Linear Data Structures

Linked List

1. Let Node be the following structure

```
struct Node {

int key;

Node* next;

};
```

Write an algorithm that takes a pointer to Node which is the head of a NULL-terminated singly linked list and returns an integer which is the following alternating sum: first key minus second key plus third key minus fourth key, and so forth. For example,

input : a.4 → b.2 → c.5 → d.2, output : 5, because 5 = 4 − 2 + 5 − 2

input : a.4 → b.2 → c.5, output : 7, because 7 = 4 − 2 + 5

input : EMPTY – output : 0

Solution

Algorithm alt_sum(head)

1. Sum = 0
2. position = 0
3. while (head != NULL) {

```
if (position % 2 == 0)
        sum = sum +  head->key
else
        sum = sum -  head->key
position = position +1
head = head -> next
}
```

4. return sum
5. end

2. Let Node be the following structure

```
struct Node {

int key;

Node* next;

};
```

Write an algorithm that takes a pointer to Node which is the head of a NULL-terminated singly linked list and modifies all of the nodes' key so that the new key of each node is the sum of all the old keys from that node to the end of the list.

For example, if the list has four elements a.4 → b.2 → c.5 → d.2, then after running the new payloads are

a.13 → b.9 → c.7 → d.2.

```
Algorithm add(head) {

for (; head != NULL; head = head->next) {

      for (tmp = head->next; tmp != NULL; tmp = tmp->next)

                  head-> key = head->key  +  tmp->key

}

end
```

3. Let Node be the following structure

```
struct Node {

int key;

Node* next;

}
```

Write an algorithm that takes a pointer to Node which is the head of a NULL-terminated singly linked list, and a key x. The algorithm is to remove all nodes with key = x and returns the pointer to the head of the resulting linked list. Only pointer manipulation is allowed.

4. Let Node be the following structure

```
struct Node {

int key ;

 Node* next; }
```

Write an algorithm that takes a pointer to Node which is the head of a NULL-terminated singly linked list. The function swaps the first and the second element, the third and the fourth, the fifth and the sixth, etc., and then returns a pointer to the head of the new list. Only pointer manipulation is allowed. If there are an odd number of elements, then the last one is left alone.

For example, if the input list is a → b → c → d → e → f → NULL

then the output list is b → a → d → c → f → e → NULL, and a pointer to b is returned

And, if the input list is a → b → c → d → e → NULL

then the output list is b → a → d → c → e → NULL, and a pointer to b is returned.

```
Algorithm swap(Head)
if (head == NULL || head->next == NULL) return head;
ret = head->next;
prev = NULL;
first = head;
while (first != NULL && first->next != NULL) {
    second = first->next;
    if (prev != NULL)
            prev->next = second;
    first->next = second->next;
    second->next = first;
    prev = first;
    first = prev->next;
}
return ret;
```

Stacks

1. Suppose you have a single array of size N and want to implement two stacks so that you won't get overflow until the total number of elements on both stacks is N+1. How would you proceed?
2. Write algorithm to convert an infix expression to postfix form.
3. Write algorithm to check the well formedness of parenthesis in an expression.
4. Write an algorithm range count() that takes a stack S of integers and an integer a as arguments, and returns the number of integers in stack that are at least a.

```
Algorithm range_count(S, a)
1. count = 0;
2. while (isEmpty(S)==False) {
    if (Top(S) >= a) count++
    pop(S);
  }
3. return count
4. end
```