Magic Linked List

content: Structure; Pointer; Memory Management

Ziling Liu, 2024.10

Background

In data structures, a linked list is a flexible storage structure that allows you to dynamically add and delete data items during program execution. This assignment will require you to implement a simple unidirectional linked list whose node can store data with its bytes' number. Morever, each node should save the address of next node, making the whole linked list connected.
Through this project, you will gain a deeper understanding of memory management, pointer operations, and handling of multi type data in the C language.

Description

You need to design and implement a linked list data structure called `MagicLinkedList`, where each node in the linked list can store its data and its byte number.

The following operation should be supported by `MagicLinkedList`:

1. `create`: Create an empty `MagicLinkedList`.
2. `append_node`: Append a node to the end of the linked list.
3. `insert_node`: Insert a new node at a specified location in the linked list.
4. `delete_node`: Delete the node at the specified location and free up its memory.
5. `destroy`: Destroy the linked list and release the memory of all nodes.
6. `printx`: Print the data of each node in the linked list in order from the first node.
7. `printx_reverse`: Reverse printing the data of each node in the linked list from the tail node.

Input

The first line contains an integer T, the number of operations. Then the following lines are operations.

- `C`: Initialize a magic linked list.
- `A l s`: append an item to the linked list, where `l` represents the length of a subsequent string, and `s` represents the octal string, starting with "0o" and within ['1'-'7']. In this assignment every three octal number represent a byte, the last bit is ignored. For example, `A 8 0o123321` means two bytes `001 010 01` and `011 010 00`. If the number of characters in octal cannot be divided by 3, pad the input with 0. For example, 0o7 -> 0o007
- `I i l s`: Insert a new node at position `i` in the linked list, `l`, `s` are same as above.
- `D i`: delete the `i` th node and release its memory.
- `X`: Destroy the linked list and release all memory.

Output

- `Q`: Print the stored data of the linked list from head node in order and print the data of each node.
- `R`: Reverse printing the data of each node in the linked list from the tail node.

Reminder: In this assignment, be careful of null pointer and invalid index access. `memset()` *and* `strcat()` *are banned.*

Sample Input #1

```
4
C
A 17 0o220312330330336
I 1 20 0o256336344330310102
Q
```

Sample Output #1

```
Hello
World!
```

Explaination:
220 -> 010 010 000 -> 0100 1000 -> 72 -> 'H'
312 -> 011 001 010 > 0110 0101 -> 101 -> 'e'

Sample Input #2

```
9
C
D -1
I 0 22 0o206246144142162
A 23 0o2563123303063363323l2
D 3
I 1 8 0o350336
A 8 0o334336
D 3
R
```

Sample Output #2

```
Welcome
to
CS219
```

Template

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct Node Node;
/*
The parameters in the following two structures are enough to finish the
assignment. However, If adding some parameters to these two structures can be
helpful, you can also do that.
*/
typedef struct Node {
    void* data;           // pointer to node data
    size_t byte_num;     // The size of the data in bytes
```

```
    Node* next;    // pointer to next node
} Node;

// Definition of linked list node structure
typedef struct MagicLinkedList {
    Node* head;           // head node
    Node* tail;           // tail node
} MagicLinkedList;

/* Your code starts here */

// Save every three octal string into bytes
void octal2byte(void *dst, char *octal) {
    /*
    Function: For the given octal string "0o123321", write integer equaling
binary representation '00101001 01101000' into dst. Save each byte into dst.

    Notes:
    - The length of octal might be ODD, make sure the each octal number can be
grouped in a byte. For example, "0o66777" should be stored as 0o066777 by simply
padding 0 here).
    - You should return immediately if any failure happens.
    */
    // TODO: Implement this function
    return;
}

// Create a new linked list
MagicLinkedList* create() {
    // Allocate memory for the linked list
    return NULL; // TODO: Implement this function
}

// Append a new node to the end of the linked list
void append_node(MagicLinkedList* list, void* data) {
    // Allocate memory for a new node and set its data and size
    // Link the new node to the end of the linked list
    // TODO: Implement this function
}

// Insert a new node at a specific position in the linked list
void insert_node(MagicLinkedList* list, int position, void* data) {
    // Allocate memory for a new node and set its data and size
    // Insert the node at the specified position in the linked list
    // position is non-negative and start from 0. If position is invalid or out
of bound, don't operate on list.
    // TODO: Implement this function
}

// Delete a node from the linked list at a specific position
void delete_node(MagicLinkedList* list, int position) {
    // Remove the node at the specified position and free its memory.
    // position is non-negative and start from 0. If position is invalid or out
of bound, don't operate on list.
    // TODO: Implement this function
}
```

```c
// Destroy the linked list and free all allocated memory
void destroy(MagicLinkedList* list) {
    // Free all nodes in the list and the list itself
    // TODO: Implement this function
}

void printx(MagicLinkedList *LinkedList){
    // TODO: print the node's data with its corresponding ASCII character from head node to tail node. Print each node in one line.
    /*
    Note:
    For each node, print every 1 byte (char) in [33, 127) as a character, if data is not within the domain, use '#' instead. You should print a node in one line.
    */
    return;
}

void printx_reverse(MagicLinkedList *LinkedList){
    // TODO: Similar with printx, only reverse the order of nodes, the values inside the nodes are not reversed.
}
// Do not make any modifications to this main function
int main() {
    int T;
    scanf("%d", &T);
    MagicLinkedList *m_list = NULL;

    while (T--) {
        char op;
        scanf(" %c", &op);
        switch (op) {
        case 'C': {
            if (m_list != NULL) {
                destroy(m_list);
            }
            m_list = create();
            break;
        }
        case 'A': {
            int len;
            scanf("%d", &len);
            char *octal_str = (char *)malloc(len + 1);
            scanf("%s", octal_str);
            append_node(m_list, octal_str);
            free(octal_str);
            break;
        }
        case 'I': {
            int index;
            scanf("%d", &index);
            int len;
            scanf("%d", &len);
            char *octal_str = (char *)malloc(len + 1);
            scanf("%s", octal_str);
            insert_node(m_list, index, octal_str);
            free(octal_str);
            break;
        }
```

```
        case 'D': {
            int item_id;
            scanf("%d", &item_id);
            delete_node(m_list, item_id);
            break;
        }
        case 'X': {
            destroy(m_list);
            m_list = NULL;
            break;
        }
        case 'Q': {
            printx(m_list);
            break;
        }
        case 'R': {
            printx_reverse(m_list);
            break;
        }
        default: {
            break;
        }
        }
    }

    if (m_list != NULL) {
        destroy(m_list);
    }

    return 0;
}
```