In signing this statement, I hereby certify that the work on this exam is my own and that I have not copied the work of any other student while completing it. I understand that, if I fail to honor this agreement, I will receive a score of ZERO for this exam and will be subject to possible disciplinary action.

Signature:

You must sign here. Otherwise you will receive a 2-point penalty.

Read the questions carefully.
Some questions have conditions and restrictions.

This is an open-book, open-note exam. You may use any book, notes, or program printouts. No personal electronic device is allowed. You may not borrow books from other students.

Two learning objectives (recursion and structure) are tested in this exam. To pass an objective, you must receive 50% or more points in this exam.
## Contents

1. Recursion 1 Recursive Equation (5 points) 5  
2. Recursion 2: Recursive Function (5 points) 6  
3. Structure 1: Read and Write Data (5 points) 8  
4. Structure 2: Objects and Pointers (5 points) 11

<table>
<thead>
<tr>
<th>Learning Objective 1 (Recursion)</th>
<th>Pass</th>
<th>Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Objective 2 (Structure)</td>
<td>Pass</td>
<td>Fail</td>
</tr>
</tbody>
</table>

Total Score:
This page is blank. You can write answers here if you need more space.
This page is blank. You can write answers here if you need more space.
1 Recursion 1 Recursive Equation (5 points)

There are unlimited red (R), green (G), and blue (B) balls. You need to select \( n \) balls. Two adjacent balls cannot be both red or both green. The orders matter: RB and BR are different. When \( n \) is one, there are three options:

1. R
2. G
3. B

When \( n \) is two, there are seven options. Please notice that RR and GG are invalid options.

1. R G
2. R B
3. G R
4. G B
5. B R
6. B G
7. B B

Q1 Suppose \( r(n) \), \( g(n) \), and \( b(n) \) are the number of options when selecting \( n \) balls and the first ball is R, G, and B respectively. Fill this table (3 points, 0.5 point for each answer).

<table>
<thead>
<tr>
<th>( n )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r(n) )</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( g(n) )</td>
<td>1</td>
<td>2</td>
<td>same as ( r(3) )</td>
<td>same as ( r(4) )</td>
<td>same as ( r(5) )</td>
</tr>
<tr>
<td>( b(n) )</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q2 How many options are there when selecting 5 balls? (2 points)

You can write an expression without writing the final answer. For example, you may write \( 1 + 2 + 9 + 13 \) without writing 25.
2 Recursion 2: Recursive Function (5 points)

Consider the following incorrect implementation of recursive binary search.

```c
// search.c
int binarysearch(int * arr, int key, int low, int high)
{
    if (low > high) {
        return -1;
    }
    int mid = (low + high) / 2;
    if (arr[mid] == key) {
        return mid;
    }
    if (arr[mid] > key) {
        return binarysearch(arr, key, low, mid - 1);
    }
    return binarysearch(arr, key, mid, high); // ERROR
    // ERROR, should be mid + 1 but it is mid
}
```

The main function is shown below.

```c
// main.c
#include <stdio.h>
#include <stdlib.h>
int binarysearch(int * arr, int key, int low, int high);
#define ARRAYSIZE 10
int main(int argc, char * * argv)
{
    int arr[ARRAYSIZE] = {1, 12, 23, 44, 65, 76, 77, 98, 109, 110};
    int ind;
    for (ind = 0; ind < ARRAYSIZE; ind++) {
        printf("%d\n", binarysearch(arr, arr[ind], 0, ARRAYSIZE));
    }
    return EXIT_SUCCESS;
}
```

One line in `binarysearch` is incorrect, as marked by a comment. If `binarysearch` were correct, the expected output would be 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Because of this mistake, the
recursive calls will not end for some cases. For each of these cases, the program eventually runs out of the memory for the callstack.

Q1 Please write down the smallest value of ind in main when the program runs out of the memory for the callstack. (2 points)

Q2 Write down the values of key, low, and high at the top frame when the program runs out the memory for the callstack. (3 points, 1 point for each answer)
3 Structure 1: Read and Write Data (5 points)

Consider the following structure for a two-dimensional array.

```c
#ifndef ARRAY_H
#define ARRAY_H

#pragma pack(1)
typedef struct {
    int length;
    int * data;
} Array;
#endif
```

The following is a program using the structure.

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "array.h"

int main(int argc, char **argv)
{
    int length = 10;
    char * filename = "data";
    // create an object
    Array * arrptr1 = NULL;
    printf("sizeof(arrptr1) = %d\n", (int) sizeof(arrptr1));
    arrptr1 = malloc(sizeof(Array));
    printf("sizeof(arrptr1) = %d, sizeof(Array) = %d\n",
           (int) sizeof(arrptr1), (int) sizeof(Array));
    // allocate memory for the data
    arrptr1 -> length = length;
    arrptr1 -> data = malloc(sizeof(int) * (arrptr1 -> length));
    printf("sizeof(arrptr1) = %d, sizeof(arrptr1 -> data) = %d\n",
           (int) sizeof(arrptr1), (int) sizeof(arrptr1 -> data));
    // initialize the values of the array
    int ind;
    for (ind = 0; ind < (arrptr1 -> length); ind ++)
    {
        arrptr1 -> data[ind] = ind;
    }
    // save the data to a file
    FILE * fptr = fopen(filename, "w");
```
// write the data to the file
if (fwrite(arrptr1, sizeof(Array), 1, fptr) != 1)
{
    // fwrite fail
    return EXIT_FAILURE;
}
printf("ftell(fptr) = %d\n", (int) ftell(fptr));
fclose(fptr);

// fill the array with random numbers
// ensure the heap contains garbage before releasing it
srand(time(NULL)); // set the seed of the random number
for (ind = 0; ind < (arrptr1 -> length); ind++)
{
    arrptr1 -> data[ind] = rand();
}

// release memory
free(arrptr1 -> data);
free(arrptr1);

// read the data from the file
Array * arrptr2 = NULL;
arrptr2 = malloc(sizeof(Array));
fptr = fopen(filename, "r");
if (fread(arrptr2, sizeof(Array), 1, fptr) != 1)
{
    // fread fail
    return EXIT_FAILURE;
}

// add the data
int sum = 0;
for (ind = 0; ind < (arrptr2 -> length); ind++)
{
    sum += arrptr2 -> data[ind];
}
printf("sum = %d\n", sum);
// release memory
free(arrptr2);
return EXIT_SUCCESS;
Assume this program runs on a 64-bit (8 bytes) processor and one integer uses 4 bytes. Assume that the program never returns EXIT_FAILURE.

Q1 What is the output of this program? (3.5 points, 0.5 each answer)
If the program crashes before printing a particular line, please write “Do not reach here”. If a variable is not initialized, please write “garbage”.

sizeof(arrptr1) =

sizeof(arrptr1) = , sizeof(Array) =

sizeof(arrptr1) = , sizeof(arrptr1 -> data) =

ftell(fptr) =

sum =

Q2 Valgrind reports one (or more) memory error. Which line causes this error? (0.5 point) Explain the reason (1 point).

The following information is for your reference.

long ftell(FILE *stream);

The ftell() function obtains the current value of the file position indicator for the stream pointed to by stream.
4 Structure 2: Objects and Pointers (5 points)

Fill in the missing numbers in the program’s output.

swapT1-----------------------------
t: x = 2, y = 6
u: x = -4, y = -3
t: x = , y =
u: x = , y =

swapT2-----------------------------
t: x = 2, y = 6
u: x = -4, y = -3
t: x = , y =
u: x = , y =

swapT3-----------------------------
t: x = 2, y = 6
u: x = -4, y = -3
t: x = , y =
u: x = , y =

swapT4-----------------------------
t: x = 2, y = 6
u: x = -4, y = -3
t: x = , y =
u: x = , y =

Please see the code below and fill in the missing numbers.

1 #include <stdio.h>
2 #include <stdlib.h>
typedef struct
{
    int * x;
    int * y;
} VectorT;

void printT1(const char * label, VectorT a)
{
    printf("%s: x = %2d, y = %2d\n", label, * (a.x), *(a.y));
}

void printT2(const char * label, VectorT * a)
{
    printf("%s: x = %2d, y = %2d\n", label, * (a -> x), *(a -> y));
}

void swapT1(VectorT a, VectorT b)
{
    VectorT tmp = a;
    a = b;
    b = tmp;
}

void swapT2(VectorT *a, VectorT *b)
{
    VectorT tmp = *a;
    *a = *b;
    *b = tmp;
}

void swapT3(VectorT a, VectorT b)
{
    int * tmp = a.x;
    a.x = b.x;
    b.x = tmp;
    tmp = a.y;
    a.y = b.y;
    b.y = tmp;
}
void swapT4(VectorT *a, VectorT *b) {
    int tmp = * (a -> x);
    *(a -> x) = *(b -> x);
    *(b -> x) = tmp;
    tmp = *(a -> y);
    *(a -> y) = *(b -> y);
    *(b -> y) = tmp;
}

int main(int argc, char **argv) {
    printf("swapT1-----------------------------\n");
    VectorT t;
    VectorT u;
    t.x = malloc(sizeof(int));
    t.y = malloc(sizeof(int));
    u.x = malloc(sizeof(int));
    u.y = malloc(sizeof(int));
    *(t.x) = 2;
    *(t.y) = 6;
    *(u.x) = -4;
    *(u.y) = -3;
    printT1("t", t);
    printT1("u", u);
    swapT1(t, u);
    printT1("t", t);
    printT1("u", u);

    printf("swapT2-----------------------------\n");
    *(t.x) = 2;
    *(t.y) = 6;
    *(u.x) = -4;
    *(u.y) = -3;
    printT1("t", t);
    printT1("u", u);
    swapT2(& t, & u);
    printT1("t", t);
    printT1("u", u);

    printf("swapT3-----------------------------\n");
}
* (t.x) = 2;
* (t.y) = 6;
* (u.x) = -4;
* (u.y) = -3;
printT1("t", t);
printT1("u", u);
swapT3(t, u);
printT1("t", t);
printT1("u", u);

printf(" swapT4 ----------------------------\n");
* (t.x) = 2;
* (t.y) = 6;
* (u.x) = -4;
* (u.y) = -3;
printT1("t", t);
printT1("u", u);
swapT4(& t, & u);
printT1("t", t);
printT1("u", u);

// release memory
free (t.x);
free (t.y);
free (u.x);
free (u.y);

return EXIT_SUCCESS;
}