# ECE264 Summer 2013
# Exam 3, July 18, 2013

**Name:**

By 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 also declare that I will not discuss/share this exam with anybody today. 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* **2-point** *penalty.*

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.

Two learning objectives (structure and dynamic structure) are tested in this exam. To pass an objective, you must receive 50% or more points in the corresponding question. If you have passed a learning objective, you will not "unpass" later.

## Contents

| | | |
|---|---|---|
| Learning Objective 3 (Structure) | Pass | Fail |
| Learning Objective 4 (Dynamic Structure) | Pass | Fail |
| Total | | |

## 1 Special Array (8 points, Structure)

This question asks you to create an array with the following special properties:

1. Each element is an integer
2. All elements are initialized to zero.
3. Only **positive** integers can be inserted into the array.
   If a negative integer or zero is inserted into the array, the array is not changed.
4. When a positive integer $v_{new}$ is inserted, follow this procedure:
   (a) Set $k$ to zero.
   (b) Repeat the following steps:
   (c) If $k$ is an invalid index of the array, stop, i.e., the insert function should not continue.
   (d) Set $v_{compare}$ to the value at index $k$.
   (e) Compare $v_{new}$ with $v_{compare}$. If $v_{compare}$ is zero, change $v_{compare}$ to $v_{new}$. Stop.
   (f) If $v_{new} = v_{compare}$, do not change the array. Stop.
   (g) If $v_{new} < v_{compare}$, set the new value of $k$ to $1 + 2 \times$ old value of k.
   (h) If $v_{new} > v_{compare}$, set the new value of $k$ to $2 + 2 \times$ old value of k.

---

Consider the following command sequence.

1. Create a `SpecialArray` object with space for 20 elements.
2. Insert 7.
3. Insert 3.
4. Insert 12.
5. Insert 5.
6. Insert 1.
7. Insert 9.
8. Insert 8.
9. Insert 4.
10. Insert -3.
11. Insert 5.

Write down the values of the array (2 points).

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| value | | | | | | | | | | |
| index | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| value | | | | | | | | | | |

Searching whether a value is stored in the array can be very efficient. You need to design and implement an efficient solution **without** checking every element in the array.
In the following program, the array's size is given by the input argument, **not** 20. You will lose 2 points if you hard-code the array's size to 20.

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

typedef struct
{
  int size;
  int *data;
} SpecialArray;

/*
  Create and return an object of SpecialArray.

  Allocate memory for the object and the data (0.25 point)

  The data attribute should be able to hold enough integers (0.25
  point)

  The data's size is given by the input argument (0.25 point)

  Initialize all elements to zero (0.25 point)
*/

SpecialArray *SpecialArray_create(int size)
{















}

/*
   Insert an integer into the array. Follow the rules described
   earlier (2 points)

   If the array is changed, return 1.  If the array is not changed,
   return 0. (0.5 point)
*/

int SpecialArray_insert(SpecialArray *arr, int value)
{
  /* set k to zero */



  /* check whether value is positive. do not insert a negative value
     or zero */







  /* check whether the index is still valid */




  /* If the element at index k is zero, set it to value */






  /* If the element at index k is the same as value, do nothing */





  /* change the value of k */















}

/*
  Search whether a value is stored in the special array.
  If the value is zero or negative, return -1.
  If the value is in the array, return the index.
  If the value is not in the array, return -1.

  This function MUST NOT check the elements one by one in the
  array. More specially, this function must determine whether the
  value is in the array using *at most* log(size) comparisons, here
  size is the array's size.

  You will receive no point if the array's index increments by
  one each time.

  2 points
*/

int SpecialArray_contains(SpecialArray *arr, int value)
{




}


/* release memory used by the special array 0.5 point */

void SpecialArray_destroy(SpecialArray * arr)
{






}
```

# 2 Linked List (9 points, Dynamic Structure)

This question asks you to implement a linked list with the following property: If an odd integer is inserted, it is inserted at the beginning of the list. If an even integer is inserted, it is inserted at the end of the list.

## 2.1 Program with Bug (6 points)

The following program intends to implement the `List_insert` function. It contains one (or several) bug. If there are multiple lines that can cause problems, point out the **first** problem encountered when running the program.

Do not worry about any bug in the other functions.

1. What is the output of this program? (0.5 point) It is possible that the program prints nothing before the problem occurs. In this case, your answer is “nothing”.
   If the program has “Segmentation Fault”, points out which line causes the problem.
2. Explain the reason (1 point).
3. Draw the linked list (`next` and `value`) **when** the problem occurs. Do not draw the call stack and heap. (2 points)
4. Is **all** memory allocated by calling `malloc` still reachable **when** the problem occurs? Explain your answer. (2 points)

Your answer should be clear, precise, and **short**.

Do not fix this program. You will receive no point by fixing the program.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  typedef struct listnode
4  {
5    struct listnode * next;
6    int value;
7  } Node;
8
9  Node * Node_construct(int v)
10 {
11   Node * n = malloc(sizeof(Node));
12   n -> value = v;
13   n -> next = NULL;
14   return n;
15 }
16
17 /*
18    insert a value v to a linked list.
19    head: the head of the original list.
20    return the head of the new list.
```

```
*/
Node * List_insert(Node * head, int v)
{
  Node * p = Node_construct(v);
  if (head == NULL)
    {
      return p;
    }
  if ((v % 2) == 1) /* odd number */
    {
      /* insert at the beginning */
      p -> next = head;
    }
  Node * q = head;
  /* insert at the end */
  while ((q -> next) != NULL)
    {
      q = q -> next;
    }
  q -> next = p;
  return head;
}

/* delete every node */
void List_destroy(Node * head)
{
  while (head != NULL)
    {
      Node * p = head -> next;
      free (head);
      head = p;
    }
}

void List_print(Node * head)
{
  printf("\nPrint the whole list:\n");
  while (head != NULL)
    {
      printf("%d ", head -> value);
      head = head -> next;
    }
  printf("\n\n");
}

int main(int argc, char * argv[])
{
  Node * head = NULL; /* must initialize it to NULL */
  head = List_insert(head, 917);
  head = List_insert(head, -504);
  head = List_insert(head, 263);
  head = List_insert(head, 326);
  head = List_insert(head, 138);
  head = List_insert(head, -64);
  List_print(head);
  List_destroy(head);
  return EXIT_SUCCESS;
}
```

## 2.2 Sparse Array (3 points)

The fourth programming assignment asks you to implement a sparse array using a binary search tree. This problem asks you to debug the merge function for a sparse array using a linked list. **Give a scenario** (properties of `arr1` and `arr2`) when this program will cause "Segmentation Fault" and briefly explain why this occurs. If there are multiple bugs, you need to find only one.
Assume that both `arr1` and `arr2` are valid spare arrays. Each can be `NULL`, if the array has no element.

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

typedef struct snode {
    int index;
    int value;
    struct snode *next;
} SparseNode;

SparseNode *SparseArray_insert(SparseNode *arr,
                               int index, int value);
SparseNode *SparseArray_removeZero(SparseNode *arr);

SparseNode *SparseNode_create(int index, int value)
{
  SparseNode * sp = malloc(sizeof(SparseNode));
  sp -> index = index;
  sp -> value = value;
  sp -> next  = NULL;
  return sp;
}
// create a new sparse array that has the nodes from arr1 and arr2
// copy the nodes from arr1, create a new array called arr3
//
// add the nodes from arr2 to arr3, if two nodes have the same index,
// add the values together
//
// remove the nodes whose values are zero
//
// If arr1 is NULL, this function returns a copy of arr2.
// If arr2 is NULL, this function returns a copy of arr1.
// If both arr1 and arr2 are NULL, this function returns NULL.

SparseNode *SparseArray_merge(SparseNode *arr1, SparseNode *arr2)
{
  SparseNode * arr3 = NULL;
  // copy arr1 to a new array called arr3
  while (arr1 != NULL)
    {
      arr3 = SparseArray_insert(arr3, arr1 -> index, arr1 -> value);
      arr1 = arr1 -> next;
    }
  // insert the nodes in arr2 into the end of arr3
  while (arr2 != NULL)
    {
      SparseNode * arr4 = arr3;
      int found = 0;
      while ((found == 0) && ((arr4 -> next) != NULL))
        {
          if ((arr4 -> index) == (arr2 -> index))
            {
              arr4 -> value += arr2 -> index;
              found = 1;
            }
          else
            {
              arr4 = arr4 -> next;
            }
        }
      if (found == 0)
        {
          // arr4 is the last node of arr3
          // add a new node at the end
          arr4 -> next =
            SparseNode_create(arr2 -> index, arr3 -> value);
        }
      arr2 = arr2 -> next;
    }
  // remove the nodes whose values are zero. assume the removeZero
  // function has been correctly implemented
  arr3 = SparseArray_removeZero(arr3);
  return arr3;
}
```

## 3 Binary Search Tree (3 points, Dynamic Structure)

### 3.1 Insertion (1 point)

Draw the binary search tree after inserting
7, 3, 6, 2, 9, 8, 13, 1, 12, 10

### 3.2 Program Result (2 points)

Consider the following function for `Tree_delete`. There is a (or several) mistake in the following program.

```
#include "tree.h"
#include <stdlib.h>
/*
  Delete the node whose value is val.
  root: the root of the tree
  val: the value to search

  If val is not stored in the tree, this function does not delete
  anything.

  The function returns the root of the tree after deleteing the node.
*/
Tree * Tree_delete(Tree * root, int val)
{
  if (root == NULL)
    {
      return NULL;
    }
  if (val < (root -> value))
    {
      root -> left = Tree_delete(root -> left, val);
      return root;
    }
  if (val > (root -> value))
    {
      root -> right = Tree_delete(root -> right, val);
      return root;
    }
  /* root's value is the the same as val, needs to delete root */
  if (((root ->  left) == NULL) && ((root ->  right) == NULL))
    {
      /* root has no child */
      free (root);
      return NULL;
    }
  if ((root ->  left) == NULL)
    {
      Tree * rc = root ->  right;
      free (root);
      return rc;
    }
  if ((root ->  right) == NULL)
    {
      Tree * lc = root ->  left;
      free (root);
      return lc;
    }
  /* root have two children */

  /* There is a (or several) mistake in the following lines */
  /* ================================================ */
  /* BELOW THIS LINE */

  /* find the immediate successor */
  Tree * su = root ->  right; /* su must not be NULL */

  while ((su -> left) != NULL)
    {
      su = su -> left;
    }
  /* su is root's immediate successor */
  /* swap their values */
  root ->  value = su -> value;
  su -> value = val;
  /* delete the successor */
  root ->  right = Tree_delete(root, val);
  /* ABOVE THIS LINE */
  /* ================================================ */
  return root;
}
```

Use the root of the binary search tree created from the previous question as the first argument. The second argument is 7.
Draw the binary tree returned by this function. (1 point)
Briefly explain why the tree is the way you draw. (1 point)
This question does **not** ask you to correct the mistake (because you can find the correction from the course note). Do not explain how to correct the program. You will not receive any point by correcting this program.