** that gets an integer (e.g 1234) and returns its corresponding string ("1234") via the out-parameter *s*. You can assume that the integer number is not negative.
Note: there are functions in the C standard library that convert an int to a string. You may not use them.

Problem 18

Write a program called *myProg* that gets two options *-i* and *-o* via command-line arguments. The options can be provided in any order. So both of the following are acceptable:

```
./myProg -i -o
./myProg -o -i
```

The program must check the following conditions. If they are not met, it should print an appropriate message to standard error and exit with exit code 1:

- At most, two command-line arguments are provided.
- One of the command-line arguments is "*-i*" and the other one is "*-o*".
- The name of the executable is "*myProg*".