# ECE264 Fall 2017
# Exam 2, 8-930PM, October 18

## Name:

## Purdue ID:

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* **1-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 electronic device is allowed. You may **not** borrow books from other students.

This exam tests two learning objectives:
Recursion (Question 1, 3, and 4)
Structure (Question 2)
You must obtain 50% or more points in each of the corresponding question to pass the learning objective. For recursion, you can pass the objective by question 1, question 3, or question 4.

**Remove the top sheet.**
**Write your answers on page 2.**
**Return only the top sheet.**

# Contents



**Total Score:**

Learning Objective

| | | |
|---|---|---|
| Structure | Pass | Fail |
| Recursion | Pass | Fail |

| Q1 | |
|---|---|
| A | |
| B | |
| C | |
| D | |
| E | |
| F | |
| G | |
| H | |

| Q2 | | Q3 | | Q4 | |
|---|---|---|---|---|---|
| A | | A | | A | |
| B | | B | | B | |
| C | | C | | C | |
| D | | D | | D | |
| E | | E | | E | |
| F | | F | | | |

This page is blank.

# 1 Recursion (32 points)

## 1.1 Recursive Program (24 points)

The combination problem selects $k$ items out of $n$ ($k \leq n$) distinct objects and the orders do not matter. This can be expressed as the $\mathbb{C}(n, k)$ problem. Consider {A, B, C, D ...} as the objects.
$\mathbb{C}(3, 1)$ means selecting one out of {A, B, C} and the result is

```
A
B
C
```

$\mathbb{C}(3, 2)$ means selecting two out of {A, B, C} and the result is

```
A B
A C
B C
```

It is also acceptable if a program ouputs

```
B A
C A
B C
```

since `A B` and `B A` are considered the same.
The following is another acceptable output

```
B A
A C
C B
```

since `A C` and `C A` are considered the same.
Now, suppose the definition of combination is changed. If an item is selected, it may appear **either once or twice**. This new definition is called "new combination" or $\mathbb{NC}(n, k)$,
$\mathbb{NC}(3, 1)$ means selecting one out of {A, B, C}; each selected item may appear once or twice. The result is

```
A A
A
B B
B
C C
C
```

The orders of these lines do not matter. It is also acceptable if `C` appears first, for example,

```
C
C C
B
B B
A
A A
```

$\mathbb{NC}(3, 2)$ means selecting two out of {A, B, C}; each selected item may appear once or twice. The result can be

```
A A B B
A A B
A A C C
A A C
A B B
A B
A C C
A C
B B C C
B B C
B C C
B C
```

Modify the following program as one possible solution to implement "new combination".

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 void printArray(int * arr, int length)
 5 {
 6   int ind;
 7   for (ind = 0; ind < length; ind ++)
 8     {
 9       if (arr[ind] == 1)
10         {
11           printf("%c ", ind + 'A');
12         }
13       if (arr[ind] == 2)
14         {
15           printf("%c %c ", ind + 'A', ind + 'A');
16         }
17     }
18   printf("\n");
19 }
```

```
20 void combineHelp(int * arr, int ind, int num,
21                   int sel)
22 {
23   if (sel == 0) // select enough items
24     {
25       printArray(arr, num);
26       return;
27      }
28   if (ind == num) // end of array, no more item to select
29     {
30       return;
31     }
32
33   // select this element twice
34   // <--- FIX ME --->
35   arr[ind] = ????;          // Question 1.A
36   combineHelp(arr, ????);   // Question 1.B
37   // provide all necessary arguments
38   // make sure there is no syntax error
39
40   // select this element once
41   // <--- FIX ME --->
42   arr[ind] = ????;          // Question 1.C
43   combineHelp(arr, ????);   // Question 1.D
44   // provide all necessary arguments
45   // make sure there is no syntax error
46
47   // do not select this element
48   // <--- FIX ME --->
49   arr[ind] = ????;          // Question 1.E
50   combineHelp(arr, ????);   // Question 1.F
51   // provide all necessary arguments
52   // make sure there is no syntax error
53 }
54
55 void combine(int * arr, int num, int sel)
56 {
57   combineHelp(arr, 0, num, sel);
58 }
59
60 #define ARRAY_SIZE 4
61
```

```
62 int main(int argc, char * argv[])
63 {
64   int arr[ARRAY_SIZE] = {0};
65   int sel = 2;
66   int num = ARRAY_SIZE;
67   combine(arr, num, sel);
68   return EXIT_SUCCESS;
69 }
```

For your reference, the program's output is (the number are added to help you understand).

```
 1 A A B B
 2 A A B
 3 A A C C
 4 A A C
 5 A A D D
 6 A A D
 7 A B B
 8 A B
 9 A C C
10 A C
11 A D D
12 A D
13 B B C C
14 B B C
15 B B D D
16 B B D
17 B C C
18 B C
19 B D D
20 B D
21 C C D D
22 C C D
23 C D D
24 C D
```

## 1.2 Recursive Equation (8 points)

This question asks you to calculate how many lines of outputs $\mathbb{NC}(5,1)$ and $\mathbb{NC}(5,2)$ produce. The following table is provided for your reference.

| $n$ | 1 | 2 | | 3 | | | 4 | | | | 5 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $k$ | 1 | 1 | 2 | 1 | 2 | 3 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 5 |
| $\mathbb{NC}(n,k)$ | 1 | 4 | 4 | 6 | 12 | 8 | 8 | 24 | 32 | 16 | **1.G** | **1.H** | 80 | 80 | 32 |

**Answer:**

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 void printArray(int * arr, int length)
 5 {
 6   int ind;
 7   for (ind = 0; ind < length; ind ++)
 8     {
 9       if (arr[ind] == 1)
10         {
11           printf("%c ", ind + 'A');
12         }
13       if (arr[ind] == 2)
14         {
15           printf("%c %c ", ind + 'A', ind + 'A');
16         }
17     }
18   printf("\n");
19 }
20 void combineHelp(int * arr, int ind, int num,
21                  int sel)
22 {
23   if (sel == 0) // select enough items
24     {
25       printArray(arr, num);
26       return;
27      }
28   if (ind == num) // end of array, no more item to select
29     {
30       return;
31     }
32   // select this element twice
33   arr[ind] = 2;
```

```
34    combineHelp(arr, ind + 1, num, sel - 1);
35    // select this element once
36    arr[ind] = 1;
37    combineHelp(arr, ind + 1, num, sel - 1);
38    // do not select this element
39    arr[ind] = 0;
40    combineHelp(arr, ind + 1, num, sel);
41  }
42  void combine(int * arr, int num, int sel)
43  {
44    combineHelp(arr, 0, num, sel);
45  }
46
47  #define ARRAY_SIZE 6
48
49  int main(int argc, char * argv[])
50  {
51    int arr[ARRAY_SIZE] = {0};
52    for (int num = 1; num <= 5; num ++)
53      {
54        for (int sel = 1; sel <= num; sel ++)
55          {
56            printf("n = %d, k = %d\n", num, sel);
57            combine(arr, num, sel);
58          }
59      }
60    return EXIT_SUCCESS;
61  }
```

10 and 40
The general equation is $\frac{n!}{k!(n-k)!}2^k$

## 2 Structure (24 points)

Consider the following program:

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4 #pragma pack(1) // tell compiler not to pad any space
 5 // assume
 6 // sizeof(char) = 1,     sizeof(int) = 4,
 7 // sizeof(double) = 8, sizeof(a pointer) = 8
 8 #define NAME_LENGTH 64
 9 typedef struct
10 {
11   int ID;
12   char * name;
13 } Student;
14
15 int main(int argc, char ** argv)
16 {
17   Student * sptr;
18   printf("sizeof(sptr)   = %ld\n", sizeof(sptr));   // answer 2.A
19   printf("sizeof(* sptr) = %ld\n", sizeof(* sptr)); // answer 2.B
20   sptr = malloc(sizeof(* sptr) * 2); // two students
21   sptr[0].ID = 264;
22   sptr[0].name = malloc(sizeof(*sptr[0].name) * NAME_LENGTH);
23   strcpy(sptr[0].name, "Purdue");
24   printf("sizeof(sptr[0]) = %ld\n", sizeof(sptr[0])); // answer 2.C
25
26   sptr[1] = sptr[0]; // copy the attributes
27
28   sptr[1].ID = 2017;
29   strcpy(sptr[1].name, "ECE264");
30
31   printf("sptr[0].ID = %d, sptr[0].name = %s\n",
32          sptr[0].ID, sptr[0].name);
33   // answers 2.D and 2.E
34
35   return EXIT_SUCCESS;
36
37   // do not free memory
38   // how many bytes of memory is leaked?
39   // include both direct and indirect leaking
```

```
40    // answer 2.F
41 }
```

Answer the questions marked in the program.
**Answer:**

```
8
12
12
264, ECE 264
88 (12 x 2 + 64)
```

# 3 Recusion (24 points)

The following program implements a method to determine whether a number is stored in an array that has already been sorted by the ascending order. Assume that every element in the array is distinct. If this number is in the array, the function returns the index. If this number is not in the array, the function returns -1 (because -1 is an invalid index).

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int bisearch(int * arr, int value, int low, int high)
5  {
6    if (low >= high) // ERROR, should be low > high
7      {
8        return -1; // cannot find value
9      }
10   int mid = (low + high) / 2; // the index in the middle
11   if (arr[mid] == value)
12     {
13       return mid; // find it
14     }
15   if (arr[mid] > value) // possibly find for index < mid
16     {
17       return bisearch(arr, value, low, mid - 1);
18     }
19   // possibly find for index > mid
20   return bisearch(arr, value, mid + 1, high);
21 }
22
23
24 int search(int * arr, int size, int value)
25 // arr: sorted in the ascending order. elements are distinct
26 // size: the size of the array
27 // value: the value to search
28 // if value is an element in the array, return the index
29 // if value is not an element in the array, return -1
30 {
31   return bisearch(arr, value, 0, size - 1);
32 }
33
34 int main(int argc, char ** argv)
35 {
36   int arr[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

```
37    for (int val = 0; val < 12; val ++)
38      {
39        printf("search(%d) gets %d\n", val, search(arr, 10, val));
40      }
41    return EXIT_SUCCESS;
42  }
```

The correct output of the program (using `if (low > high)`) should be

```
search(0) gets 0
search(1) gets 1
search(2) gets 2
search(3) gets 3
search(4) gets 4
search(5) gets 5
search(6) gets 6
search(7) gets 7
search(8) gets 8
search(9) gets 9
search(10) gets -1
search(11) gets -1
```

This program's output is *sometimes* correct and *sometimes* wrong. Write down six different values of `val` (between 0 and 11, inclusive) when the outputs are still correct.
**DO NOT** correct the program. The purpose of this question is to show that some test cases may reveal coding mistakes while some other test cases cannot.

*Answer:*

`1, 2, 4, 5, 7, 8, 10, 11`

# 4 Recursion (20 points)

Consider the following program:

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <string.h>
 4
 5 int f1(int n, int * count1, int * count2)
 6 {
 7   (* count1) ++;
 8   if (n <= 1)
 9     {
10       return 1;
11     }
12   (* count2) ++;
13   int a = f1(n - 1, count1, count2);
14   int b = f1(n - 2, count1, count2);
15   return a + b;
16 }
17
18 int f2(int n, int * count1, int * count2)
19 {
20   (* count1) ++;
21   if (n <= 1)
22     {
23       return 1;
24     }
25   (* count2) ++;
26   int a = f2(n --, count1, count2);
27   // <--- ERROR --->
28   // notice the difference, f1 uses n - 1, f2 uses n --
29   int b = f2(n - 2, count1, count2);
30   return a + b;
31 }
32
33 int main(int argc, char ** argv)
34 {
35   int count1;
36   int count2;
37   count1 = 0;
38   count2 = 0;
39   int val1 = f1(5, & count1, & count2);
```

```
40    printf("val1 = %d, count1 = %d, count2 = %d\n",
41           val1, count1, count2);
42    // what is the output? Answers 4.A, 4.B, 4.C
43
44    count1 = 0;
45    count2 = 0;
46    int val2 = f2(5, & count1, & count2);
47    printf("val2 = %d, count1 = %d, count2 = %d\n",
48           val2, count1, count2);
49
50    return EXIT_SUCCESS;
51  }
```

The program has "Segmentation fault (core dumped)". If you use `gdb` with the following commands, what is the output?

```
(gdb) r
Program received signal SIGSEGV, Segmentation fault.
...
(gdb) bt
#0  0x000000000040061c in f2 ...
...
#1  0x000000000040066d in f2 (n=??, count1=..., count2=...) at q4.c:????
```

Question 4.D: the value of `n` at frame `#1` (marked by `??`).
Question 4.E: the line number at frame `#1` after `q4.c:` (marked by `????`).
**Answer:**

8
15
7
5
26