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

1. Recursion (20 points)  3
2. Structure (20 points)  4
3. Binary Search Tree (30 points)  8
4. Linked List (30 points)  12

## Total Score:

### Learning Objective

<table>
<thead>
<tr>
<th></th>
<th>Pass</th>
<th>Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recursion (Q1, Q3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structure (Q2, Q3, Q4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic Structure (Q3, Q4)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A val1 =</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>B val2 =</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>C val3 =</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>D val4 =</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>
1 Recursion (20 points)

Write the outputs of this program.

```c
#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.

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

```c
#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);

// Option 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);

// Option 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;

// Option 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;

// Option 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.

```c
// 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
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.

```c
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         { tn -> left = Tree_delete(tn -> left, val); return tn; }
9     if (val > (tn -> value))
10         { tn -> right = Tree_delete(tn -> right, val); return tn; }
11     // v is the same as tn -> value
12     if (((tn -> left) == NULL) && ((tn -> right) == NULL))
13         { // tn has no child
14             free (tn);
15             return NULL;
16         }
17     if ((tn -> left) == NULL)
18         {...
19     
20         }
21     
22     if ((tn -> left) == NULL)
23         {...
24     
25     }
```
```c
{ 
    // 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
// 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.

```
<table>
<thead>
<tr>
<th>address</th>
<th>left</th>
<th>right</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x010</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>0x030</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x050</td>
<td>-</td>
<td>-</td>
<td>D</td>
</tr>
<tr>
<td>0x070</td>
<td>-</td>
<td>-</td>
<td>E</td>
</tr>
<tr>
<td>0x090</td>
<td>F</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x0b0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x0d0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x0f0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>address</th>
<th>value</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x010</td>
<td>E</td>
<td>NULL</td>
</tr>
<tr>
<td>0x030</td>
<td>D</td>
<td>0x010</td>
</tr>
<tr>
<td>0x050</td>
<td>C</td>
<td>0x030</td>
</tr>
<tr>
<td>0x070</td>
<td>B</td>
<td>0x050</td>
</tr>
<tr>
<td>0x090</td>
<td>A</td>
<td>0x070</td>
</tr>
</tbody>
</table>

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.**

```c
#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.)

<table>
<thead>
<tr>
<th>address</th>
<th>value</th>
<th>next</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x010</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>0x030</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td>0x050</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td>0x070</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>0x090</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>