CS23200 – Introduction to C and Unix – Fall 2020
2nd Exam Solution

Name: ______________________

Instructions

- This is a closed-book, closed-notes exam. But you can have two sheets of your own notes. Moreover, you can bring the handout "Reference Material from K&R".
- This exam consists of 8 questions and 10 pages.
- Please read through all questions carefully before working on a problem.
- You have 90 minutes in total, and please budget your time wisely.
- Please write your answers clearly and make sure that your writing is readable.
- Please ask if you have any questions.

Question	Score
1 (10 points)	
2 (10 points)	
3 (10 points)	
4 (10 points)	
5 (10 points)	
6 (15 points)	
7 (15 points)	
8 (20 points)	
Total (100)	

1. (10 points) **Structure**.
    1) (5 points) Define a structure that has two integer members.

```
struct myStruct {
        int x ;
        int y ;
};
```

    2) (5 points) Declare an instance of your structure and initialize its members to -1.

```
struct myStruct s;
s.x  =  -1;
s.y  =  -1;
```

2. (10 points) **Memory Management**.
   1) (5 points) After the following declarations, allocate space on the heap for **z** floats and set **data** to point to that memory.

```
const int z = 3000;
float *data = NULL;
```

```
data = malloc(sizeof(float) * z);
```

   2) (5 points) Write a line of code to deallocate the memory you allocated in the previous part.

```
free(data);
```

3. (10 points) **Memory Management**. What is wrong with the following code? Please identify all bugs and fix them.

```
main(){
    int n = 250;
    int array = malloc(sizeof(int) * n);
    int i;

    for (i =0; i < n; i++) {
        array[i] = 0;
        free(array[i]);
    }
    for (i = 0; i < n; i++) {
        array[i] = i;
    }
}
```

Correct code:

```
main(){
  int n = 250;
  int *array = malloc(sizeof(int) * n);
  int i;

  for(i =0; i < n; i++){
     array[i] = 0;
   }
  for(i = 0; i < n; i++){
    array[i] = i;
  }
  free(array);
}
```

4. (10 points) **Pointers**. For each of the following parts, determine if the code is valid. If not valid, indicate why. If valid, answer the questions about the values.

   1)
      ```
      char * buffer;
      *buffer = (char) getchar();
      ```

      What is the value of buffer[0] when the input is “hello”?

      Invalid. Not initialized.

   2)
      ```
      char buffer[10];
      *buffer = (char) getchar();
      ```

      What is the value of buffer[0] when the input is “goodbye”?

      buffer[0] = ‘g’

   3)
      ```
      char * buffer = "CS232";
      *buffer = (char) getchar();
      ```

      What is the value of buffer[0] when the input is “test”?

      Invalid. buffer points to read-only memory.

5. (10 points) **Multiple Files and Makefile**. Write a Makefile to compile the following files to create an executable program:
   1) file1.o depends upon file1.c, file1.h
   2) file2.o depends upon file2.c, file2.h
   3) file3.o depends upon file3.c, file3.h
   4) mainProg depends upon mainProg.c, file1.o, file2.o, and file3.o

   Please make sure that your Makefile includes "all" and "clean" targets.

```
CC = gcc
CFLAGS = -g -Wall
OBJS = file1.o file2.o file3.o

all: mainProg

mainProg: mainProg.c $(OBJS)
    $(CC) $(CFLAGS) $(OBJS) $< -o $@

%.o : %.c %.h
    $(CC) $(CFLAGS) -c $< -o $@

clean:
    rm *.o mainProg
```

6. (15 points) **Command-Line Arguments**. Write a main() method that takes three integers as command-line arguments, calculates the sum, and prints out the result. Please write the code to match the following sample runs:

```
$ ./p4
USAGE: ./p4 int1 int2 int3
$ ./p4 1 2
USAGE: ./p4 int1 int2 int3
$ ./p4 1 2 3 4
USAGE: ./p4 int1 int2 int3
$ ./p4 1 2 3
The sum is 6
```

```
#include <stdio.h>
#define NUM 3

int main(int argc, char * argv[]) {
    int sum = 0, i;

    if (argc != NUM + 1) {
        printf ("USAGE: %s int1 int2 int3\n", argv[0]);
        return -1;
    }

    for (i = 0; i < NUM; i++) {
        sum += atoi(argv[i+1]);
    }

    printf("The sum is %d\n", sum);
}
```

7. (15 points) **File I/O and Standard Library**. The following function takes a file name and an integer as arguments. The file contains a bunch of whitespace-delimited integers. Complete the body of the function so that it returns the number of times **target** occurs in the file. For instance, if target is 12 and the file contains

```
34  7  8  129
12  2
12  -8  12   4
```

then the function should return 3. Please fill in the blanks (5 places).

```
int countOccurrences (const char* filename, const int target) {
   FILE * fp = fopen(filename, "r");
   if (fp == NULL) {
      printf("Error: cannot open file %s\n", filename);
      return -1;
   }

   const int len = 1000;
   char line[len];
   int count = 0, num, charRead;

   while(fgets(line, len, fp) != NULL) {
      char * pChar = line;

      while (sscanf(pChar, "%d%n", &num, &charRead) == 1) {
         if (num == target)
            count++;

         pChar = pChar + charRead;
      }
   }

   fclose(fp);

   return count;
}
```

8. (20 points) **Memory Management, Pointers, File I/O, and Standard Library**. The program reads a list of strings from a data file and stores it in a **binary tree**, and then prints out all strings in alphabetic order. For example, if a data file "data_p8.txt" contains:

```
this
is
cs232
university
pfw
```

When running the program, it shows

```
$ ./p8 data_p8.txt
cs232
is
pfw
this
university
```

1) (14 points) Fill in the blanks in the **main** and **insertValue** functions.

```
/* header files define here */
……
#define MAX_LEN 1000

struct node {
   char * data;
   struct node *left;
   struct node *right;
};

void insertValue(char * value, struct node *root);
int createNode(char * value, struct node ** newNode);
void printTree(struct node *root);
void destroyTree(struct node *root);

int main(int argc, char * argv[]){

   if (____argc != 2______________)
      return -1;

   FILE * fp = __fopen(argv[1], "r")___________;
   if (fp == NULL)
      return -1;
```

```
    char line[MAX_LEN];

    if (____fgets(line, MAX_LEN, fp) == NULL_____)
        return -1;

    struct node *root;
    if(createNode(line, &root) != 0)
        return -1;

    while (__fgets(line, MAX_LEN, fp) != NULL___)
        insertValue(line, root);

    printTree(root);
    destroyTree(root);
    fclose(fp);
    return 0;
}

void insertValue(char * value, struct node *root){

    if( ___strcmp(value, root->data) > 0___) {
        if(root->right == NULL) {
            struct node *newNode;
            if(createNode(value, &newNode) != 0)
                return;

            root->right = newNode__________;
        }
        else
            insertValue(value, root->right);
    }
    else {
        if(root->left == NULL){
            struct node *newNode;
            if(createNode(value, &newNode) != 0)
                return;

             root->left = newNode_____________;
        }
        else
            insertValue(value, root->left);
    }
}
```

2) (6 points) Implement **createNode** function.

```
int createNode(char * value, struct node ** newNode) {

        *newNode = malloc(sizeof(struct node));

        if(*newNode == NULL){
                return -1;
        }

        int len = strlen(value);
        (*newNode)->data = malloc(sizeof(char) * (len+1));

        if ((*newNode)->data == NULL) {
                free(*newNode);
                return -1;
        }

        strncpy((*newNode)->data, value, len+1);

        (*newNode)->left = NULL;
        (*newNode)->right = NULL;

        return 0;



}
```