ECE264 Fall 2017
Final Exam, 330-530PM, December 15

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.

Total Score:

Learning Objective

Dynamic Structure	Pass	Fail
Recursion	Pass	Fail

	Q1	Q2	Q3
A			
B			
C			
D			
E			
F			
G			
H			

1 Binary Search Tree (32 points)

This is the order of numbers inserted into the tree:
5290 6719 6788 4575 3061 6498 864 9277 8745 9627
Consider the following two methods, `Tree_printPreorder` and `Tree_printPostorder`, to print the nodes of a binary search tree.

```
1  // treeprint.c
2  #include "tree.h"
3  static void TreeNode_print(TreeNode *tn)
4  {
5    printf("address = %10p ", tn);
6    printf("value = %5d ", tn -> value);
7    printf("left = %10p ", (void *) tn -> left);
8    printf("right = %10p\n", (void *) tn -> right);
9  }
10
11 void Tree_printPreorder(TreeNode *tn)
12 {
13   if (tn == NULL)
14     {
15       return;
16     }
17   TreeNode_print(tn);
18   Tree_printPreorder(tn -> left);
19   Tree_printPreorder(tn -> right);
20 }
21
22 void Tree_printPostorder(TreeNode *tn)
23 {
24   if (tn == NULL)
25     {
26       return;
27     }
28   Tree_printPostorder(tn -> left);
29   Tree_printPostorder(tn -> right);
30   TreeNode_print(tn);
31 }
```

If the output of `Tree_printPreorder` is shown below:

```
address =   0x450 value =  5290 left =   0x4b0 right =   0x470
address =   0x4b0 value =  4575 left =   0x4d0 right =   (nil)
address =   0x4d0 value =  3061 left =   0x510 right =   (nil)
address =   0x510 value =   864 left =   (nil) right =   (nil)
address =   0x470 value =  6719 left =   0x4f0 right =   0x490
address =   0x4f0 value =  6498 left =   (nil) right =   (nil)
address =   0x490 value =  6788 left =   (nil) right =   0x530
address =   0x530 value =  9277 left =   0x550 right =   0x570
address =   0x550 value =  8745 left =   (nil) right =   (nil)
address =   0x570 value =  9627 left =   (nil) right =   (nil)
```

1.1 Postorder (20 points)

Fill the following table as the output of `Tree_printPostorder`. Do not worry about the places marked as *.

```
address =  1.A  value =  1.B  left =  1.C  right =  *
address =  1.D  value =  1.E  left =  1.F  right =  1.G
address =  1.H  value =    *  left =    *  right =  *
address =  1.I  value =    *  left =    *  right =  *
address =  1.J  value =    *  left =    *  right =  *
address =    *  value =    *  left =    *  right =  *
address =    *  value =    *  left =    *  right =  *
address =    *  value =    *  left =    *  right =  *
address =    *  value =    *  left =    *  right =  *
address =    *  value =    *  left =    *  right =  *
```

1.2 Delete Last Inserted Value (6 points)

Consider the following function `Tree_delete` deleting one node in the tree. If the last added value (9627) is deleted, what is the output of the program when using `printPreorder`?

```
1  // treedelete.c
2  #include "tree.h"
3  #include <stdlib.h>
4  TreeNode * Tree_delete(TreeNode * tn, int val)
5  {
6    if (tn == NULL) { return NULL; }
7    if (val < (tn -> value))
8      {
9        tn -> left = Tree_delete(tn -> left, val);
10       return tn;
11     }
12   if (val > (tn -> value))
13     {
14       tn -> right = Tree_delete(tn -> right, val);
15       return tn;
16     }
17   // v is the same as tn ->  value
18   if (((tn ->  left) == NULL) && ((tn ->  right) == NULL))
19     {
20       // tn has no child
21       free (tn);
22       return NULL;
23     }
24   if ((tn ->  left) == NULL)
25     {
26       // tn ->  right must not be NULL
27       TreeNode * rc = tn ->  right;
28       free (tn);
29       return rc;
30     }
31   if ((tn ->  right) == NULL)
32     {
33       // tn ->  left must not be NULL
34       TreeNode * lc = tn ->  left;
35       free (tn);
36       return lc;
37     }
38   // tn have two children
39   TreeNode * su = tn ->  right; // su must not be NULL
```

```
40    while ((su -> left) != NULL)
41      {
42        su = su -> left;
43      }
44    // su is tn's immediate successor
45    // swap their values
46    tn ->  value = su -> value;
47    su -> value = val;
48    // delete su
49    tn ->  right = Tree_delete(tn ->  right, val);
50    return tn;
51  }
```

```
address =  1.K  value =  *  left =  *  right =  *
address =  1.L  value =  *  left =  *  right =  *
address =    *  value =  *  left =  *  right =  *
address =    *  value =  *  left =  *  right =  *
address =    *  value =  *  left =  *  right =  *
address =    *  value =  *  left =  *  right =  *
address =    *  value =  *  left =  *  right =  *
address =    *  value =  *  left =  *  right =  *
address =  1.M  value =  *  left =  *  right =  *
```

1.3 Delete First Inserted Value (6 points)

If the first added value (5290) is deleted (after deleting 9627), what is the output of the program when using `printPreorder`?

```
address =  *  value =  1.N  left =  1.O  right =  1.P
address =  *  value =    *  left =    *  right =  *
address =  *  value =    *  left =    *  right =  *
address =  *  value =    *  left =    *  right =  *
address =  *  value =    *  left =    *  right =  *
address =  *  value =    *  left =    *  right =  *
address =  *  value =    *  left =    *  right =  *
address =  *  value =    *  left =    *  right =  *
address =  *  value =    *  left =    *  right =  *
```

Answer:

```
address =   0x510 value =   864 left =   (nil) right =   (nil)
address =   0x4d0 value =  3061 left =   0x510 right =   (nil)
address =   0x4b0 value =  4575 left =   0x4d0 right =   (nil)
address =   0x4f0 value =  6498 left =   (nil) right =   (nil)
address =   0x550 value =  8745 left =   (nil) right =   (nil)
address =   0x570 value =  9627 left =   (nil) right =   (nil)
```

```
address =   0x530 value =  9277 left =   0x550 right =   0x570
address =   0x490 value =  6788 left =   (nil) right =   0x530
address =   0x470 value =  6719 left =   0x4f0 right =   0x490
address =   0x450 value =  5290 left =   0x4b0 right =   0x470

address =   0x450 value =  5290 left =   0x4b0 right =   0x470
address =   0x4b0 value =  4575 left =   0x4d0 right =   (nil)
address =   0x4d0 value =  3061 left =   0x510 right =   (nil)
address =   0x510 value =   864 left =   (nil) right =   (nil)
address =   0x470 value =  6719 left =   0x4f0 right =   0x490
address =   0x4f0 value =  6498 left =   (nil) right =   (nil)
address =   0x490 value =  6788 left =   (nil) right =   0x530
address =   0x530 value =  9277 left =   0x550 right =   (nil)
address =   0x550 value =  8745 left =   (nil) right =   (nil)

address =   0x450 value =  6498 left =   0x4b0 right =   0x470
address =   0x4b0 value =  4575 left =   0x4d0 right =   (nil)
address =   0x4d0 value =  3061 left =   0x510 right =   (nil)
address =   0x510 value =   864 left =   (nil) right =   (nil)
address =   0x470 value =  6719 left =   (nil) right =   0x490
address =   0x490 value =  6788 left =   (nil) right =   0x530
address =   0x530 value =  9277 left =   0x550 right =   (nil)
address =   0x550 value =  8745 left =   (nil) right =   (nil)
```

2 Structure and File (24 points)

Hint: Consider what you learned for HW15.

```
// q2.h
#ifndef Q2_H
#define Q2_H
#include <stdio.h>
#include <stdlib.h>
typedef struct q2char
{
  char data[4];
} CharType;

typedef struct q2int
{
  int data;
} IntType;
#endif
```

```
#include <stdio.h>
#include <stdlib.h>
#include "q2.h"

void cleanup(IntType ** intarr, CharType ** chararr, FILE ** fptr)
{
  if ((* intarr)  != NULL) { free   (* intarr); }
  if ((* chararr) != NULL) { free   (* chararr); }
  if ((* fptr)    != NULL) { fclose (* fptr); }
  * intarr  = NULL;
  * chararr = NULL;
  * fptr    = NULL;
}

int main(int argc, char * * argv)
{
  // assume sizeof(int) = 4 and sizeof(char) = 1
  IntType  * intarr  = NULL;
  CharType * chararr = NULL;
  FILE     * fptr    = NULL;
  intarr = malloc(sizeof(* intarr) * 2);
  if (intarr == NULL)
    {
      fprintf(stderr, "malloc fail\n");
```

```
25        return EXIT_FAILURE;
26      }
27    intarr[0].data = 0X45434550;
28    intarr[1].data = 0X55524432;
29    fptr = fopen("data", "w");
30    if (fptr == NULL)
31      {
32        fprintf(stderr, "fopen fail\n");
33        cleanup(& intarr, & chararr, & fptr);
34        return EXIT_FAILURE;
35      }
36    if (fwrite(intarr, sizeof(* intarr), 2, fptr) != 2)
37      {
38        fprintf(stderr, "fwrite fail\n");
39        cleanup(& intarr, & chararr, & fptr);
40        return EXIT_FAILURE;
41      }
42    cleanup(& intarr, & chararr, & fptr);
43    fptr = fopen("data", "r");
44    if (fptr == NULL)
45      {
46        fprintf(stderr, "fopen fail\n");
47        cleanup(& intarr, & chararr, & fptr);
48        return EXIT_FAILURE;
49      }
50    chararr = malloc(sizeof(* chararr) * 2);
51    if (chararr == NULL)
52      {
53        fprintf(stderr, "malloc fail\n");
54        cleanup(& intarr, & chararr, & fptr);
55        return EXIT_FAILURE;
56      }
57    if (fread(chararr, sizeof(* chararr), 2, fptr) != 2)
58      {
59        fprintf(stderr, "fread fail\n");
60        cleanup(& intarr, & chararr, & fptr);
61        return EXIT_FAILURE;
62      }
63    int i, j;
64    for (i = 0; i < 2; i ++)
65      {
66        for (j = 0; j < 4; j ++)
```

```
67          {
68            printf("%c", chararr[i].data[j]);
69          }
70      }
71    printf("\n");
72    cleanup(& intarr, & chararr, & fptr);
73    return EXIT_SUCCESS;
74  }
```

What is the output of this program? The output has 8 characters as 2.A - 2.H.
Write down the output of `xxd data`? Do not worry about the characters after the four integers. The answers should be entered into 2.I - 2.L on page 2.
`00000000: 2.I 2.J 2.K 2.L`
Answer:
PECE2DRU
half points for ECEPURD2
5045 4345 3244 5255

3 Sort Linked Lists (20 points)

Consider the following function that sort a linked list of generic type (using `void *`).

```
1  // list.h
2  #ifndef LIST_H
3  #define LIST_H
4  #include <stdio.h>
5  #include <stdlib.h>
6  typedef struct listnode
7  {
8    struct listnode * next;
9    void * ptr;
10 } Node;
11
12 Node * List_insert(Node * head, void * data);
13
14 // sort in the ascending order
15 Node * List_sort(Node * head,
16                  int (*compfunc)(const void *, const void *));
17 void List_print(Node * head, void (* printfunc)(const void * data));
18 void List_destroy(Node * head,  void (* deletefunc)(void * data));
19
20 #endif
```

Please fill the code:

```
1  // listmain.c
2  #include <stdlib.h>
3  #include <stdio.h>
4  #include <time.h>
5  #include "list.h"
6
7  void printInt(const void * data)
8  {
9    // 3.A convert const void * to int *
10   // <--- FIX ME --->
11   printf("%d ", * ptr);
12 }
13
14 void deleteInt(void * data)
15 {
16   int * ptr = (int *) data;
17   free (ptr);
18 }
```

```
19
20 int compInt(const void * data1, const void * data2)
21 {
22   // 3.B
23   // <--- FIX ME --->
24   // convert data1 to int * and call it ptr1
25   // convert data2 to int * and call it ptr2
26
27   int val1 = * ptr1;  // get integer val1 from the ptr1
28   int val2 = * ptr2;  // get integer val2 from the ptr2
29   return (val1 - val2);
30 }
31
32 int main(int argc, char * argv[])
33 {
34   srand(time(NULL));
35   Node * head = NULL;
36   for (int i = 0; i < 10; i ++)
37     {
38       int * iptr = malloc(sizeof(int));
39       * iptr = rand() % 1000;
40       head = List_insert(head, iptr);
41     }
42   List_print(head, printInt);
43   head = List_sort(head, compInt);
44   List_print(head, printInt);
45   List_destroy(head, deleteInt);
46   return EXIT_SUCCESS;
47 }
48
49 void List_destroy(Node * head,  void (* delete)(void * data))
50 {
51   Node * p = head;
52   while (p != NULL)
53     {
54       p = p -> next;
55       delete(head -> ptr);
56       free (head);
57       head = p;
58     }
59 }
60
```

```
 61 Node * List_insert(Node * head, void * data)
 62 {
 63   Node * n = malloc(sizeof(Node));
 64   n -> ptr = data;
 65   n -> next = head; // insert at the beginning
 66   return n;
 67 }
 68
 69 void List_print(Node * head, void (* printfunc)(const void * data))
 70 {
 71   while (head != NULL)
 72     {
 73       printfunc(head -> ptr);
 74       if  (head -> next != NULL)
 75         {
 76           printf(" => ");
 77         }
 78       head = head -> next;
 79     }
 80   printf("\n\n");
 81 }
 82
 83 Node * List_sort(Node * head,
 84                  int (*compfunc)(const void *, const void *))
 85 {
 86   Node * p = head;
 87   Node * q;
 88   Node * r;
 89   if (p == NULL) { return p; } // nothing to sort
 90   if ((p -> next) == NULL) { return p; } // only one node
 91   while (p != NULL)
 92     {
 93       r = p;
 94       q = p -> next;
 95       while (q != NULL)
 96         {
 97           // 3.C
 98           // <--- FIX ME --->
 99           // call compfunc to compare the data stored in r and q
100           // Hint: You should not compare r and q; instead, you
101           // should compare what are stored in their ptr
102           if (3.D)    // <--- FIX ME --->
```

```
103             {
104               r = q;
105             }
106           q = q -> next;
107         }
108       // 3.E
109       // <--- FIX ME --->
110       // exchange the data stored in p and r
111
112       p = p -> next;
113     }
114   return head;
115 }
```

Answer:

```
 1 // listmain.c
 2 #include "list.h"
 3 #include <stdlib.h>
 4 #include <stdio.h>
 5 #include <time.h>
 6
 7 void printInt(const void * data)
 8 {
 9   const int * ptr = (const int *) data;
10   printf("%d ", * ptr);
11 }
12
13 void deleteInt(void * data)
14 {
15   int * ptr = (int *) data;
16   free (ptr);
17 }
18
19 int compInt(const void * data1, const void * data2)
20 {
21   const int * ptr1 = (const int *) data1;
22   const int * ptr2 = (const int *) data2;
23   int val1 = * ptr1;
24   int val2 = * ptr2;
25   return (val1 - val2);
26 }
27
28 int main(int argc, char * argv[])
```

```
29 {
30   srand(time(NULL));
31   Node * head = NULL;
32   for (int i = 0; i < 10; i ++)
33     {
34       int * iptr = malloc(sizeof(int));
35       * iptr = rand() % 1000;
36       head = List_insert(head, iptr);
37     }
38   List_print(head, printInt);
39   head = List_sort(head, compInt);
40   List_print(head, printInt);
41   List_destroy(head, deleteInt);
42   return EXIT_SUCCESS;
43 }
```

```
1 #include "list.h"
2 void List_destroy(Node * head,  void (* delete)(void * data))
3 {
4   Node * p = head;
5   while (p != NULL)
6     {
7       p = p -> next;
8       delete(head -> ptr);
9       free (head);
10      head = p;
11    }
12 }
```

```
1 // listinsert.c
2 #include "list.h"
3 Node * List_insert(Node * head, void * data)
4 {
5   Node * n = malloc(sizeof(Node));
6   n -> ptr = data;
7   n -> next = head; // insert at the beginning
8   return n;
9 }
```

```
1 #include "list.h"
2 void List_print(Node * head, void (* printfunc)(const void * data))
3 {
4   while (head != NULL)
```

```
 5     {
 6       printfunc(head -> ptr);
 7       if  (head -> next != NULL)
 8         {
 9           printf(" => ");
10         }
11       head = head -> next;
12     }
13   printf("\n\n");
14 }
```

```
 1 #include "list.h"
 2 Node * List_sort(Node * head,
 3                  int (*compfunc)(const void *, const void *))
 4 {
 5   Node * p = head;
 6   Node * q;
 7   Node * r;
 8   if (p == NULL) { return p; } // nothing to sort
 9   if ((p -> next) == NULL) { return p; } // only one node
10   while (p != NULL)
11     {
12       r = p;
13       q = p -> next;
14       while (q != NULL)
15         {
16           if (compfunc (r -> ptr, q -> ptr) > 0)
17             {
18               r = q;
19             }
20           q = q -> next;
21         }
22       // exchange
23       void * tmpptr = p -> ptr;
24       p -> ptr = r -> ptr;
25       r -> ptr = tmpptr;
26       p = p -> next;
27     }
28   return head;
29 }
```

4 Determine Palindrome (12 points)

A *palindrome* is a word, phrase, number, or other sequence of characters which reads the same backward as forward, such as `madam` or `racecar`. This question asks you to write a *recursive* function that determines whether a file's content is a palindrome.
Here are some more examples of palindrome:
`neveroddoreven`, `madamImadam`, `madam I madam`, `neve roddor even`.
Please fill the code:

```
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
bool isPalindrome(char * filecontent, long int filesize,
                  long int curlocation)
// return true if filecontent is a palindrome
// return false if it is not
{
  // algorithm:
  // ----------------------------------------
  // ^                                      ^ compare these two
  //  ^                                    ^  compare these two
  //   ^                                  ^   compare these two
  // If the comparison reaches the middle of the file,
  // it is not necessary to compare further.
  if (curlocation >= (filesize / 2))
    {
      // <--- FIX ME ---> 4.A
    }
  if (filecontent[curlocation] !=
      filecontent[filesize - curlocation -1])
    {
      // <--- FIX ME ---> 4.B

    }
  // <--- FIX ME ---> 4.C
  // call isPalindrome again, using the correct arguments
  return
}

int main(int argc, char * argv[])
{
  if (argc != 2)
    {
```

```
35        return EXIT_FAILURE;
36      }
37    FILE * fptr = fopen(argv[1], "r");
38    if (fptr == NULL)
39      {
40        return EXIT_FAILURE;
41      }
42    // go to the end of the file
43    if (fseek(/* <--- FIX ME ---> 4.D */) == -1)
44      {
45        fprintf(stderr, "fseek fail\n");
46        return EXIT_FAILURE;
47      }
48    // find the size of the file
49    long int filesize = ftell (/* <--- FIX ME ---> 4.E */);
50
51    char * filecontent = malloc(sizeof(* filecontent) * filesize);
52    // go to the beginning of the file
53    if (fseek(fptr, 0, SEEK_SET) == -1)
54      {
55        fprintf(stderr, "fseek fail\n");
56        return EXIT_FAILURE;
57      }
58    // read the entire file and store the content in filecontent
59    if (fread(/* <--- FIX ME ---> 4.F */) != filesize)
60      {
61        fprintf(stderr, "fread fail\n");
62        return EXIT_FAILURE;
63      }
64    fclose (fptr);
65    if (isPalindrome(filecontent, filesize, 0) == true)
66      {
67        printf("Yes, %s is a palindrome\n", argv[1]);
68      }
69    else
70      {
71        printf("No, %s is not a palindrome\n", argv[1]);
72      }
73    free (filecontent);
74    return EXIT_SUCCESS;
75  }
```

The following shows the manual page of `fseek` and `ftell`.

NAME

fgetpos, fseek, fsetpos, ftell, rewind - reposition a stream

SYNOPSIS

```
#include <stdio.h>

int fseek(FILE *stream, long offset, int whence);

long ftell(FILE *stream);

void rewind(FILE *stream);

int fgetpos(FILE *stream, fpos_t *pos);
int fsetpos(FILE *stream, const fpos_t *pos);
```

DESCRIPTION

The fseek() function sets the file position indicator for the stream pointed to by stream. The new position, mea sured in bytes, is obtained by adding offset bytes to the position specified by whence. If whence is set to SEEK_SET, SEEK_CUR, or SEEK_END, the offset is relative to the start of the file, the current position indicator, or end-of-file, respectively. A successful call to the fseek() function clears the end-of-file indicator for the stream and undoes any effects of the ungetc(3) function on the same stream.

The ftell() function obtains the current value of the file position indicator for the stream pointed to by stream.

The rewind() function sets the file position indicator for the stream pointed to by stream to the beginning of the file. It is equivalent to:

```
(void) fseek(stream, 0L, SEEK_SET)
```

except that the error indicator for the stream is also cleared (see clearerr(3)).

The fgetpos() and fsetpos() functions are alternate interfaces equivalent to ftell() and fseek() (with whence set to SEEK_SET), setting and storing the current value of the file

```
       offset into or from the object referenced by pos.  On some
       non-UNIX systems, an fpos_t object may be a complex object and
       these routines may be the only way to portably reposition a
       text stream.

RETURN VALUE

       The rewind() function returns no value.  Upon successful
       completion, fgetpos(), fseek(), fsetpos() return 0, and ftell()
       returns the current offset.  Otherwise, -1 is returned and
       errno is set to indicate the error.
```

The following shows the manual page of `fread` and `fwrite`.

```
NAME
       fread, fwrite - binary stream input/output

SYNOPSIS
       #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).
```

```
       fread() does not distinguish between end-of-file and error, and
       callers must use feof(3) and ferror(3) to determine which
       occurred.
```

Answer:

```
 1 #include <stdlib.h>
 2 #include <stdio.h>
 3 #include <stdbool.h>
 4
 5 bool isPalindrome(char * filecontent, long int filesize, long int curloca
 6 {
 7   if (curlocation >= (filesize / 2))
 8     {
 9       return true;
10     }
11   if (filecontent[curlocation] != filecontent[filesize - curlocation -1])
12     {
13       return false;
14     }
15   return isPalindrome(filecontent, filesize, curlocation + 1);
16 }
17
18 int main(int argc, char * argv[])
19 {
20   if (argc != 2)
21     {
22       return EXIT_FAILURE;
23     }
24   FILE * fptr = fopen(argv[1], "r");
25   if (fptr == NULL)
26     {
27       return EXIT_FAILURE;
28     }
29   // find the size of the file
30   if (fseek(fptr, 0, SEEK_END) == -1)
31     {
32       fprintf(stderr, "fseek fail\n");
33       return EXIT_FAILURE;
34     }
35   long int filesize = ftell(fptr);
36   char * filecontent = malloc(sizeof(* filecontent) * filesize);
37   // go to the beginning of the file
38   if (fseek(fptr, 0, SEEK_SET) == -1)
```

```
39      {
40        fprintf(stderr, "fseek fail\n");
41        return EXIT_FAILURE;
42      }
43    if (fread(filecontent, sizeof(* filecontent), filesize, fptr) != filesi
44      {
45        fprintf(stderr, "fread fail\n");
46        return EXIT_FAILURE;
47      }
48    fclose (fptr);
49    if (isPalindrome(filecontent, filesize, 0) == true)
50      {
51        printf("Yes, %s is a palindrome\n", argv[1]);
52      }
53    else
54      {
55        printf("No, %s is not a palindrome\n", argv[1]);
56      }
57    free (filecontent);
58    return EXIT_SUCCESS;
59  }
```

5 Huffman Coding (12 points)

Hint: Consider what you learned for HW14.
Suppose the following is the output of the header of a file compressed using Huffman coding. This header specifies the Huffman tree using post-order traversal.

```
0000000: 10110011 11011011 11010111 00111001 00000010 11001011
0000006: 01101000 01011100 00101110 01000000
```

Suppose the following shows the data (only the beginning) using the Huffman tree expressed above.

```
0000000: 00011110 11111101 10011010 11001101 11001110
```

Write down the letters of this message.
Answer:

```
00 01 1110 1111 1101 100 1101 01 100 1101 1100 1110
g  o   p    r     h   s    h   o  s   h    e    p
```