ECE264 Fall 2016
Exam 2, 8-9PM, November 7
For Tuesday-Thursday Section

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.*

Please return only the top sheet.

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.

This exam tests one learning objective:
Recursion (Q1, Q3), Structure (Q2, Q3, Q4), Dynamic Structure (Q3, Q4).
You must obtain 50% or more points in one of the corresponding questions to pass the learning objective.

Contents

Total Score:

Learning Objective

Recursion (Q1, Q3)	Pass	Fail
Structure (Q2, Q3, Q4)	Pass	Fail
Dynamic Structure (Q3, Q4)	Pass	Fail

Q1	Q2	Q3
A val1 =	A	A
B val2 =	B	B
C val3 =	C	C
D val4 =	D	D
		E
		F

Q4
1
2
3
4
5
6

1 Recursion (20 points)

Write the outputs of this program.

```
#include <stdio.h>
#include <stdlib.h>
int func(int * arg1, int * arg2, int * arg3, int arg4, int arg5)
{
  (* arg1) ++;
  if ((* arg3) < arg4)
    {
      * arg3 =  arg4;
    }
  if (arg5 <= 1)
    {
      return 1;
    }
  (* arg2) ++;
  int sum = 0;
  int iter;
  for (iter = 1; iter <= arg5; iter ++)
    {
      sum += func(arg1, arg2, arg3, arg4 + 1, arg5 - iter);
    }
  return sum;
}

int main(int argc, char * * argv)
{
  int val1 = 0;
  int val2 = 0;
  int val3 = 0;
  int val4 = func(& val1, & val2, & val3, 1, 4);
  printf("val1 = %d, val2 = %d, val3 = %d, val4 = %d\n",
         val1, val2, val3, val4);
  return EXIT_SUCCESS;
}
```

2 Structure (20 points)

Consider the following structure.

```
#ifndef Q2_H
#define Q2_H
typedef struct
{
  int age;
  char * name;
} Person;
#endif
```

Please write the comparison functions for `qsort` to sort an array of `Person`'s by ages. This is an excerpt of `qsort`'s document.

```
       #include <stdlib.h>

       void qsort(void *base, size_t nmemb, size_t size,
                  int (*compar)(const void *, const void *));

DESCRIPTION

       The qsort() function sorts an array with nmemb elements of size
       size.  The base argument points to the start of the array.

       The contents of the array are sorted in ascending order
       according to a comparison function pointed to by compar, which
       is called with two arguments that point to the objects being
       compared.

       The comparison function must return an integer less than, equal
       to, or greater than zero if the first argument is considered to
       be respectively less than, equal to, or greater than the
       second.  If two members compare as equal, their order in the
       sorted array is unde fined.

       The qsort_r() function is identical to qsort() except that the
       comparison function compar takes a third argument.  A pointer
       is passed to the comparison function via arg.  In this way, the
       comparison function does not need to use global variables to
       pass through arbi trary arguments, and is therefore reentrant
       and safe to use in threads.
```

EXAMPLE

Another example is the following program, which sorts the strings given in its command-line arguments:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

static int
cmpstringp(const void *p1, const void *p2)
{
    /* The actual arguments to this function are "pointers to
       pointers to char", but strcmp(3) arguments are "pointers
       to char", hence the following cast plus dereference */

    return strcmp(* (char * const *) p1, * (char * const *) p2);
}
```

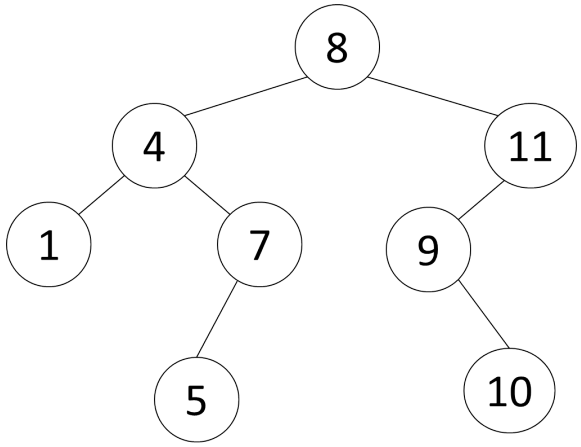
```
// comparison function
int comparebyAge(const void * p1,  const void * p2);

// create an array of 100 Person objects
Person * parray;
// -------------------------------
// Question A (choose one from the following 6 options)
// Option 1
parray = malloc(sizeof(int)  * 100);

// Option 2
parray = malloc(sizeof(parray)  * 100);

// Option 3
parray = malloc(sizeof(& parray)  * 100);

// Option 4
parray = malloc(sizeof(void *)  * 100);

// Option 5
parray = malloc(sizeof(* parray)  * 100);

// Qption 6
// None of the above
```

```

// fill in the elements of the array
// Assume the code here is correct

// sort by age
// -------------------------------
// Question B (choose one from the following 6 options)
// Option 1
qsort(parray, 100,  sizeof(Person), comparebyAge);

// Option 2
qsort(parray, 100,  sizeof(Person *), comparebyAge);

// Option 3
qsort(* parray, 100,  sizeof(Person), comparebyAge);

// Option 4
qsort(parray, 100,  sizeof(Person), * comparebyAge);

// Option 5
qsort(parray[0], sizeof(Person),  100, comparebyAge);

// Qption 6
// None of the above

// comparison function by age
int comparebyAge(const void * p1,
                 const void * p2)
{
  // -------------------------------
  // Question C (choose one from the following 6 options
  // Option 1
  const Person  * pp1 = (const Person  *) p1;
  const Person  * pp2 = (const Person  *) p2;

  // Option 2
  const Person * * pp1 = (const Person * *) p1;
  const Person * * pp2 = (const Person * *) p2;

  // Option 3
  const Person  pp1 = (const Person) p1;
  const Person  pp2 = (const Person) p2;
```

```

  // Option 4
  const Person * pp1 = (const Person * *) p1;
  const Person * pp2 = (const Person * *) p2;

  // Option 5
  const Person * * pp1 = (const Person) p1;
  const Person * * pp2 = (const Person) p2;

  // Qption 6
  // None of the above

  // -------------------------------
  // Question D (choose one from the following 6 options
  // Option 1
  const Person * pv1 = & pp1;
  const Person * pv2 = & pp2;

  // Option 2
  const Person * * pv1 = * pp1;
  const Person * * pv2 = * pp2;

  // Option 3
  const Person * pv1 = * pp1;
  const Person * pv2 = * pp2;

  // Option 4
  const Person pv1 = * pp1;
  const Person pv2 = * pp2;

  // Option 5
  const Person * * pv1 = & pp1;
  const Person * * pv2 = & pp2;

  // Qption 6
  // None of the above

  // compare the ages
  return // the difference of the ages
}
```

3 Binary Search Tree (30 points)

Consider the following graphical representation of a binary search tree.



The following is the definition of a tree node.

```
// tree.h
#ifndef TREE_H
#define TREE_H
#include <stdio.h>
typedef struct treenode
{
  struct treenode * left;
  struct treenode * right;
  int value;
} TreeNode;
#endif
```

The following table shows the memory

address	left	right	value
0x010	0x030	0x0b0	8
0x030	0x070	0x050	4
0x050	0x090	NULL	7
0x070	NULL	NULL	1
0x090	NULL	NULL	5
0x0b0	0x0d0	NULL	11
0x0d0	NULL	0x0f0	9
0x0f0	NULL	NULL	10

The following is a function that intends to delete a node whose value is the same as the second argument. The function has a mistake.

```
// treedelete.c
#include "tree.h"
#include <stdlib.h>
TreeNode * Tree_delete(TreeNode * tn, int val)
{
  if (tn == NULL) { return NULL; }
  if (val < (tn -> value))
    {
      tn -> left = Tree_delete(tn -> left, val);
      return tn;
    }
  if (val > (tn -> value))
    {
      tn -> right = Tree_delete(tn -> right, val);
      return tn;
    }
  // v is the same as tn ->  value
  if (((tn ->  left) == NULL) && ((tn ->  right) == NULL))
    {
      // tn has no child
      free (tn);
      return NULL;
    }
  if ((tn ->  left) == NULL)
```

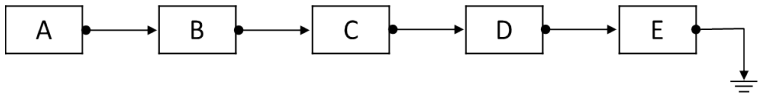
```
    {
      // tn ->  right must not be NULL
      TreeNode * rc = tn ->  right;
      free (tn);
      return rc;
    }
  if ((tn ->  right) == NULL)
    {
      // tn ->  left must not be NULL
      TreeNode * lc = tn ->  left;
      free (tn);
      return lc;
    }
  // tn have two children
  // find the immediate successor

  // the following is the correct code
  // TreeNode * su = tn ->  right;
  // ---->>> BUG <<<---
  // ---->>> BUG <<<---
  // the following is incorrect but actually used
  TreeNode * su = tn;

  while ((su -> left) != NULL)
    {
      su = su -> left;
    }
  tn ->  value = su -> value;
  su -> value = val;
  tn ->  right = Tree_delete(tn ->  right, val);
  return tn;
}
```

DO NOT CORRECT THE PROGRAM. The correct program is already given to you. Please fill the table after calling this function. "-" means the value in this cell is not tested in this exam.

address	left	right	value
0x010	A	B	C
0x030	-	-	-
0x050	-	-	D
0x070	-	-	E
0x090	F	-	-
0x0b0	-	-	-
0x0d0	-	-	-
0x0f0	-	-	-

4 Linked List (30 points)

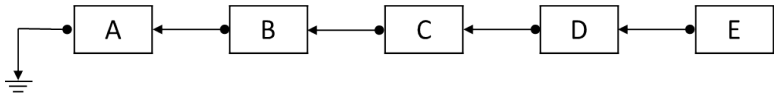
Consider the following graphical representation of a linked list. The list's head points to `A`.



The following table shows the memory

address	value	next
0x010	E	NULL
0x030	D	0x010
0x050	C	0x030
0x070	B	0x050
0x090	A	0x070

Also, the value of `head` is 0x090.
The reverse of the linked is shown below. The list's head points to `E`.



The following is an incorrect reverse function. **DO NOT CORRECT THE PROGRAM.**

```
#include "list.h"
Node * List_reverse(Node * head)
{
  if (head == NULL)
    {
      // empty list, nothing to do
      return NULL;
    }
  Node * orighead = head;
  Node * revhead = NULL; // must initialize to NULL
  Node * origsec; // will be assigned before using
  while (orighead != NULL)
    {
      origsec = orighead -> next;
      orighead -> next = revhead;
      orighead = origsec;
      revhead = orighead;
    }
  return revhead;
}
```

Fill the following table. "-" means the value in this cell is not tested in this exam.
The value of `head` is now ???. (This is answer 1.)

address	value	next
0x010	-	2
0x030	-	3
0x050	-	4
0x070	-	-
0x090	5	6