

ECE 264 Exam 3

01:30-02:25PM, April 03, 2009

1 Recursion (3.1 points)

1.1 Recursive Function (1.0 point)

Write a recursive function to implement

$$f(n, k) = \begin{cases} 0 & \text{if } n < k, \\ 1 & \text{if } n \geq k \text{ and } k = 0, \\ n & \text{if } n \geq k \text{ and } k = 1, \\ 1 & \text{if } n = k \text{ and } k \neq 0, \\ f(n - 1, k - 1) + f(n - 1, k) & \text{otherwise} \end{cases} \quad (1)$$

1.2 Non-Recursive Function (2.1 points)

Remove recursion from the function. The function **cannot** call itself. For each pair of (n, k) , the value of $f(n, k)$ can be calculated only **once**. In other words, if you use a 2-D array to store the values, $f[n][k]$ can be on the left side of assignment “=” only once.

Answer:

```
#include <stdlib.h>
#include <stdio.h>
int binomial(int n, int k)
{
    /* use recursion */
    if (n < k) { return 0; }
    if (k == 0) { return 1; }
    if (k == 1) { return n; }
    if (n == k) { return 1; }
    return (binomial(n - 1, k - 1) + binomial(n - 1, k));
}

int binomial2(int n, int k)
{
    /* do not use recursion */
```

```

int * * bino;
int ncnt;
int kcnt;
int result = 0;
printf("compute (%d, %d)\n", n, k);
if (n < k) { return 0; }
if (k == 0) { return 1; }
bino = malloc((n + 1) * sizeof (int));
// + 1 because the index is up to n
for (ncnt = 0; ncnt <= n; ncnt++)
{
    bino[ncnt] = malloc((k + 1) * sizeof(int));
    for (kcnt = 0; kcnt <= k; kcnt++)
    {
        bino[ncnt][kcnt] = -1; /* initialize to garbage */
    }
    printf("A: update [%d][0]\n", ncnt);
    bino[ncnt][0] = 1; /* k == 0 */
}
/* if k == 0, this condition will cause problems */
for (ncnt = 0; ncnt <= n; ncnt++)
{
    bino[ncnt][1] = ncnt; /* k == 1 */
    printf("B: update [%d][1]\n", ncnt);
}
for (kcnt = 2; (kcnt <= n) && (kcnt <= k); kcnt++)
{
    bino[kcnt][kcnt] = 1; /* n == k */
    printf("C: update [%d][%d]\n", kcnt, kcnt);
}
/* pay special attention to the indexes */
for (ncnt = 1; ncnt <= n; ncnt++)
{
    for (kcnt = 2; (kcnt < ncnt) && (kcnt <= k); kcnt++)
    {
        bino[ncnt][kcnt] = bino[ncnt - 1][kcnt - 1] +
            bino[ncnt - 1][kcnt];
        printf("D: update [%d][%d]\n", ncnt, kcnt);
    }
}
result = bino[n][k];
for (ncnt = 0; ncnt <= k; ncnt++)
{
    free(bino[ncnt]);
}

```

```

    free (bino);
    return result;
}

/* main function not necessary for the exam */
int main(int argc, char * argv[])
{
    int maxN = 15;
    int ncnt;
    int kcnt;
    for (ncnt = 1; ncnt <= maxN; ncnt++)
    {
        for (kcnt = 0; kcnt <= ncnt; kcnt++)
        {
            int b1 = binomial(ncnt, kcnt);
            int b2 = binomial2(ncnt, kcnt);
            printf("binomial(%2d, %2d) = %8d, %8d, diff = %d\n\n",
                   ncnt, kcnt, b1, b2, b1 - b2);
        }
        printf("\n");
    }
    return 0;
}

```

2 Structure and Stack Memory (1.5 points)

What are the outputs of the following code? **Justify your answers.** Assume the `Test_` functions are called by `main`.

2.1 Vector (0.5 point)

```

#ifndef VECTOR_H
#define VECTOR_H
typedef struct
{
    int x;
    int y;
} Vector;
Vector Vector_construct(int a, int b);
void Vector_print(Vector v);
void Test_Vector();
#endif

```

```

#include "vector.h"
#include <stdio.h>
Vector Vector_construct(int a, int b)
{
    Vector v;
    v.x = a;
    v.y = b;
    Vector_print(v);
    return v;
}

void Vector_print(Vector v)
{
    printf("( %d, %d )\n", v.x, v.y);
}

void Test_Vector()
{
    Vector v1 = Vector_construct(2, 6);
    Vector_print(v1);
}

```

2.2 Person (0.5 point)

```

#ifndef PERSON_H
#define PERSON_H
typedef struct
{
    int age;
    char name[60];
} Person;
Person Person_construct(int a, char * n);
void Person_print(Person p);
void Test_Person();
#endif

#include "person.h"
#include <stdio.h>
#include <string.h>
Person Person_construct(int a, char * n)
{
    Person p;
    p.age = a;
    strcpy(p.name, n);
    Person_print(p);
    return p;
}

```

```

}

void Person_print(Person p)
{
    printf("(%d,%s)\n", p.age, p.name);
}

void Test_Person()
{
    Person p2 = Person_construct(18, "Amy Smith");
    Person_print(p2);
}

```

2.3 Student (0.5 point)

```

#ifndef STUDENT_H
#define STUDENT_H
typedef struct
{
    int year;
    char * school;
    char * department;
} Student;
Student Student_construct(int y, char * s, char * d);
void Student_print(Student s);
void Test_Student();
#endif

#include "student.h"
#include <stdio.h>
#include <string.h>
Student Student_construct(int y, char * s, char * d)
{
    Student st;
    char sch[60];
    char dept[60];
    st.year = y;
    st.school = sch;
    st.department = dept;
    strcpy(st.school, s);
    strcpy(st.department, d);
    Student_print(st);
    return st;
}

void Student_print(Student s)

```

```

{
    printf( "(%d,%s,%s)\n", s.year, s.school, s.department );
}

void Test_Student()
{
    Student s3 = Student_construct(2, "Purdue", "ECE");
    Student_print(s3);
}

```

Answer:

(2,6)
(2,6)
(18,Amy Smith)
(18,Amy Smith)
(2,Purdue,ECE)
(2,garbage)

If a structure contains only primitive types (such as `int`) or fixed size arrays (such as `char name[60]`), compilers can allocate the correct amounts of space. These are the cases for the first four answers.

For the fifth answer, there is space inside the `Student_construct` function so the strings are copied correctly. When the function returns, the local arrays inside the function are popped so the space is no longer valid.

3 Dynamic Structure (Outcome 3, 5.4 points)

Some code is provided to you at the end of this question. You do not have to write the functions that are already available to you.

3.1 Linked List Node (0.4 point)

Create a structure called `Node` for a node in linked lists. This structure has (1) an attribute to store one integer (2) another attribute to store a link to the next node.

3.2 Insertion (2.5 points)

Write an insertion function called `List_insert`

```
void List_insert(Node * * n, int v)
```

- * n: pointing to the first node in the linked list
- v: the value to be inserted into the list

The values in the linked list, **starting from the first node**, follow these rules:

- All even numbers appear before all odd numbers.
- Odd numbers appear in the **same** order as they are inserted (i.e. a queue).
- Even numbers appear in the **reverse** order as they are inserted (i.e. a stack).
- The same values **may** appear in multiple nodes.

3.3 Deletion (1.5 points)

Write a deletion function called `List_delete`

```
int List_delete(Node * * n, int v)
```

- * n: pointing to the first node in the linked list
- v: the value to be deleted from the list

- return value: 1 if a node has value v and this node is deleted; 0 if no node has value v.

From the beginning of the list, the function finds and deletes the **first** node whose value is the same as v. If no node has value v, the function does not change the list. This function deletes **at most one** node and does **not** change the order of the remaining nodes.

3.4 Multiple Deletions (1.0 point)

Use `List_delete` to write a function `List_deleteAll`

```
int List_deleteAll(Node * * n, int v)
```

- * n: pointing to the first node in the linked list
- v: the value to be deleted from the list
- return value: 1 if at least one node is deleted; 0 if no node is deleted.

The function deletes **all** nodes whose values are the same as v. If no node has value v, the function does not change the list. This function does **not** change the order of the remaining nodes.

The following is an example showing how to use the functions.

```

#ifndef QUESTION3_H
#define QUESTION3_H
#include "answer3.h"
Node * Node_construct(int v);
void Node_destruct(Node * n);
void List_destruct(Node * n);
void Node_print(Node * n);
void List_print(Node * n);
int isOdd(Node * n);
void List_insert(Node ** n, int v);
int List_delete(Node ** n, int v);
int List_deleteAll(Node ** n, int v);
#endif

#include <stdio.h>
#include <stdlib.h>
#include "question3.h"

Node * Node_construct(int v)
{
    Node * n = malloc(sizeof(Node));
    n -> value = v;
    n -> next = 0;
    return n;
}

void Node_destruct(Node * n)
{
    free (n);
}

void List_destruct(Node * n)
{
    Node * prev = n;
    Node * next;
    while (prev != 0)
    {
        next = prev -> next;
        Node_destruct(prev);
        prev = next;
    }
}

void Node_print(Node * n)
{
    printf("%d ", n -> value);
}

```

```

}

void List_print(Node * n)
{
    Node * curr = n;
    while (curr != 0)
    {
        Node_print(curr);
        curr = curr -> next;
    }
    printf("\n\n");
}

int isOdd(Node * n)
{
    return ((n -> value) % 2);
}

int main(int argc, char * argv[])
{
    int data1[] = {1, 2, 5, 6, 8, 10, 9, 2, 3, 5, 6, 7, 8, 3, 2, 4};
    int data2[] = {2, 4, 6, 1, 6, 8, 10, 9, 5, 2, 3, 5, 6, 7, 8, 3, 2, 4};
    int cnt;
    Node * list = 0;
    for (cnt = 0; cnt < sizeof(data1)/sizeof(int); cnt++)
    {
        List_insert(& list, data1[cnt]);
    }
    List_print(list);
    List_destruct(list);
    list = 0;
    for (cnt = 0; cnt < sizeof(data2)/sizeof(int); cnt++)
    {
        List_insert(& list, data2[cnt]);
    }
    List_print(list);

    printf("List_delete(&list, 5) = %d\n", List_delete(&list, 5));
    List_print(list);
    printf("List_delete(&list, 0) = %d\n", List_delete(&list, 0));
    List_print(list);
    printf("List_delete(&list, 4) = %d\n", List_delete(&list, 4));
    List_print(list);

    printf("List_deleteAll(&list, 2) = %d\n", List_deleteAll(&list, 2));
}

```

```

List_print(list);
printf("List_deleteAll(&list, 2) = %d\n", List_deleteAll(&list, 2));
List_print(list);
printf("List_deleteAll(&list, 3) = %d\n", List_deleteAll(&list, 3));
List_print(list);

List_destruct(list);
return 0;
}

```

Answer:

```

#ifndef ANSWER3_H
#define ANSWER3_H
typedef struct listnode
{
    struct listnode * next;
    int value;
} Node;
#endif

#include "question3.h"
void List_insert(Node ** n, int v)
{
    Node * p;
    Node * prev;
    Node * curr;
    p = Node_construct(v);
    if (*n == 0) /* original list is empty */
    {
        *n = p;
        return;
    }
    if (isOdd(p) == 0) /* v is even */
    {
        p -> next = *n;
        *n = p;
        return;
    }
    /* find the end of the list */
    prev = *n;
    curr = *n;
    while (curr != 0)
    {
        prev = curr;
        curr = curr -> next;

```

```

        }
    prev -> next = p;
}

int List_delete(Node ** n, int v)
{
    Node * prev;
    Node * curr;
    if (*n == 0) /* empty list, nothing to delete */
        { return 0; }
    curr = *n;
    if ((curr -> value) == v)
    {
        /* delete the first node */
        *n = curr -> next;
        Node_destruct(curr);
        return 1;
    }
    prev = *n;
    curr = *n;
    while ((curr != 0) && ((curr -> value) != v))
    {
        prev = curr;
        curr = curr -> next;
    }
    if (curr == 0) /* v is not in the list */
    {
        return 0;
    }
    prev -> next = curr -> next;
    Node_destruct(curr);
    return 1;
}

int List_deleteAll(Node ** n, int v)
{
    int deleteAny = 0;
    int deleteCount = 0;
    do
    {
        deleteAny = List_delete(n, v);
        deleteCount += deleteAny;
    } while (deleteAny == 1);
    return (deleteCount > 0);
}

```