Guided Exercise 03

Linked lists & Memory Management

Open Book; Open Notes; Time Given =120 minutes

"By proceeding I certify that I have neither received nor given unpermitted aid on this examination and that I have reported all such incidents observed by me in which unpermitted aid is given."

Zip the assignment folder and rename it to '<rollnumer>_GE3.zip'. Upload your zip file in the corresponding LMS submission tab. Example, '26100181_GE3.zip'

Task 1: Self-referential structures (marks = 20).

Implement a simple linked list using self-referential structures. The linked list will be singly linked, meaning that each node in the list will only point to one other node (the next node in the list).

The linked list will be capable of performing two simple tasks, adding data and printing the data.

```
typedef struct Node {
    int data;
    struct Node* next;
} Node;
```

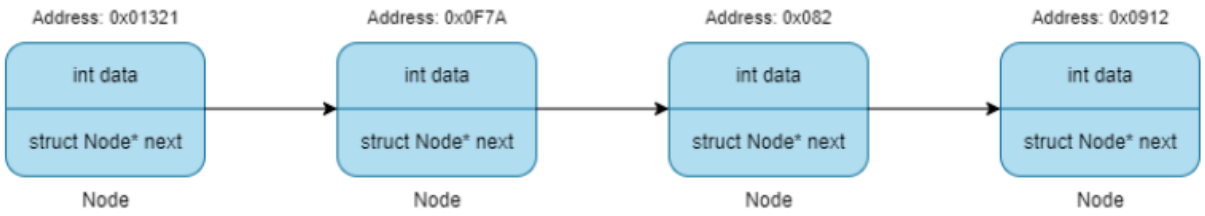

Figure 1: Linked list formed by self-referential nodes

Note that the addresses for each node are completely random. This means that the nodes are not adjacent inside the memory, so the representation above is a simplification to clarify how the nodes link with each other.

The C language implementation of following functions are provided to you

1. **Node* create_node(int value)**
2. **void add_node(Node* head, Node* to_add)**

Your task is to implement the following functions for linked list:

- **int removeNode(Node* head, int target_value)**

Remove the node holding the 'target_value' from the linked list. Reconnect the list after removing the node and free all unrequired memory. Return 1 if successful; else, return 0. For this task, you may assume that the node at the start of the list (pointed to by a head pointer) will NOT be removed. You are not required to implement functionality to delete a head pointer and reassign it. All target values passed will be from the second node onwards in the list.

- **int Count(Node* head, int searchFor)**

Count the number of times a given integer 'searchFor' occurs in a linked list. Traverse the list and increment the count wherever that number is found. Return the count.

- **int evaluatePolynomial(Node* head, int x)**

Evaluate the value of a polynomial for a given value of x, where the polynomial is represented as a singly linked list. Each node in the list contains a coefficient of the polynomial, starting from the constant term.

- The **first node** of the list represents the constant term (i.e., the coefficient of x^0).
- The **second node** represents the coefficient of x^1
- The **third node** represents the coefficient of x^2, and so on, with each subsequent node corresponding to higher powers of x

For example, if the linked list represents the polynomial $3 + 2x + 5x^2$(where the first node has 3, the second node has 2, and the third node has 5), and $x = 2$, then:

$$evaluatePolynomial(head, 2) = 3 + 2(2) + 5(2)^2 = 3 + 4 + 20 = 27$$

Hint1: C allows you to compute power directly using the function pow(base, exponent)