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

```java
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 stack by inserting a new Node at the head of the list.

```java
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.

```c
#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`. Also determine the program output, and print it on the line below exactly as it would appear on the screen.

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

int time;

int a(int x){
    int i;
    i = x*x;
    time++;
    return(i);
}

int b(int y){
    int j;
    j = y+a(y);
    time++;
    return(j);
}

int c(int z){
    int k;
    k = a(z)+b(z);
    time++;
    return(k);
}

int main(void){
    int q, r;
    time = 0;
    q = b(5);
    r = c(2);
    printf("q=%d, r=%d, time=%d\n", q, r, time);
    return(EXIT_SUCCESS);
}
```

Output:

```
```

State of the function call stack when `time=4`:
5. Consider the following C program.

```c
#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
", *A, *B, *p, *q);
    A = B;
    printf("%f, %f, %f, %f
", *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.

```c
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 may 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.

```c
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 `wasteTime()` 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. $T(n) = \Theta($some simple function of $n$).

```c
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;
            }
        }
    }
}
```