Name : _____ Student No. : _____

Section	Points
Section 1	/30
Section 2	/25
Section 3	/53
Total	/100

Section 1: multiple-choice questions (30 points)

Question 1 (3 points) Given the following lines of code, what addresses will the pointers be pointing to?

```
int* ptr1= 0x7fff98ef9170;
void* ptr2 = 0x7fff98ef9170;
ptr1 += 1;
ptr2 += 1;
```

 A. ptr1: 0x7fff98ef9174, ptr2: 0x7fff98ef9174   B. ptr1: 0x7fff98ef9171 ptr2: 0x7fff98ef9171
 C. ptr1: 0x7fff98ef9171 ptr2: 0x7fff98ef9174   **D. ptr1: 0x7fff98ef9174 ptr2: 0x7fff98ef9171**

Adding 1 to an integer pointer will add 1 * sizeof(int). Adding 1 to a void pointer will literally add 1 to it.

Question 2 (3 points) A struct can contain an instance of itself.

A. True   **B. False**

A struct can however contain a pointer to an instance of itself.

Question 3 (3 points) Consider the following files. What would be outputted if we used the "diff" command on the two of them?

WIDTH=1920	WIDTH=1920
HEIGHT=1080	HEIGHT=1080
MONITORS=2	MONITORS=3

A. No output.

B.
3c3
< MONITORS=2
—
> MONITORS=3

C.
< MONITORS=2
—
> MONITORS=3

D.
WIDTH=1920
HEIGHT=1080
< MONITORS=2
—
> MONITORS=3

Question 4 (3 points) Given the following lines of code, pick the most correct answer.

i) sizeof(int);
ii) sizeof(int*);
iii) sizeof(int**);

A. i), ii), and iii) would compile successfully and return 4.
B. i), ii), and iii) would compile successfully but return different values.
C. i) and ii) would compile successfully and return 4, but iii) would not compile.
D. i) and ii) would compile successfully but return different values, and iii) would not compile.

Question 5 (3 points) Suppose you have a pointer to a struct. Which of the following are valid ways of accessing a struct's field using the pointer?

A. ptrStruct -> myField   B. (*ptrStruct).myField
C. A and B   D. None of the above.

Question 6 (3 points) What does the following line of code do?

cat notes.txt | head -3 | tail -3

A. Retrieves the first three lines of "notes.txt"
B. Retrieves the last three lines of "notes.txt"
C. Retrieves the first three lines and the last three lines of "notes.txt"
D. Retrieves all the lines except the first three lines and the last three lines of "notes.txt"

Question 7 (3 points) What does calloc() do?

A. It deallocates a chunk of memory.
B. It allocates n chunks of memory and overwrites previous data with a specified value.
C. It allocates n chunks of memory and overwrites previous data with "0"s.
D. It changes the amount of memory allocated to a pointer.

Question 8 (3 points) Which of the following statements are correct?

A. An uninitialized pointer will point to a random location of valid memory.
B. An uninitialized pointer will point to a random location of potentially valid memory.
C. An uninitialized pointer will point to null.
D. The usage of an uninitialized pointer is guaranteed to cause a runtime error.

Question 9 (3 points) What should you not do when documenting your code?

A. Explain how it works.   B. Use self-documenting variable names.
C. Comment everything.   D. Document the purpose of the file.

Don't overcomment! Only commend necessary things, like the tops of functions or important blocks of code.

Question 10 (3 points) Given the following line of code, which of these is not a valid way to get the address of the first element in an array?

int arr[5] = 1,2,3,4,5;

A. arr   B. &arr   C. &arr[0]   **D. None of the above.**

Section 2: program segment questions (25 points)

Question 11 (5 points) The output of the following shell script is:

```
a=0

while [ "$a" -lt 10 ]
do
    b=$a

    if [ "$b" -ge 3 ]; then
        echo -e $b hello >> temp.txt
    else
        echo hello $a > temp.txt
    fi

    a=$(($a+1))

done

# tac reverses order of lines (tac is cat spelt backwards)
cat temp.txt | head -5 | tac
```

Answer:
6 hello
5 hello
4 hello
3 hello
hello 2

Question 12 (5 points) The output of the following C program segment is:

```c
int main(void) {
  int *i, s[4], t[4], *u = s;

  for (i = s; i - s <= 3; i++)
  {
    *i = i - s;
    *(t + *(i)) = 2* *(s + *(i));
  }

  printf("s:t\n");

  for (i = s; i-s <= 3; i++)
    printf("%d:%d\n", *(s + *i), *(t + *i));

  printf("u = %d\n", *u);
}
```

Answer:
s:t
0:0
1:2
2:4
3:6
u = 0

Question 13 (5 points) The output of the following C program segment is:

```c
// Implementation of Queue is not shown. Behaves like a Queue data type.
// queue(int, *Queue): adds an integer to the Queue
// dequeue(*Queue): removes the next integer from the Queue. returns the removed integer.

// Implementation of Stack is not shown. Behaves like a Stack data type.
// push(int, *Stack): adds an integer to the Stack
// pop(*Stack): removes the next integer from the Stack. returns the removed integer.

int main(void) {
    Queue q; // Creates an empty queue of integers
    Stack s; // Creates an empty stack of integers.

    for(int i = 0; i < 10; i++) {
        queue(i*2, &q);
        push(i*2+1, &s);
    }

    for(int i = 0; i < 10; i++) {
        printf("%d:", dequeue(&q));
        printf("%d\n", pop(&p));
    }
}
```

Answer:
0:19
2:17
4:15
6:13
8:11
10:9
12:7
14:5
16:3
18:1

Question 14 (5 points) The output of the following C program segment is:

```c
int myFunc(int a, int *b, int *c, int d) {
    a = *b;
    *b = *c;
    *c = d;
    d = a;
    return (a * *c) + (*b * d);
}

int main(void) {
    int w = -2;
    int x = 5;
    int y = 3;
    int z = 6;

    int f = myFunc(w, &y, &x, z);

    printf("%d, %d, %d, %d, %d", w, x, z, f, y);
}
```

Answer:
-2, 6, 6, 33, 5

Question 15 (5 points) The output of the following C program segment is:

```c
int main(void) {
    int num = 255;

    printf("%d\n", num);
    printf("%i\n", num);
    printf("%o\n", num);
    printf("%x\n", num);
    printf("%X\n", num);

    printf("%5d\n", num);
    printf("%4d\n", num);
    printf("%3d\n", num);
    printf("%2d\n", num);
    printf("%1d\n", num);

    printf("%.5d\n", num);
    printf("%.3d\n", num);
    printf("%.1d\n", num);
}
```

Answer:
255
255
377
ff
FF
  255
 255
255
255
255
00255
255
255

Section 3: coding (53 points)

Question 16 (8 points) Please write a shell script that does the following.
i) Creates an empty file "myCollection.txt" if it doesn't exist.
ii) Give all files in the current directory executable permissions.
iii) Appends lines 3-7 of "main.bash" to "myCollection.txt"
You can assume that the current directory contains only files.
Answer:

```
if ! [ -f "myCollection.txt" ]; then
    touch myCollection.txt
fi

ls | xargs chmod +x

cat main.bash | head -7 | tail -5 | xargs -i echo {} >> "myCollection.txt"
```

This won't produce perfect results (it ignores empty lines), but you'll get full marks if I were the prof

Question 17 (12 points) Write a C program that reads the content of "myCollection.txt" and copies all vowel characters over to "yoeioo.txt".

You can assume that the files already exist. Include y in the vowels.

Answer:

```c
#include <stdio.h>

int main(void) {
    FILE *inputFile = fopen("myCollection.txt", "r");
    FILE *outputFile = fopen("yoeioo.txt", "w");

    char t;

    while(fscanf(inputFile, "%c", &t) == 1) {
        if(t == 'a' || t == 'e' || t == 'i' || t == 'o' || t == 'u' || t == 'y') {
            fprintf(outputFile, "%c", t);
        }
    }

    fclose(inputFile);
    fclose(outputFile);
}
```

Question 18 (15 points) Given the root of a binary tree of integers, write a function that finds the sum of all nodes in the tree.

```
struct BinTree {
    int value;
    struct BinTree *left;
    struct BinTree *right;
};
```

Tree Summing Algorithm. The basic steps to perform Tree Summation are:
1) If the left tree doesn't exist (i.e, is a null pointer), its sum is zero. Move on to step 3.
2) If the left tree exists, we recursively call the summing function on the left tree.
3) If the right tree doesn't exist (i.e, is a null pointer), its sum is zero. Move on to step 5.
4) If the right tree exists, we recursively call the summing function on the right tree.
5) Add up the sum of the left tree, right tree, and the value of the current Node and return this value.
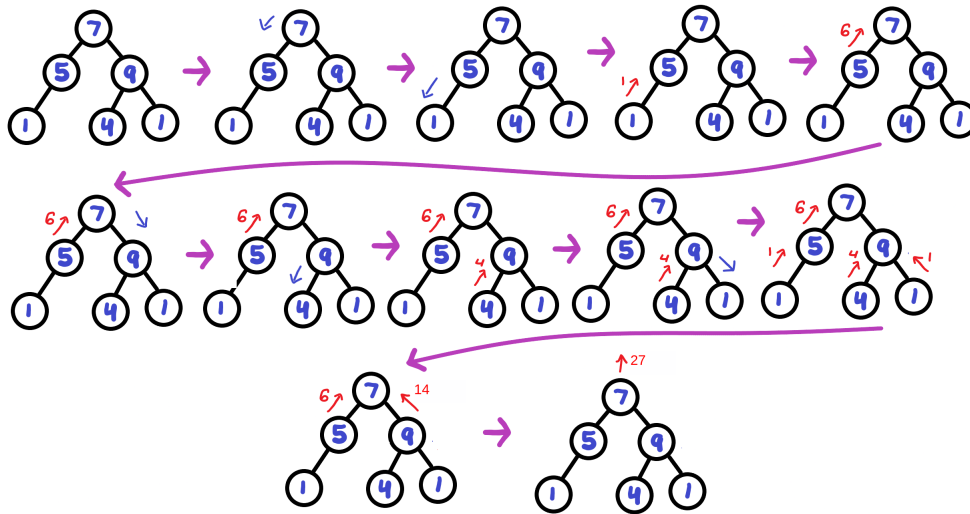
Answer:

```
typedef struct BinTree bintree;
int treeSum(bintree root) {

    int leftSum = 0;
    int rightSum = 0;

    if(root.left != 0) {
        leftSum = treeSum(*root.left);
    }

    if(root.right != 0) {
        rightSum = treeSum(*root.right);
    }

    return root.value + leftSum + rightSum;
}
```

Question 19 (18 points) There are two parts to this question. Part 1 is 10 points, Part 2 is 8 points.

i) Create a struct for a Singly-Linked List named "Student". It should store an integer field named "debt". Suppose you have a pointer "head" that points to the head of a linked list. Create a function "frontInsert" that takes in a pointer to the head pointer and an integer "newDebt". The function should create a new Student with "newDebt" debt and set that to be the new head of the linked list.

ii) Create a function "reverseList" that takes in a pointer to the head pointer. The function should reverse the order of the linked list, so that the end of the linked list becomes the new head.

Answer:

```c
typedef struct Student student;

struct Student {
    int debt;
    student *next;
};

void frontInsert(student **head, int newDebt) {
    student *newStudent = malloc(sizeof(student));
    newStudent -> next = *head;
    newStudent -> debt = newDebt;
    *head = newStudent;
}

void reverse(student **head) {
    student *previous = 0;
    student *current = *head;
    student *next = *head;

    while(next != 0) {
        next = current -> next;
        current -> next = previous;
        previous = current;
        current = next;
    }

    *head = previous;
}
```

Note: A variable declared within a function will **disappear** after the function returns. By using malloc, we allocate the neccessary memory to allow newStudent to persist after the function returns. LMK if I'm just being dumb and I'll update the solution.

Here is a wonderful resource that helps explain the reversing of a linked list.

END OF EXAM