# CMPS 12B
# Introduction to Data Structures
# Midterm 2 Review Problems

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

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

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

static int product(Node H){
   // Your code goes here









}
```

2. Write functions push() and pop() for the Java implementation of an integer stack outlined below. The stack is implemented as a singly linked list with a top Node reference. Function push() inserts a new item onto the top of the stack by inserting a new Node at the head of the list. Function pop() deletes the top item, and returns its value.

```
class Stack{
   private class Node{
      int item;
      Node next;
      Node(int item){
         this.item = item;
         this.next = null;
      }
   }
   private Node top;
   private int numItems;
   public Stack(){top = null; numItems = 0;}

   void push(int x){
      // your code goes here









   }

   int pop(){
      // your code goes here









   }
   // other Stack methods would follow
}
```

3. Trace the following C program and place the output on the lines below *exactly* as it would appear on the screen.

```
#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);
}
```

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

4. 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=4` (i.e. at the instant `time` becomes equal to `4`). 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=4`, indicate that fact by stating its value as `undef`. Assume that the terms in the right hand side of line 19 are evaluated from left to right, i.e. first call `a()` then call `b()`. 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 = y+a(y);
13.)    time++;
14.)    return(j);
15.) }
17.) int c(int z){
18.)    int k;
19.)    k = a(z)+b(z); // first call a() then b()
20.)    time++;
21.)    return(k);
22.) }
23.) int main(void){
24.)    int q, r;
25.)    time = 0;
26.)    q = b(5);
27.)    r = c(2);
28.)    printf("q=%d, r=%d, time=%d\n", q, r, time);
29.)    return(EXIT_SUCCESS);
30.) }
```

Output:

_____

State of the function call stack when time=4:

5. 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[] = {1.2, 5.3, 2.1, 3.4};
   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;
   }
   printf("%f, %f, %f, %f\n", *A, *B, *p, *q);
   A = B;
   printf("%f, %f, %f, %f\n", *A, *(A+1), *(A+2), *(A+3) );
   return(EXIT_SUCCESS);
}
```

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

_____

_____

b. List the pointer variables in this program, and for each one, state whether it points to stack memory or heap memory. If at some point in the program the pointer changes from stack to heap or heap to stack, note the point in the program where that happens.

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

6. Write a C function called `search()` with the prototype below that takes as input a null terminated `char` array S (i.e. a string) and a single `char c`, and returns the leftmost index in S at which the target `c` appears, or returns -1 if no such index exists.

```
int search(char* S, char c){
  // your code goes here









}
```

7. Write a C function called `diff()` with the prototype below that takes as input two null terminated char arrays (i.e. strings) A and B and returns the *difference* string consisting of all chars in A that are not in B. The returned chars should be stored in a null terminated char array in the same order they appeared in A, possibly with repetitions. The returned array should be allocated from heap memory to be the same length as A (although the null terminator might not appear at the end of this array.) You may use the `strlen()` function found in the library `string.h` to determine this length. You may also assume the existence of a C function called `search()` as described in problem 6.

```
char* diff(char* A, char* B){
  // your code goes here









}
```

8. Consider the C function below called `wasteTime()`. Your goal is to determine how much time `wastTime()` wastes. The stared (*) lines below are to be considered basic operations, which do nothing but waste a multiple of some unspecified time unit. Determine the total amount T(*n*) of time wasted on the input *n*. Find the asymptotic runtime of this algorithm, i.e. T(*n*) = Θ(some simple function of *n*).

```
 void wasteTime(int n){
    int i, j, k;
*   waste 2 units of time;
    for(i=0; i<n; i++){
*      waste 5 units of time;
       for(j=0; j<n j++){
*         waste 12 units of time;
          for(k=0; k<n; k++){
*            waste 3 units of time;
          }
       }
    }
 }
```

9. For each of the Java programs in problems 5 and 6 of the Midterm 1 review, do the following:
   a. Translate the program into C.
   b. Perform a box trace of the recursive function (`getValue(3, 13, 5)` and `displayOctal(100)` respectively).
   c. Perform a stack trace for the recursive function (`getValue(3, 13, 5)` and `displayOctal(100)` respectively).

10. Rewrite the sorting algorithms `BubbleSort()`, `SelectionSort()` and `InsertionSort()` (found on the class webpage in Examples/Lecture/C_Programs/SortingSearching/Sort.c) so that they sort in decreasing instead of increasing order.

11. Adapt the same sorting algorithms from problem 10 so that they operate on arrays of strings (i.e. null (\0) terminated char arrays) instead of ints.

12. Write a C function called `CountComparisons()` that takes as input an `int` array `A`, and `int n` giving the length of `A`, and an `int i` specifying an index to `A`. The function will return an `int` giving the number of elements in `A` that are less than `A[i]`. Determine the number of comparisons performed by your function (in terms of the array length *n*). How can you use your function as the basis for a sorting algorithm?

```
int CountComparisons(int* A, int n, int i){
   // your code goes here









}
```

13. Use the function `CountComparisons()` in the previous problem to create a sorting function with heading `void ComparisonSort(int* A, int* B, int n)` that takes an int array `A[]` as input, and copies the elements in `A[]` into the int array `B[]` in sorted order. (Hint: First assume the elements of `A[]` are distinct. In this case the number of numbers in `A[]` that are less than `A[i]` is the index where `A[i]` belongs in the output array `B[]`. Figure out what to do in the case that `A[]` contains repeated elements.)