ECE264 Fall 2019 Exam 2

8-9 PM, October 17

Please keep the answer sheet clean. Do not use the answer sheet as your scratch space. The answer sheet should contain **only** the answers.

Do not write anything that is not the answers.
Otherwise, you may lose points.

Please use **DARK** ink. If your pen is too light, your answer may not be graded.
Please write at the center of each box.

Value	Character	Value	Character
65	A	97	a
66	B	98	b
67	C	99	c
68	D	100	d
69	E	101	e
70	F	102	f
71	G	103	g
72	H	104	h
73	I	105	i
74	J	106	j
75	K	107	k
76	L	108	l
77	M	109	m
78	N	110	n
79	O	111	o
80	P	112	p
81	Q	113	q
82	R	114	r
83	S	115	s
84	T	116	t
85	U	117	u
86	V	118	v
87	W	119	w
88	X	120	x
89	Y	121	y
90	Z	122	z

1 Integer Partition

A positive integer can be expressed as the sum of a sequence of **positive** integers, or itself. *Integer partition* creates such sequences of integers. For example, 5 can be broken in to the sum of 1 + 2 + 2 or 2 + 3. These two partitions use different numbers, and thus are considered unique partitions. The order of the number in the partition is also important. Thus, 1 + 2 + 2 and 2 + 1 + 2 are considered different partitions because 1 appears in different positions. Below are some examples of integer partitions:

```
1 = 1

2 = 1 + 1
  = 2

3 = 1 + 1 + 1
  = 1 + 2
  = 2 + 1
  = 3

4 = 1 + 1+ 1 + 1
  = 1 + 1 + 2
  = 1 + 2 + 1
  = 1 + 3
  = 2 + 1 + 1
  = 2 + 2
  = 3 + 1
  = 4
```

Number n can be partitioned in 2^{n-1} different ways. For example, 4 can be partitioned in $2^{4-1} = 8$ different ways. Restrictions may be added, for example:

- If 1 cannot be used (i.e., removing all lines that contain 1), 2 can be partitioned in only one way: 2.
- If 1 cannot be used, 4 can be partitioned in only two ways: 2 + 2 and 4.
- If 2 must be used (i.e., keeping only the lines that contain 2), 2 can be partitioned in only one way: 2.
- If 2 must be used, 3 can be partitioned in two ways: 2 + 1 and 1 + 2.
- If 2 must be used, 4 can be partitioned in four ways:
  (a) 1 + 1 + 2
  (b) 1 + 2 + 1
  (c) 2 + 1 + 1
  (d) 2 + 2

  Please notice 2 + 2 is counted only once, even though 2 is used twice in the **same** line.
- If 2 must be used, 5 can be partitioned in nine ways:
  (a) 1 + 1 + 1 + 2
  (b) 1 + 1 + 2 + 1
  (c) 1 + 2 + 1 + 1

(d) $1 + 2 + 2$

(e) $2 + 1 + 1 + 1$

(f) $2 + 1 + 2$

(g) $2 + 2 + 1$

(h) $2 + 3$

(i) $3 + 2$

Please fill this table about how many ways to partition numbers. Please write your answers for A - D in the answer sheet (NOT in the table).

	Restrictions	1	2	3	4	5	6	7	8	9
Number of Lines	No restriction	1	2	4	8	16	32	64	128	256
	1 must not be used	0	1	1	2	3	**A**	8	**B**	21
	2 must be used	0	1	2	4	9	20	**C**	**D**	191

Consider the following program with one mistake. How many lines does this program print? This is answer E. Your answer may be a number, "Segmentation Fault", "Infinite Loop", or something else.

```c
// partition.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void printPartition(int * arr, int length)
{
  int ind;
  for (ind = 0; ind < length - 1; ind ++)
    {
      printf("%d + ", arr[ind]);
    }
  printf("%d\n", arr[length - 1]);
}

void partition(int * arr, int ind, int left)
{
  int val;
  if (left == 0)
    {
      printPartition(arr, ind);
      return; // not necessary
    }
  for (val = 1; val < left; val ++)
    // <---
    // ERROR, should be <=, but it is < here
    // DO NOT CORRECT THE MISTAKE
    // --->
    {
      arr[ind] = val;
      partition(arr, ind + 1, left - val);
    }
}

int main(int argc, char * argv[])
{
  int n = 6;
  int * arr;
  arr = malloc(sizeof(int) * n);
  partition(arr, 0, n);
  free (arr);
  return EXIT_SUCCESS;
}
```

Answer:

Number to partition	1	2	3	4	5	6	7	8	9
No restriction	1	2	4	8	16	32	64	128	256
1 must not be used	0	1	1	2	3	**5**	8	**13**	21
2 must be used	0	1	2	4	9	20	**43**	**91**	191

0

2 Recursion

An arithmetic expression may include one or multiple pairs of parentheses, for example

```
(3 + 5) * 4
(3 + (5 + 4) * 6) * 2
3 + (5 + 4) * (6 + 2)
(3 + (5 + 4)) * (6 + 2)
```

Consider a program (shown at the end of this question) that generates pairs of parentheses by following these **rules**:

- The number of left parentheses `(` must be the same as number of right parentheses `)`.
- From left to right, the number of right parentheses `)` must not exceed the number of left parentheses `(` that have already been seen.

The program's ouputs for one, two, three pairs are

```
()

(())
()()

((()))
(()())
(())()
()(())
()()()
```

The following are *invalid* and *not* generated by this program:

```
)(
())(
))((
)()()(
```

This question asks you to determine **the number of lines** printed by this program for different numbers of inputs.

Number of pairs	Number of lines in the output
1	1
2	2
3	5

You need to answer the numbers of lines for 4, 5, and 6 pairs. To guide you through this problem, consider to create a function $f(n)$ as the number of lines when n pairs of `(` and `)` are printed. Since this problem considers `(` and `)`, $f(n)$ is further expressed by a function with two arguments: $f(n) = g(n, n)$. Here

> $g(n, m)$ is defined as **the number of lines to print** n `(` **and** m `)`,
> while satifying the rules described above.

Consider printing three pairs of `(` and `)`. The first is always `(`. After using one `(`, two `(` and three `)` are left. Thus, $f(3) = g(3, 3) = g(2, 3)$.
The second can be `(` or `)`:

- If `(` is used, one `(` and three `)` are left. There are $g(1, 3)$ lines.
- If `)` is used, two `(` and two `)` are left. There are $g(2, 2)$ lines.

Please answer the following questions. Your answers may be numbers or symbols.

A: What is $g(0, m)$ when $m > 0$?

B: What is $g(n, m)$ when $n > m$?

C: What is $f(4) = g(4, 4)$?

D: What is $f(5) = g(5, 5)$?

E: What is $f(6) = g(6, 6)$?

The following program is for your reference. You do not need to read this program to answer the questions.

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 void generate(char * parentheses, int num, int left, int right)
 4 {
 5   // num: total number of pairs
 6   // left: how many left parentheses have been used
 7   // right: how many right parentheses have been used
 8   int ind = left + right;
 9   if (left == num)   // use up all '('
10     {
11       for (int i = 0; i < ind; i ++)
12         {
13           printf("%c", parentheses[i]);
```

```
14          }
15        // use all remaining ')'
16        for (int i = right; i < num; i ++)
17          {
18            printf(")");
19          }
20        printf("\n");
21        return;
22      }
23    // case 1: add '('. always possible because left < num
24    parentheses[ind] = '(';
25    generate(parentheses, num, left + 1, right);
26    // case 2: check whether ')' can be added
27    if (left > right)
28      {
29        parentheses[ind] = ')';
30        generate(parentheses, num, left, right + 1);
31      }
32  }
33
34  int main(int argc, char * * argv)
35  {
36    if (argc < 2)
37      {
38        return EXIT_FAILURE;
39      }
40    // num: how many pairs
41    int num = (int) strtol(argv[1], NULL, 10);
42    if (num < 1)
43      {
44        return EXIT_FAILURE;
45      }
46    char * parentheses = malloc(sizeof (* parentheses) * num * 2);
47    generate(parentheses, num, 0, 0);
48    free (parentheses);
49    return EXIT_SUCCESS;
50  }
```

Answer:
A: 1
B: 0
C: 14
D: 42

E: 132

3 Structure and File

Consider the following C program. What are the outputs?

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #pragma pack(1)
4  // tell compiler not to pad any space between
5  // attributes in structure
6
7  // assume the size of the data types
8  // char:    1 byte
9  // int:     4 bytes
10 // double:  8 bytes
11 // pointer: 8 bytes
12
13 #define NUMVECTOR 10
14 typedef struct
15 {
16   int x;
17   int y;
18   int z;
19   double t;
20 } Vector;
21
22 int main(int argc, char ** argv)
23 {
24   Vector varr[NUMVECTOR];
25   printf("%ld\n", sizeof(varr)); // <- Answer A: the output?
26   Vector * vptr;
27   vptr = & varr[0];
28   printf("%ld\n", sizeof(vptr)); // <- Answer B: the output?
29   // Hint: think carefully about the difference of A and B
30
31   // initialize the attributes
32   int ind;
33   for (ind = 0; ind < NUMVECTOR; ind ++)
34     {
35       varr[ind].x = ind;
36       varr[ind].y = ind * 2;
37       varr[ind].z = ind * 3;
38       varr[ind].t = ind * 4.0;
39     }
```

```
40
41    // allocate memory
42    vptr = malloc(sizeof(Vector) * 100);
43
44    FILE * fptr = fopen("data", "w");
45    // assume fopen succeeds
46    fwrite(varr, sizeof(Vector), NUMVECTOR, fptr);
47    fclose (fptr);
48
49    fptr = fopen("data", "r");
50    int rtv = fseek(fptr, 48, SEEK_SET);
51    if (rtv != 0)
52      {
53        fclose (fptr);
54        return EXIT_FAILURE;
55      }
56    // assume fseek succeeds
57
58    int a = -2019;
59    rtv = fread(& a, sizeof(int), 1, fptr);
60    if (rtv == 0)
61      {
62        fclose (fptr);
63        return EXIT_FAILURE;
64      }
65
66    printf("%d\n", a);  // <- Answer C: the output?
67
68    Vector v;
69    v = varr[3];
70    v.x = -264;
71
72    printf("%d\n", varr[3].x - v.x); // <- Answer D: the output?
73
74    // release memory
75    free (vptr);
76
77    printf("%d\n", (vptr == NULL));  // <- Answer E: the output?
78    // please notice ==, it is not =
79
80    return EXIT_SUCCESS;
81  }
```

Answer:

`200, 8, 6, 267, 0`

4 Trace Recursive Function

What is the output of this program? Hint: Draw the call tree.

```
 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 a;
12   int b;
13   int c;
14   int d;
15 } Counter;
16
17 int func(Counter * cnt, int n)
18 {
19   (cnt -> a) ++;
20   if (n == 0)
21     {
22       (cnt -> b) ++;
23       return 1;
24     }
25   if (n == 1)
26     {
27       (cnt -> c) ++;
28       return 2;
29     }
30   int x = func(cnt, n - 1);
31   int y = func(cnt, n - 2);
32   (cnt -> d) ++;
33   return (x + y);
34 }
35
36 int main(int argc, char ** argv)
37 {
38   Counter cnt =
39     {
```

```
40        .a = 0,
41        .b = 0,
42        .c = 0,
43        .d = 0
44      };
45    int v = func(& cnt, 5);
46    // The outpus are the answers for A - D
47    printf("cnt = (%d, %d, %d, %d)\n",
48           cnt.a, cnt.b, cnt.c, cnt.d);
49
50    // The output is the answer E
51    printf("v = %d\n", v);
52    return EXIT_SUCCESS;
53  }
```

Answer:

```
15
3
5
7
13
```