ECE 264-1 Exam 3

01:30-02:20AM, April 09, 2010

I certify that I will not receive nor provide aid to any other student for this exam.

Signature:

You must sign here. Otherwise, you will receive **1-point** *penalty.*

This exam is printed **double sides**. Please read the questions carefully. Two common mistakes are answering wrong questions and failing to answer all questions.

This is an *open-book, open-note* exam. You can use any book or note or program printouts.

Please turn off your cellular phone and iPod. No electronic device is allowed.

Outcome 3 is evaluated in one question. To pass the outcome, you need to obtain 50% or more of the total score in the question.

Contents

Total Score: out of 20.

Passed Outcome 3: Yes No

1 Deep Copy (4 points)

Implement deep copy of the structure.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef struct
{
  int size;
  double * value;
} Vector;

/* write the constructor (1 point) */
/* The created Vector object has an array of s elements.
   The elements' values are initialized by the array v. */
Vector * Vector_construct(double * v, int s)
{
  Vector * vec;









  return vec;
}
/* release memory (1 point) */
void Vector_destruct(Vector * vec)
{










}
/* copy constructor */
Vector * Vector_copy(Vector * vec)
{
  return Vector_construct(vec -> value, vec -> size);
}
/* assignment (1 points) */
void Vector_assign(Vector ** dest, Vector * src)
{
















}
void Vector_print(Vector * vec)
{
  int i;
  printf("Vector with %d elements: \n", vec -> size);
  for (i = 0; i < (vec -> size); i ++)
    {
      printf("\t%f\n", vec -> value[i]);
    }
}
int main(int argc, char * argv[])
{
  double data[] = {2.6, 4.1, -3.2, 7.6, 9.8, 34.5, 5.7};
  int size = sizeof(data) / sizeof(double);
  Vector * v1 = Vector_construct(data, size);
  Vector * v2 = Vector_copy(v1);
  Vector_print(v2);
  v2 -> value[2] = 2010;
  Vector_print(v1);
  Vector_print(v2);
  /* assign v1 to v2, how to use Vector_assign? (0.5 point) */
  /* v1 is the source and v2 is the destination */






  v2 -> value[2] = 218;
  Vector_print(v1);
  Vector_print(v2);
  /* assign v2 to v2 (itself), should not change v2 (0.5 point) */




  Vector_destruct(v1);
  Vector_destruct(v2);
  return 0;
}
```

2 Recursion (4 points)

The *Ackermann function* is defined as

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0, \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0, \\ A(m-1, A(m, n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases} \tag{1}$$

Write a C function to implement the Ackermann function using recursion.

3 Linked List (outcome 3, 12 points)

Write the code for a linked list. If the inserted value is an even number, the number is inserted at the beginning. If the inserted number is odd, it is inserted at the end. Do **not** check whether the value has already been inserted earlier.

```
#include <stdio.h>
#include <stdlib.h>
/* create a structure called Node (1 point)
   Each Node has two fields:
      an integer called "value"
      a pointer to another Node called "next" */
```

```
Node * Node_construct(int v)
{
  /* constructor for Node (1 point) */
```

```
}
void Node_destruct(Node * n)
{
  /* destructor for Node (1 point) */
```

```
}
void List_insert(Node ** h, int v)
{
  /* create a new Node called n and its value is v (1 point) */
  /* If the list is currently empty, make n the first Node (1 point) */





  /* If v is an even number, make the new Node the first Node (1 point) */






  /* If v is an odd number, find the last Node in the current list
     (1 point) */




  /* make the new Node the last Node in the list (1 point) */




}
void List_delete(Node * * h, int v)
{
  /* delete the first occurrence of the value v */
  /* If the list has no Node of value v, do nothing */
  /* Caution: the first Node’s value may be v */
  if ((*h) == 0)
    {
      /* list is empty, do nothing */
      return;
    }
  Node * p = * h;
  /* check whether the head’s value is v */
  if ((p -> value) == v)
    {
      /* if it is v, move the head to the next Node and delete the
         original head (1 point) */






      return;
    }
  /* Otherwise, find the first Node whose value is v (1 point) */









  /* If no Node whose value is v, do nothing. If a Node is found,
     delete it from the list (1 point) */











  /* If a Node is found, destroy the Node and release memory (1 point) */
```

```
}
void List_destruct(Node * * h)
{
  /* delete every Node in the list */
  Node * p = * h;
  Node * q;
  while (p != 0)
    {
      q = p -> next;
      Node_destruct(p);
      p = q;
    }
}
void Node_print(Node * n)
{ printf("%d ", n -> value); }
void List_print(Node * n)
{
  Node * p = n;
  while (p != 0)
    {
      Node_print(p);
      p = p -> next;
    }
  printf("\n\n");
}
int main(int argc, char * argv[])
{
  /* You do not need to modify the main function */
  int x[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 5, 7, 4, 2, 0};
  int n = sizeof(x) / sizeof(int);
  int i;
  Node * head = 0;
  for (i = 0; i < n; i ++)
    { List_insert(& head, x[i]); }
  List_print(head);
  List_delete(& head, 4);
  List_print(head);
  List_destruct(& head);
  return 0;
}
```

Evaluation of Automatic Grading

This is **not** part of the exam. Please fill this survey **after** the exam. If you return it before April 23, you will receive one evaluation point.

For each of the programming assignments (IPA1 - IPA3), how many times did you submit to the grading server?

none   once   twice   three times   many times

If your answer is “none” or “once”, do you know that you could have submitted many times?

Yes   No

If your answer is “Yes”, why did you not take advantage of this option?

If your answer is “No”, what can be done to help you know this option?

If your answer is “twice” or more, do you find this option helpful?

Yes   No

If your answer is “No”, can you suggest how to make it more helpful?

Do your scores improve from earlier submissions to later submissions?

Yes   No

Please suggest how to improve the grading process. Thank you.