CMPS 12B
Introduction to Data Structures
Spring 2016
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.

```
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) Trace the following C program and place the output on the lines below *exactly* as it would appear on the screen. (Note: there are more lines printed below than are actually needed.)

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

int f(int x, int y){
   int u;
   u = x*y;
   printf("in f\n");
   return( x+u+y );
}

int g(int* p, int* q){
   int v;
   v = *p + *q;
   printf("in g, before f\n");
   *q = f(v, *p);
   printf("in g, after f\n");
   return( v-*q );
}

int main(void){
   int a=1, b=2, c=3;
   printf("in main, before f and g\n");
   a = f(a, b);
   b = g(&b, &c);
   printf("in main, after f and g\n");
   printf("a = %d, b = %d, c = %d\n", a, b, c);
   return(EXIT_SUCCESS);
}
```

Program Output:

```
in main, before f and g
in f
in g, before f
in f
in g, after f
in main, after f and g
a = 5, b = -12, c = 17
```

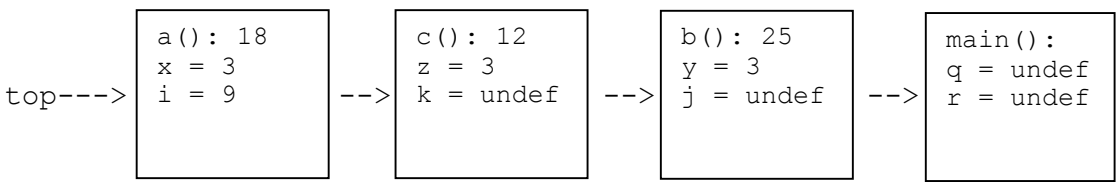
3. (20 Points) The following C program includes a global variable called `time`. Since it is declared outside of all functions (on line 3), its scope is the entire file. Notice that `time` is incremented before each of the functions `a`, `b`, and `c` return. Show the state of the function call stack when `time=2` (i.e. at the instant `time` becomes equal to `2`). Each stack frame should show the values of all function arguments and local variables, as well as the line to which execution will transfer when the function returns. If a local variable has not yet been assigned a value at `time=2`, indicate that fact 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.

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

Program Output:

q=21, r=30, time=6

State of the function call stack when time=2:



4. (20 Points) Consider the following C program.

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

int main(void){
   int i, j;
   double x = 4.2, y;
   double * A = calloc(4, sizeof(double));
   double B[] = {1.3, 5.2, 2.4, 3.5};
   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:

```
3.500000, 1.300000, 4.200000, 6.200000
3.500000, 3.400000, 7.200000, 4.300000
1.300000, 5.200000, 2.400000, 3.500000
```

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.

5. (20 Points) Write a C function called `concatenate()` with the prototype below that takes as input two int arrays *A* and *B* of lengths *n* and *m* respectively, allocates a new int array from heap memory of length $n+m$, copies the contents of *A* and *B* into that array (*A* first, *B* second), then returns the newly allocated array.

```
int* concatenate(int* A, int n, int* B, int m){

   // your code begins here

   int i;
   int* C = calloc(n+m, sizeof(int));

   for(i=0; i<n; i++){
      C[i] = A[i];
   }
   for(i=0; i<m; i++){
      C[n+i] = B[i];
   }

   return C;


   // your code ends here
}
```