CMSC216 Final Exam Review

Unix

1. Write a command to change the permissions of "data.txt" so that the user has read, write, and execute permissions, and everybody else only has read permissions.

   chmod 744 data.txt

2. Write a Unix command to copy all of the files ending with .txt from the ~/216 directory into the current directory.

   cp ~/216/*.txt ./

3. Write a Unix command that copies the ~/216public/lecture_examples directory into the parent directory of the current directory.

   cp -r ~/216public/lecture_examples ../

4. Write a Unix command using **grep** to list the lines of all the .c files in the current directory that contain the word **TODO**. Use the -i flag for case insensitivity.

   grep -i TODO *.c

Pointers

Write what the output would be/what would happen in each of the following code segments. Assume all code is in **main**.

1.
```
char str[20] = "Finals!";
char *ptr = str;

if (sizeof(ptr) == sizeof(str)) {
    printf("Same\n");
}
```

True or False: Changing the second to line **char ptr[20] = str;** would compile and make the program print "Same". False: cannot assign one array to another, compilation error

True or False: Changing the first line to **char *str = "Finals!";** would compile and make the program print "Same". False: str and ptr are string literals, so they should be const char * str, *ptr for initialization and assignment to compile.

2.
```
int num_one, num_two, num_three;
int *ptr_one = &num_one, *ptr_two, *ptr_three;

/* Line */
```

What would happen if we replaced the comment with printf("%d", *ptr_one)?

Prints garbage value since num_one was never initialized to any value;

What would happen if we replaced the comment with
printf("%d", num_three)?

Prints garbage value since num_three was never initialized to any value;

What would happen if we replaced the comment with
printf("%d", *ptr_two);

Runtime error could occur because ptr_two was never initialized so we may not be able to derefrence any value from it because it was never referencing a value in memory to begin with.

What would happen if we replaced the comment with
*ptr_three = 3;
printf("%d", *ptr_three);

Runtime error could occur because ptr_three was never initialized so we may not be able to derefrence any value from it because it was never referencing a value in memory to begin with.

```
3.    const int num_one = 1;        /* Line 1 */
      int const num_two = 1;        /* Line 2 */

      int a, b;

      int *const p1 = &a;           /* Line 6 */
      const int *p2 = &a;           /* Line 7 */
      const int *const p3 = &a;     /* Line 8 */
```

What, if any, are the differences between the declarations on Line 1 and Line 2?

No differences.

What, if any, are the differences between the declarations on Lines 6, 7, and 8?

6: const pointer so it cannot reference a new memory
7: const integer so cannot be assigned new value
8: const int and pointer so cannot reference new memory or assigned new value

```
4.    int x = 100;
      int *ptr;
      void *v_ptr;
```

True or **False**: The assignment v_ptr = &x; is allowed. True

True or **False**: The assignment v_ptr = ptr; is allowed. True

Why can't we dereference v_ptr directly?

Its void type, so compiler does not know the size of the value/number of bytes to retrieve when dereferencing memory

Write a code segment that would use use v_ptr to print x.

v_ptr = &x;
printf("%d", *((int*)v_ptr));

Memory Map

Suppose we have the following code:

```
      int *foo(int **a, int b, int *c) {
```

```
    int new_var = 10000;

    **a = 3;
    *a = NULL;

    b = 3;

    /* Draw memory map at this point */

    /* Point A – assume we return something here */
  }


  int main(void) {
       int *x, y, *z, *ret;
        int a[4] = {1, 1, 1};

        y = 10; x = &y;

        ret = foo(&x, y, z);                                    }
```

Draw a memory map at the indicated point of the program.

At Point A, what would happen if we returned &new_var?

Compiles but then theres a probem because we are returning adress of local variable in foo method so adter foo is removed from stack (after execution), reference to new_var is lost.

Strings/Command Line Arguments

1. Implement the C library function **strlen** to calculate the length of an input string. Assume it has a null byte.

```
int strlen(char *str) {

    int len = 0;
    while(*str != '\0'){
       len++;
       str++;
    }

    return len;


}
```

2. Implement a program that uses command line arguments to reads three inputs. The first input will be a string that is either "sum" or "difference". The second and third inputs will be integers. Assume

that if the proper number of arguments are present, the input will be valid. Here's some example runs of what your code should do:

a.out sum 1 5
6

a.out difference 2 7
-5

a.out invalid number of arguments
Invalid input

```
int main( int argc, char *argv[]          ) {
   if(argc != 4){
      printf("Invalid input");
   } else{
      int num1, num2;
      num1 = atoi(argv[2]);
      num2 = atoi(argv[3]);
      if(strcmp(argv[1], "sum") == 0){
         int sum = num1+num2;
         printf("%d", sum);
      } else if(strcmp(argv[1], "difference") == 0){
         int diff = num1-num2;
         printf("%d", diff);
      }
   }
}
   return 0;
```

Structures and Arrays

1. True or False: You can assign structures to each other.
2. True or False: You can assign arrays to each other.
3. True or False: You can compare structures with '=='
   If you answered true, what does the comparison compare?

4. True or False: You can compare arrays with '=='
   If you answered true, what does the comparison compare?

executes pointer comparison between &a[0] and &b[0]

IO

1. Name one key difference between Unix I/O and Standard I/O.

stdio uses buffers whereas unix does not

Write a program **void count_money(char *file_name)** that counts how much money was spent at one of two stores: "storeA" and "storeB". Assume the file with name **file_name** has lines that are transactions of the form "store_name, money_spent". Your function should count how much money was spent at both storeA and storeB and print out each of the counts at the end. Assume lines are 80 characters max and are valid.

Example file:
storeA, 20

```
storeB , 35
storeA, 10

Output:
storeA: 30
storeB: 35
```

```
void count_money(char *file_name){
   FILE *file;
   file = fopen(file, "r");
   if(file != NULL){
      char line[80 + 1];
      int total_money_spent[2] = {0,0};
      while(fgets(line, 80+1, file) != EOF){
         char store_name[80 + 1];
         int money_spent;
         sscanf(line, "%s , %d", store_name, &money_spent);
         if(strcmp(store_name, "storeA") == 0){
            total_money_spent[0] += money_spent;
         } else if(strcmp(store_name, "storeB") == 0){
            total_money_spent[1] += money_spent;
         }
      }

      printf("storeA: %d\n", total_money_spent[0]);
      printf("storeB: %d\n", total_money_spent[1]);
      fclose(file);
   }
}
```

53/2 = 26 r1
26/2 = 13 r0
13/2 = 6 r1
6/2 = 3 r0
3/2 = 1 r1
1/2 = 0 r1

Data Representation and Bitwise Operators

1. Convert the following decimal numbers to binary. Assume these are all 8 bit signed integers:

(i) 53 (ii) 32 (iii) 28

i) 53 => 110101 = 00110101
ii) 32 => 100000 = 00100000
iii) 28 => 11100 = 00011100

2. Find the negative representation of each of the numbers in #1. Use two's complement.

i) -53 => 11001010 + 1 = 11001011
ii) -32 => 11011111 + 1 = 11100000
iii) -28 => 11100011 + 1 = 11100100

3. Write the output for the following program.

```
unsigned char num1 = 0x4d, num2 = 0xd6;

printf("%02x\n", num1 & num2);
printf("%02x\n", num1 | num2);
printf("%02x\n", num1 ^ num2);
printf("%02x\n", num1 << 2);
printf("%02x\n", num2 >> 1);
```

num1 = 0100 1101
num2 = 1101 0110

&: 0100 0100 => 0x44
|: 1101 1111 => 0xdf
^: 1001 1011 => 0x9b
n1<<2: 0011 0100 => 0x34
n2>>1: 0110 1011 => 0x6b

4. Write a program **count_ones(unsigned int num)** that prints out the number of "1"s in the binary representation of **num.**

```
void count_ones(unsigned int num){
   int count = 0;
   while(num != 0){
      count += num % 2;
      num /= 2;
   }
   printf("%d\n", count);
}
```

5. A group of 4 bits is known as a "nibble". This means that one byte consists of two nibbles. Write a function **flip(unsigned char num)** flips the nibbles of **num**. For example, 11010000 would turn into 00001101, and 0x53 would turn into 0x35.

```
unsigned char flip(unsigned char num){
   return num>>4 | num<<4;
}
```

Dynamic Memory Allocation

1. Where is dynamically allocated memory stored?

heap

2. Why do calloc() and malloc() return void pointers?

can be assigned to any type

3. What is the difference between malloc() and calloc()?

calloc initialize all of the memory to 0/null & calloc takes 2 parameters whereas malloc only has 1 parameter

4. Examine the following code segment:

```
int *a, *b, *c;

a = malloc(sizeof(int*));

*a = 100;
b = a;
c = b;

free(b);
printf("%d", *c);
```

Identify any of errors/bad practices in the above program and fix them.

a = malloc(sizeof(int)); [NOT sizeof(int*)] & abc become dangling pointers after free(b) since all of them pointerd to same memory in heap which was just freed

After we call free(b) above, how many dangling pointers are there in main?

3, abc all are dangling pointers

Should we call free(a) and free(c) after calling free(b) to avoid a memory leak?

no because free(b) already frees a and c because they all point to same memory in heap

5. Look at the following code segment:

```
int main(void) {
      int x = 5;
      int *y = &x;
```

```
      y = calloc(10, sizeof(int));
      y = NULL;
      free(y);
}
```

Are there any errors in the above code? If so, fix them.

free(y) then y=NULL, otherwise memory in heap is being leaked with out reference to it and y is being unnecisarily freed

Makefiles and Preprocessor

Write a Makefile to create an executable called **study**, which relies on the review.c and student.c files. The review.c file includes review.h, and student.c includes student.h. Create a Makefile that contains a target for the **study** executable.
Use -Wall and -g as your CFLAGS.

```
CC = gcc
CFLAGS = -Wall -g

study: review.o student.o
   $(CC) -o study review.o student.o

review.o: review.c review.h
   $(CC) $(CFLAGS) -c review.c

student.o: student.c student.h
   $(CC) $(CFLAGS) -c student.c
```

Suppose you have a file student.h defined below:

```
typedef struct {
      char *name;
      int id;
} Student;
```

We are worried because we have a file that includes two other .h files that both include student.h. What can we change about student.h to prevent the Student struct from being defined twice?

add to student.h

```
#if !defined(STUDENT_H)
#define STUDENT_H
...
#endif
```

Linked Lists

Suppose we have a Linked List of numbers that is specified by the following structs. Each node has a dynamically allocated integer.

```
typedef struct node {
```

```
        struct node *next;
        int *num;

} Node;

Typedef struct list {
      Node *head;

} Linked_list;
```

Write a function **remove_evens(Linked_list *list)** that will remove nodes with even numbers from the Linked List. Try to implement this twice: once recursively and once iteratively.

iteratively

```
-------------
void remove_evens(Linked_list *list){
   Node *curr = list->head, prev = NULL;
   while(curr != NULL){
      if(curr->num%2 == 0){
         if(prev == NULL){
            list->head = list->head->next;
            free(curr->num);
            free(curr);
            curr = list->head;
         } else{
            Node *dummy = curr;
            curr = curr->next;
            prev->next = curr;
            free(dummy->num);
            free(dummy);
         }
      }
   }
}
```

recursviely

```
--------------
void remove_evens(Linked_list *list){
   list->head = remove_evens_helper(list->head);
}

Node *remove_even_helper(Node *node){
   if(node == NULL){
      return NULL;
   }

   if(node->num%2 == 0){
      Node *next = node->next;
      free(node->num);
      free(node);
      node->next = remove_evens_helper(node->next);
      return node;
   }
}
```

Processes/Assembly: For coding questions, look at old exams and your projects to study. For process questions in particular, review all of the 5 process worksheets you were given to study (detailed solutions are now on Piazza!)