ECE264 Spring 2018
Exam 3, 630-730PM, 2018

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 3)
Dynamic Structure (Question 2 and 3)
You must obtain 50% or more points in each of the corresponding question to pass the learning objective.

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

Contents

Total Score:

Learning Objective

Dynamic Structure	Pass	Fail
Recursion	Pass	Fail

	Q1	Q2	Q3
A			
B			
C			
D			
E			
F			
G			
H			

This page is blank.

1 Bit Operations (30 points)

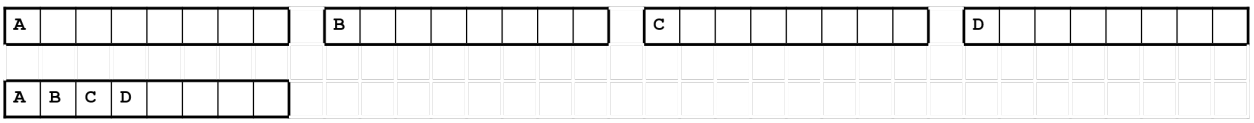

Figure 1: Compress data by taking the leftmost bit from each byte.

This problem asks you to compress an input file by taking one bit from each byte. If the length of the input file is not a multiple of eight, zeros are added at the end of the last byte of the output.
Fill the following code

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  // take the leftmost bit from each byte in orig
6  // pack these bits to dest
7  // size if the number of bytes in orig
8  //
9  // dest should have enough memory to store the result
10 // convert does not allow memory in heap
11
12 void convert(const unsigned char * orig,
13              unsigned char * dest,
14              int size)
15 {
16   int bitCount = 0;  // how many bits have been added
17   int destCount = 0; // index for the destination array
18
19   // ---> FIX ME <--- 1.A
20   unsigned char bitMask = ??; // binary 1000 0000
21
22
23   unsigned char destByte = 0;  // destination byte
24   int cnt;
25   for (cnt = 0; cnt < size; cnt ++)
26     {
27       // get the leftmost bit
28       // ---> FIX ME <--- 1.B
29       unsigned char oneBit = ???;
30
```

```
31        // shift right to the correct location
32        // ---> FIX ME <--- 1.C
33        unsigned char destBit = ???;
34
35        // add the bit to the destination
36        // ---> FIX ME <--- 1.D
37        destByte = ???;
38
39        // increment bitCount
40        bitCount ++;
41
42        // fill one byte
43        if ((bitCount % 8) == 0) // filled one output byte
44          {
45            dest[destCount] = destByte;
46            destCount ++;
47            destByte = 0;
48          }
49      }
50    // If size is not a multliple of 8, handle the last byte
51    if ((size % 8) != 0)
52      {
53        dest[destCount] = destByte;
54        // no need to update destCount
55      }
56  }
57
58  int main(int argc, char * argv[])
59  {
60    // argv[1]: input file
61    // argv[2]: output file
62    if (argc < 3)
63      {
64        return EXIT_FAILURE;
65      }
66    FILE * fpin = fopen(argv[1], "r");
67    if (fpin == NULL)
68      {
69        return EXIT_FAILURE;
70      }
71    FILE * fpout = fopen(argv[2], "w");
72    if (fpout == NULL)
```

```
73      {
74        fclose (fpin);
75        return EXIT_FAILURE;
76      }
77
78    // find the length of the input file
79    fseek(fpin, 0, SEEK_END);
80    long length = ftell(fpin);
81    // allocate memory for the input
82    unsigned char * orig = malloc(sizeof(unsigned char) * length);
83    // return the beginning of the file
84    fseek(fpin, 0, SEEK_SET);
85
86    // read the input from the file
87    // ---> FIX ME <--- 1.E
88    int rtv = fread(orig, sizeof(unsigned char), ???);
89    if (rtv != length)
90      {
91        printf("fread fail, rtv = %d, length = %ld\n",
92               rtv, length);
93      }
94
95    // calculate the output's length
96    long outlength = length / 8;
97    if ((length % 8) != 0)
98      {
99        outlength ++;
100     }
101
102   // allocate memory for the output
103   unsigned char * dest = malloc(sizeof(unsigned char) * outlength);
104
105   convert(orig, dest, length);
106
107   // write the result to the output file
108   rtv = fwrite(dest, sizeof(unsigned char), outlength, fpout);
109
110   if (rtv != outlength)
111     {
112       printf("fwrite fail, rtv = %d, outlength = %ld\n",
113              rtv, outlength);
114     }
```

```
115
116    // ---> FIX ME <--- 1.F
117    // close the input file
118    ???;
119
120    // close the output file
121    fclose (fpout);
122    return EXIT_SUCCESS;
123 }
```

```
       #include <stdio.h>

       size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);

       size_t fwrite(const void *ptr, size_t size, size_t nmemb,
                     FILE *stream);
```

```
DESCRIPTION

       The function fread() reads nmemb items of data, each size bytes
       long, from the stream pointed to by stream, storing them at the
       location given by ptr.

       The function fwrite() writes nmemb items of data, each size
       bytes long, to the stream pointed to by stream, obtaining them
       from the location given by ptr.

       For nonlocking counterparts, see unlocked_stdio(3).

RETURN VALUE

       On success, fread() and fwrite() return the number of items
       read or written.  This number equals the number of bytes
       transferred only when size is 1.  If an error occurs, or the
       end of the file is reached, the return value is a short item
       count (or zero).
```

Answer:

```
0X80
orig[cnt] & bitMask
oneBit >> (bitCount % 8)
destByte | destBit
```

```
length, fpin
fclose (fpout);
```

2 Recursive Program (20 points)

Consider the following program and write down the answers.

```
 1 #include <stdlib.h>
 2 #include <stdio.h>
 3 #include <time.h>
 4
 5 typedef struct
 6 {
 7   int cnt1;
 8   int cnt2;
 9   int cnt3;
10 } Counters;
11
12 int func(int n, Counters * count)
13 {
14   (count -> cnt1) ++;
15   if (n == 0)
16     {
17       return 0;
18     }
19   (count -> cnt2) ++;
20   int i;
21   int sum = 0;
22   for (i = 1; i <= n; i ++)
23     {
24       (count -> cnt3) += i;
25       func(n - i, count);
26       sum += i;
27     }
28   return sum;
29 }
30
31 int main(int argc, char * argv[])
32 {
33   Counters count = // initialize attributes to zero
34     { .cnt1 = 0,
35       .cnt2 = 0,
36       .cnt3 = 0
37     };
38   int result = func(4, & count);
39   printf("result = %d\n", result);
```

```
40    printf("Counters = %d, %d, %d\n",
41           count.cnt1,
42           count.cnt2,
43           count.cnt3);
44    return EXIT_SUCCESS;
45  }
```

Answer:
10
16, 8, 26

3 Binar Search (30 points)

Consider the following function that implements binary search. If the function were correct, the output should be 0, 1, 2, 3, 4, 5. Due to a mistake in the program, the program's output is different. Please write down the program's output.

```
1  // search.c
2  int searchHelper(int * arr, int key, int low, int high)
3  {
4    if (low >= high) // ERROR, should be low > high
5      {
6        return -1;
7      }
8    int mid = (low + high) / 2;
9    if (arr[mid] == key)
10     {
11       return mid;
12     }
13   if (arr[mid] > key)
14     {
15       return searchHelper(arr, key, low, mid - 1);
16     }
17   return searchHelper(arr, key, mid + 1, high);
18 }
19 int binarysearch(int * arr, int size, int key)
20 {
21   return searchHelper(arr, key, 0, size - 1);
22 }
```

```
1  // main.c
2  #include <stdio.h>
3  #include <stdlib.h>
4  int binarysearch(int * arr, int size, int key);
5  #define ARRAYSIZE 6
6  int main(int argc, char * * argv)
7  {
8    int arr[ARRAYSIZE] = {1, 12, 23, 44, 65, 76};
9    int ind;
10   for (ind = 0; ind < ARRAYSIZE; ind ++)
11     {
12       printf("%d\n", binarysearch(arr, ARRAYSIZE, arr[ind]));
13     }
14   return EXIT_SUCCESS;
15 }
```

Answer:
0 -1 2 -1 4 -1