1. (20 Points) Write the C functions described below. Each function operates on a singly linked list built from the following Node and NodeObj types.

```c
typedef struct NodeObj* Node;
typedef struct NodeObj{
    int item;
    Node next;
}NodeObj;
```

a. (5 Points) A constructor for the Node type.

```c
Node newNode(int x){
    Node N = malloc(sizeof(NodeObj));
    N->item = x;
    N->next = NULL;
    return N;
}
```

b. (5 Points) A destructor for the Node type.

```c
void freeNode(Node* pN){
    if(pN!=NULL && *pN!=NULL){
        free(*pN);
        *pN = NULL;
    }
}
```

c. (5 Points) A recursive function that prints out the items in the list headed by \( H \) in reverse order.

```c
void printBackward(Node H){
    if(H!=NULL){
        printBackward(H->next);
        printf("%d ", H->item);
    }
}
```

d. (5 Points) A recursive function the returns the sum of the items in the list headed by \( H \). The sum of an empty list is defined to be 0.

```c
int sum(Node H){
    if( H==NULL ){
        return 0;
    }else{
        return (H->item) + sum(H->next);
    }
}
```
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=6` (i.e. at the instant `time` becomes equal to 6). 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=6`, 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
#include<stdio.h>
#include<stdlib.h>
int time;
int f(int x){
    int i;
    i = x*x+1;
    time++;
    return(i);
}
int g(int y){
    int j;
    j = f(y)+f(time); // first call f(y), then call f(time)
    time++;
    return(j);
}
int h(int z){
    int k;
    k = f(z)+g(z); // first call f(z), then call g(z)
    time++;
    return(k);
}
int main(void){
    int a, b, c;
    time = 0;
    a = f(3);
    b = g(a);
    c = h(4);
    printf("a=%d, b=%d, c=%d, time=%d\n", a, b, c, time);
    return(EXIT_SUCCESS);
}
```

Program Output:
```
a=10, b=106, c=71, time=9
```

State of the function call stack when `time=6`:

```
<table>
<thead>
<tr>
<th>Function</th>
<th>Scope</th>
<th>Arguments</th>
<th>Local Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>f()</td>
<td>12</td>
<td>x = 4</td>
<td>i = 17</td>
</tr>
<tr>
<td>g()</td>
<td>18</td>
<td>y = 4</td>
<td>j = undef</td>
</tr>
<tr>
<td>h()</td>
<td>27</td>
<td>z = 4</td>
<td>k = undef</td>
</tr>
<tr>
<td>main()</td>
<td>OS</td>
<td>a = 10</td>
<td>b = 106, c = undef</td>
</tr>
</tbody>
</table>
```

Top

Bottom
3. (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.

```c
char* mix(char* A, char* B)
{
    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;
}
```

4. (20 Points) Determine a polynomial \( T(n) \) in \( n \) giving the number of basic operations performed by the following C function.

```c
void wasteTime(int n){
    int i, j;
    // perform 1 basic operation
    for(i=1; i<=n; i++){
        // perform 2 basic operations
        for(j=1; j<i; j++){
            // perform 3 basic operations
        }
    }
}
```

**Solution:**

\[
T(n) = 1 + 2n + 3(0 + 1 + 2 + \cdots + (n - 1)) \\
= 1 + 2n + 3 \cdot \frac{n(n-1)}{2} \\
= 1 + 2n + \frac{3}{2}n^2 - \frac{3}{2}n \\
= \frac{3}{2}n^2 + \frac{1}{2}n + 1
\]

5. (20 Points) Write the C function `countAllComparisons()` below. This function will take as input an int array \( A[] \) of length \( n \). It will return a pointer to an int array of length \( n \), allocated from heap memory, whose \( i \)th entry is the number of elements in \( A[] \) that are strictly less than \( A[i] \), for \( 0 \leq i \leq (n - 1) \).

```c
int* countAllComparisons(int* A, int n){
    // your code starts here
    int i, j;
    int* C = calloc(n, sizeof(int));
    for(i=0; i<n; i++){
        C[i] = 0;
        for(j=0; j<n; j++){
        }
    }
    return C;
    // your code ends here
}
```