CMPS 12B  
Introduction to Data Structures  
Winter 2019  
Midterm Exam 2

Solutions

1. (20 Points) Write a recursive Java function called `sumList()` which, given a reference to the head of a linked list based on the Node class defined below, returns the sum of the items in the list. The sum of an empty list is defined to be zero.

```java
class Node{
    int item;
    Node next;
    Node(int x){
        item = x;
        next = null;
    }
}

// In some class in the same directory as Node:

static int sumList(Node H){
    // your code begins here
    if( H==null ){
        return 0;
    }else{
        return H.item + sumList(H.next);
    }
    // your code ends here
}
```
2. (20 Points) The following C program includes a global variable called `time`. Since it is declared outside of all functions (on line 3), its scope is the entire file. Notice `time` is incremented before each of the functions `f`, `g`, and `h` return. Show the state of the function call stack when `time=5` (i.e. at the instant `time` becomes equal to 5). Draw the stack horizontally, with top on the left and bottom on the right. Each stack frame should show the values of all function arguments and local variables, and the line to which execution will transfer when the function returns. If a local variable has not yet been assigned a value at `time=5`, indicate that by stating its value as `undef`. Also determine the program output, and print it on the line below exactly as it would appear on the screen.

```c
1) #include<stdio.h>
2) #include<stdlib.h>
3) int time;
4) int f(int x){
5)     int i;
6)     i = x*x+1;
7)     time++;
8)     return(i);
9) }
10) int g(int y){
11)     int j;
12)     j = y+f(y);
13)     time++;
14)     return(j);
15) }
16) int h(int z){
17)     int k;
18)     k = f(z)+g(z); // first call f(z), then call g(z)
19)     time++;
20)     return(k);
21) }
22) int main(void){
23)     int a, b, c;
24)     time = 0;
25)     a = f(3);
26)     b = g(a);
27)     c = h(4);
28)     printf("a=%d, b=%d, c=%d, time=%d\n", a, b, c, time);
29)     return(EXIT_SUCCESS);
30) }
```

Program Output:

```
a=10, b=111, c=38, time=7
```

State of the function call stack when `time=5`:

```
Top
```

```
bottom
```

```c
f(): 12
x = 4
i = 17
```

```c
g(): 18
y = 4
j = undef
```

```c
h(): 27
z = 4
k = undef
```

```c
main(): OS
a = 10
b = 111
c = undef
```
3. (20 Points) Consider the following C program.

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

int main(void)
{
    int i, j;
    double x = 1.2, y;
    double * A = calloc(4, sizeof(double));
    double B[] = {3.4, 5.6, 7.8, 9.0};
    double *p, *q;
    p = malloc(sizeof(double));
    y = x+2;
    q = &y;
    *p = *q + 2.5;
    for(i=0; i<4; i++){
        j = 3-i;
        *(A+i) = B[j] + i;
    }
    p = &x;
    printf("%f, %f, %f, %f\n", *A, *B, *p, *q);
    printf("%f, %f, %f, %f\n", *A, *(A+1), *(A+2), *(A+3));
    A = B;
    printf("%f, %f, %f, %f\n", *A, *(A+1), *(A+2), *(A+3));
    return(EXIT_SUCCESS);
}
```

a. (6 Points) Write the output of this program exactly as it would appear on the screen:

**Program output:**

```
9.000000, 3.400000, 1.200000, 3.200000
9.000000, 8.800000, 7.600000, 6.400000
3.400000, 5.600000, 7.800000, 9.000000
```

**Note to grader:** The zeros after the first decimal place are not necessary for full credit.

b. (8 Points) List the four pointer variables in this program, and for each one, state whether it points to stack memory or heap memory. If a pointer changes from stack to heap or heap to stack, make note of the point in the program where the change occurs.

**Solution:**

The variables `A`, `B`, `p`, and `q` are all of type pointer-to-double.

- `B` and `q` point to stack memory.
- `A` points to heap memory until the assignment `A = B`, which points it to stack memory.
- `p` points to heap memory until the assignment `p = &x`, which points it to stack memory.

c. (6 Points) Does this program contain any memory leaks? If so, what alteration(s) would be needed to eliminate the leak(s)?

**Solution:** The program contains two memory leaks. Do `free(A)` immediately before the assignment `A = B`, and do `free(p)` before the assignment `p = &x` to eliminate the leaks.
4. (20 Points) Write a C function called mix() with the heading given below, that takes as input two C-strings A and B, (i.e. null '\0' terminated char arrays), allocates a new char array from heap memory of appropriate length, copies the contents of A and B into that array by alternating one character from A and one from B (starting with A), adds the null character '\0' to make the array a valid C-string, then returns the address of the newly allocated array. Note that A and B may have different lengths. If one string is exhausted before the other, copy the remaining characters in the other string to the new array in order. Thus if A is the string "abcdef" and B is the string "xyz", then the returned string will be "axbyczdef". You may assume the availability of the C library function strlen() from string.h, which returns the length of a C-string.

Solution:
Note that there are a number of valid solutions to this problem, one of which is presented here.

char* mix(char* A, char* B){
    // your code begins here
    int i = 0, j = 0, k = 0;
    int n = strlen(A), m = strlen(B);
    char* C = calloc(n+m+1, sizeof(char));

    while( i<n && j<m ){
        if( k%2==0 ){
            C[k] = A[i];
            i++;
        }else{
            C[k] = B[j];
            j++;
        }
        k++;
    }
    while( i<n ){
        C[k] = A[i];
        i++;
        k++;
    }
    while( j<m ){
        C[k] = B[j];
        j++;
        k++;
    }
    C[k] = '\0';
    return C;
    // your code ends here
}
5. (20 Points) Write functions enqueue() and dequeue() for the Java implementation of an integer queue outlined below. The queue is implemented as a singly linked list with head and tail references. Function enqueue() places an new item at the back (tail) of the queue. Function dequeue() removes and returns the item at the front (head) of the queue, and has as precondition that the queue is not empty. It should check this precondition (by testing the value of numItems) and throw a new RuntimeException with appropriate error message if it is violated.

```java
class Queue{
    private class Node{
        int item;
        Node next;
        Node(int item){
            this.item = item;
            this.next = null;
        }
    }
    private Node head;
    private Node tail;
    private int numItems;
    public Queue(){head = null; tail = null; numItems = 0;}
    void enqueue(int x){
        // your code goes here
        if( numItems==0 ){
            head = tail = new Node(x);
        }else{
            tail.next = new Node(x);
            tail = tail.next;
        }
        numItems++;
    }
    int dequeue() throws RuntimeException{
        // your code goes here
        if( numItems==0 ){
            throw new RuntimeException("Error: cannot dequeue() empty queue");
        }
        int x = head.item;
        head = head.next;
        if( numItems==1 ){
            tail = null;
        }
        numItems--;
        return x;
    }
    // other Queue methods would follow
}
```