04

Linked List Applications

DATA STRUCTURE LAB

SESSION - 04

Iterating over a list

Let's build a function that prints out all the items of a list. To do this, we need to use a `current` pointer that will keep track of the node we are currently printing. After printing the value of the node, we set the `current` pointer to the next node, and print again, until we've reached the end of the list (the next node is NULL).

```
void print_list(node_t * head) {
   node_t * current = head;

   while (current != NULL) {
      printf("%d\n", current->val);
      current = current->next;
   }
}
```

Adding an item to the end of the list

To iterate over all the members of the linked list, we use a pointer called `current`. We set it to start from the head and then in each step, we advance the pointer to the next item in the list, until we reach the last item.

```
void push(node_t * head, int val) {
   node_t * current = head;
   while (current->next != NULL) {
      current = current->next;
   }

   /* now we can add a new variable */
   current->next = malloc(sizeof(node_t));
   current->next->val = val;
   current->next->next = NULL;
}
```

Removing the first item (popping from the list)

To pop a variable, we will need to reverse this action:

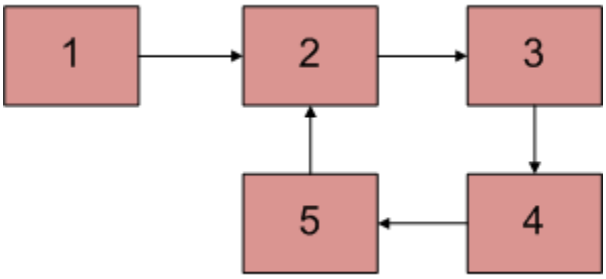
1. Take the next item that the head points to and save it
2. Free the head item
3. Set the head to be the next item that we've stored on the side

Here is the code:

```
int pop(node_t ** head) {
    int retval = -1;
    node_t * next_node = NULL;

    if (*head == NULL) {
        return -1;
    }

    next_node = (*head)->next;
    retval = (*head)->val;
    free(*head);
    *head = next_node;

    return retval;
}
```

Check linked list with a loop is palindrome or not

Given a linked list with a loop, the task is to find whether it is palindrome or not. You are not allowed to remove the loop.



Examples:

```
Input : 1 -> 2 -> 3 -> 2
            /|\     \|/
              ------- 1
Output: Palindrome
Linked list is 1 2 3 2 1 which is a
palindrome.

Input : 1 -> 2 -> 3 -> 4
            /|\     \|/
              ------- 1
Output: Palindrome
Linked list is 1 2 3 4 1 which is a
not palindrome.
```

```
// C++ program to check if a linked list with
// loop is palindrome or not.
#include<bits/stdc++.h>
using namespace std;

/* Link list node */
struct Node
{
    int data;
    struct Node * next;
};

/* Function to find loop starting node.
```

```
loop_node --> Pointer to one of the loop nodes
head --> Pointer to the start node of the linked list */
Node* getLoopstart(Node *loop_node, Node *head)
{
    Node *ptr1 = loop_node;
    Node *ptr2 = loop_node;

    // Count the number of nodes in loop
    unsigned int k = 1, i;
    while (ptr1->next != ptr2)
    {
        ptr1 = ptr1->next;
        k++;
    }

    // Fix one pointer to head
    ptr1 = head;

    // And the other pointer to k nodes after head
    ptr2 = head;
    for (i = 0; i < k; i++)
        ptr2 = ptr2->next;

    /* Move both pointers at the same pace,
    they will meet at loop starting node */
    while (ptr2 != ptr1)
    {
        ptr1 = ptr1->next;
        ptr2 = ptr2->next;
    }
    return ptr1;
}

/* This function detects and find loop starting
  node  in the list*/
Node* detectAndgetLoopstarting(Node *head)
{
    Node *slow_p = head, *fast_p = head,*loop_start;

    //Start traversing list and detect loop
    while (slow_p && fast_p && fast_p->next)
    {
        slow_p = slow_p->next;
        fast_p = fast_p->next->next;

        /* If slow_p and fast_p meet then find
           the loop starting node*/
        if (slow_p == fast_p)
        {
            loop_start = getLoopstart(slow_p, head);
            break;
        }
    }

    // Return starting node of loop
```

```
    return loop_start;
}

// Utility function to check if a linked list with loop
// is palindrome with given starting point.
bool isPalindromeUtil(Node *head, Node* loop start)
{
    Node *ptr = head;
    stack<int> s;

    // Traverse linked list until last node is equal
    // to loop_start and store the elements till start
    // in a stack
    int count = 0;
    while (ptr != loop_start || count != 1)
    {
        s.push(ptr->data);
        if (ptr == loop_start)
            count = 1;
        ptr = ptr->next;
    }
    ptr = head;
    count = 0;

    // Traverse linked list until last node is
    // equal to loop_start second time
    while (ptr != loop_start || count != 1)
    {
        // Compare data of node with the top of stack
        // If equal then continue
        if (ptr->data == s.top())
            s.pop();

        // Else return false
        else
            return false;

        if (ptr == loop_start)
            count = 1;
        ptr = ptr->next;
    }

    // Return true if linked list is palindrome
    return true;
}

// Function to find if linked list is palindrome or not
bool isPalindrome(Node* head)
{
    // Find the loop starting node
    Node* loop_start = detectAndgetLoopstarting(head);

    // Check if linked list is palindrome
    return isPalindromeUtil(head, loop_start);
}
```

```
Node *newNode(int key)
{
    Node *temp = new Node;
    temp->data = key;
    temp->next = NULL;
    return temp;
}

/* Driver program to test above function*/
int main()
{
    Node *head = newNode(50);
    head->next = newNode(20);
    head->next->next = newNode(15);
    head->next->next->next = newNode(20);
    head->next->next->next->next = newNode(50);

    /* Create a loop for testing */
    head->next->next->next->next->next = head->next->next;

    isPalindrome(head)? cout << "\nPalindrome"
                      : cout << "\nNot Palindrome";

    return 0;
}
```

Output:

```
Palindrome
```

**Iteratively Reverse a linked list using only 2 pointers (An Interesting Method)

Given pointer to the head node of a linked list, the task is to reverse the linked list.

Examples:

```
Input : Head of following linked list
       1->2->3->4->NULL
Output : Linked list should be changed to,
       4->3->2->1->NULL


Input : Head of following linked list
```

```
       1->2->3->4->5->NULL

Output : Linked list should be changed to,

       5->4->3->2->1->NULL
```

```
// C++ program to reverse a linked list using two pointers.
#include<bits/stdc++.h>
using namespace std;
typedef uintptr_t ut;

/* Link list node */
struct node
{
    int data;
    struct node* next;
};

/* Function to reverse the linked list using 2 pointers */
void reverse(struct node** head_ref)
{
    struct node* prev   = NULL;
    struct node* current = *head_ref;

    // at last prev points to new head
    while (current != NULL)
    {
        // This expression evaluates from left to right
        // current->next = prev, changes the link fron
        //                 next to prev node
        // prev = current, moves prev to current node for
        //        next reversal of node
        // This example of list will clear it more 1->2->3->4
        // initially prev = 1, current = 2
        // Final expression will be current = 1^2^3^2^1,
        // as we know that bitwise XOR of two same
        // numbers will always be 0 i.e; 1^1 = 2^2 = 0
        // After the evaluation of expression current = 3 that
        // means it has been moved by one node from its
        // previous position
        current = (struct node *) ((ut)prev^(ut)current^(ut)(current->next)^
                                   (ut)(current->next = prev)^
                                   (ut)(prev = current));
    }

    *head_ref = prev;
}

/* Function to push a node */
void push(struct node** head_ref, int new_data)
{
    /* allocate node */
    struct node* new_node =
```

```
        (struct node*) malloc(sizeof(struct node));

    /* put in the data  */
    new_node->data  = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}

/* Function to print linked list */
void printList(struct node *head)
{
    struct node *temp = head;
    while (temp != NULL)
    {
        printf("%d  ", temp->data);
        temp = temp->next;
    }
}

/* Driver program to test above function*/
int main()
{
    /* Start with the empty list */
    struct node* head = NULL;

    push(&head, 20);
    push(&head, 4);
    push(&head, 15);
    push(&head, 85);

    printf("Given linked list\n");
    printList(head);
    reverse(&head);
    printf("\nReversed Linked list \n");
    printList(head);
    return 0;
}
```

Output:

```
Given linked list

85 15 4 20

Reversed Linked list

20 4 15 85
```