CSE 15
Introduction to Data Structures
Fall 2019
Midterm Exam 2 Solutions

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) Write functions void push(int x) and int pop() for a C implementation of an integer stack based on the Node type defined in problem (1). Both functions assume the existence of a global Node reference variable, called top, pointing to a linked list data structure regarded as a stack. Function push() adds x to the top of the stack and deals correctly with the case of an empty stack. Function pop() returns and removes the top element, and has as precondition that the stack is not empty (top!=NULL). Neither function creates any memory leaks.

```c
// global
Node top = NULL;

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

    Node N = newNode(x);
    N->next = top; // works if stack is empty (top==NULL)
    top = N;

} // your code ends here

int pop(){
    // your code starts here

    int x;
    Node N;
    if(top==NULL){
        fprintf(stderr, "cannot pop() empty stack\n");
        exit(EXIT_FAILURE);
    }

    N = top;
    top = top->next;
    x = N->item;
    freeNode(&N);
    return x;

} // your code ends here
```
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 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.

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 = f(y)+f(time); // first call f(y), then call f(time)
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=106, c=71, time=9

State of the function call stack when time=6:

- **f()**: \(x = 4\), \(i = 17\)
- **g()**: \(y = 4\), \(j = \text{undef}\)
- **h()**: \(z = 4\), \(k = \text{undef}\)
- **main()**: OS
  a = 10
  b = 106
  c = \text{undef}
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=0; i<n; i++){
        // perform 1 basic operation
        for(j=0; j<=i; j++){
            // perform 2 basic operations
        }
    }
}
```

**Solution:**

$$T(n) = 1 + n + 2(1 + 2 + 3 + \cdots + n)$$

$$= 1 + n + 2 \cdot \frac{n(n+1)}{2}$$

$$= 1 + n + n(n + 1)$$

$$= (n + 1)^2 = n^2 + 2n + 1$$

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; // this line only necessary if you use malloc()
        for(j=0; j<n; j++){
        }
    }
    return C;
} // your code ends here
```