In signing this statement, I certify that I will neither receive aid from nor provide aid to any other student for this exam. I understand that if I fail to honor this agreement, I will receive a failing grade for this course and will be subject to disciplinary action.

Signature:

Printed Name:

login:

This is an open-book, open-notes exam.

No electronic devices are allowed.

There are 7 pages and all 4 questions are mandatory. Question 4 is related to Learning Objective 1 (an ability to read and write C programs that use recursion). You have to score at least 50% to satisfy the learning objective.

Answers should be concise.
1 Types and array (10 points)

In the class, we use a box to visualize each variable/parameter, with the name of the variable/parameter being the label of the box and the type of the variable/parameter written inside the box. If the value associated with a variable/parameter is known, we write down the value in the corresponding box or show the pointer if the value is an address.

For example, we have the following variables/parameters visualized in the figure to the right. For an “array,” it is fine to show only the type for the first entry.

```c
int cnt;
int *iptr;
int iarr[3];
cnt = 3;
iptr = &(iarr[0]);
```

(a) (4 points) Three boxes are shown to the right of the above figure for `iarr`. Provide four different names for the middle box (shaded) such that you can access the value stored in that box in a C program.

(i) (ii) (iii) (iv)

(b) (2 points) One of your classmates visualizes `iarr` as follows:

Do you agree? Justify your answer.

(c) (4 points) Consider another two lines of code that appear after the code fragment shown earlier:

```c
iptr = &iarr[2];
iptr[0] = 10;
```

Draw the boxes to show the variables `iptr` and `iarr`. 
2 Main function, argc, and argv (10 points)

You have successfully compiled a C program into an executable called first. Assume that we type in the following command at a terminal:

```
./first 1 22 333
```

(a) (5 points) Draw the boxes, as in Question 1, to visualize argc and argv. If any of these boxes is storing an address, you should also show the box(es) to which this address points. If the value associated with a box is known, you should write down that value in the box.

(b) (3 points) Each of the boxes you have drawn in (a) should reside in some stack frame in the call stack. Indicate in the diagram you have drawn above the stack frame(s) in which the boxes reside. If you don’t know the function name, you may use, for example, “a stack frame above the stack frame for main,” “a stack frame below the stack frame for main.”

(c) (2 points) What is the value stored in each of the following?

(i) `argv[0][2]`:

(ii) `argv[1][2]`:

(iii) `argv[2][2]`:

(iv) `argv[3][2]`:

If there is insufficient information for you to determine the answer for any of these locations, you may state “insufficient information” and explain.
3 Call stack and stack frames (10 points)

Consider the following program. The statements that correspond to questions (a) and (b) are highlighted with comments.

```c
#include <stdio.h>
#include <stdlib.h>
void swap(int *a, int *b, int **iptrptr)
{
    int temp = (*a);
    (*a) = (*b);
    (*b) = temp;
    (*iptrptr) = &temp; /* store address of temp in *iptrptr */
}

int difference(int a, int b)
{
    int *iptr;
    if (a < b) {
        swap(&a, &b, &iptr); /* Question 3(b)(iii) */
    }
    return (a - b);
}

int main(int argc, char *argv[])
{
    int i, j, *iptr;
    i = 3; j = 4;
    swap(&i, &j, &iptr);
    printf("%d %d %d\n", i, j, (*iptr)); /* Question 3(a) */
    printf("%d\n", (*iptr)); /* Question 3(a) */
    i = 3; j = 4;
    printf("%d %d %d\n", i, j, difference(i, j)); /* Questions 3(b)(i) and 3(b)(ii) */
    return EXIT_SUCCESS;
}
```

/* Questions 3(b)(i) and 3(b)(ii) */
(a) (3 points) The output of the first `printf` statement is

4 3 3

What is the output of the second `printf` statement? If there is insufficient information for you to determine the answer, you may state “insufficient information” and explain.

(b) (7 points) Draw the corresponding call stack

(i) when the third `printf` is called;
(ii) when the `difference` function in the `printf` statement is called; and
(iii) when the `swap` function in the `difference` function is called.
4 Recursion (10 points)

Consider the following incomplete code fragment:

```c
/* Question 4 */
unsigned int Fibonacci(unsigned int n)
{
    /* question 4(a) */

    return (Fibonacci(n - 1) + Fibonacci(n - 2));
}
```

(a) (2 points) Complete the preceding code fragment such that you could compute the following Fibonacci sequence:

\[ F(0) = 1, F(1) = 1, F(2) = 2, F(3) = 3, F(4) = 5, ..., F(n-2), F(n-1), F(n) = F(n-2) + F(n-1) \]

You may assume that the input \( n \) is non-negative, and you do not have to check for that. Note also that the definition of the sequence is slightly different from the one covered in class.

(b) (4 points) Draw a computation tree that shows the recursive calls of the function `Fibonacci` when \( n \) is 5. The first call of the function, which is the root of the computation tree is drawn for you below. You have to show the input parameter of the function `Fibonacci` in each node of the computation tree. For convenience, we write \( F \) instead of `Fibonacci` in the node.
(c) (4 points) For each call of $\text{Fibonacci}(0)$ that occurs in the function call of $\text{Fibonacci}(5)$, show the corresponding call stack when $\text{Fibonacci}(0)$ is called, with $\text{Fibonacci}(5)$ being the bottom-most stack frame. When you draw a call stack, label each stack frame with the function name and the input parameter to that function. The bottom most stack frame for a call stack is shown below. Again, we use $F$ as an abbreviation of $\text{Fibonacci}$. Clearly indicate which call stack is for which function call of $\text{Fibonacci}(0)$ in the computation tree you have drawn in (b).