Department of
Computer Science and Engineering

Title: Linked List and Implementation of Singly Linked List

Data Structure Lab

CSE 106

Green University of Bangladesh

1 Objective(s)

- To attain knowledge on **linked list**.
- Recalling the knowledge on **pointer**, **malloc** and **struct / structure**.
- To implement **Singly linked list** using C.

2 Problem Analysis

A linked list is a linear collection of data elements whose order is not given by their physical placement in memory. The elements in a linked list are linked using pointers. It is a data structure consisting of a collection of nodes which together represent a sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration.

2.1 Types of Linked List

- **Singly Linked List** - It is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same data type. The node contains a pointer to the next node means that the node stores the address of the next node in the sequence. A single linked list allows traversal of data only in one way.

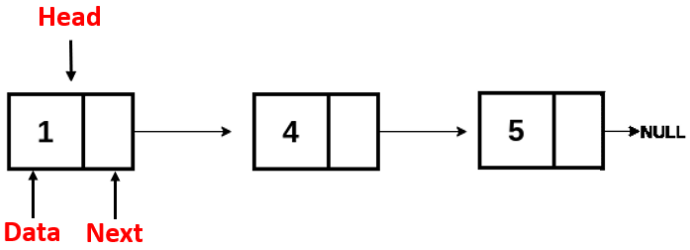


Figure 1: Singly Linked List

The nodes are connected to each other in this form where the value of the next variable of the last node is NULL i.e. next = NULL, which indicates the end of the linked list.

- **Doubly Linked List** - It is also known as two way linked list. A two-way linked list is a more complex type of linked list which contains a pointer to the next as well as the previous node in sequence, Therefore, it contains three parts are data, a pointer to the next node, and a pointer to the previous node. This would enable us to traverse the list in the backward direction as well.

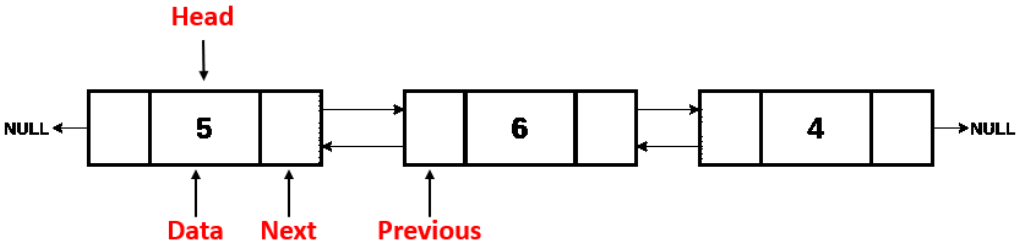


Figure 2: Doubly Linked List

- **Circular Linked List** - A circular linked list is that in which the last node contains the pointer to the first node of the list. While traversing a circular liked list, we can begin at any node and traverse the list in any direction forward and backward until we reach the same node we started. Thus, a circular linked list has no beginning and no end.
If not traversed carefully, then there is a possibility to end up in an infinite loop because NULL value

to stop the traversal is absent. Operations in a circular linked list are complex as compared to a singly linked list and doubly linked list like reversing a circular linked list.

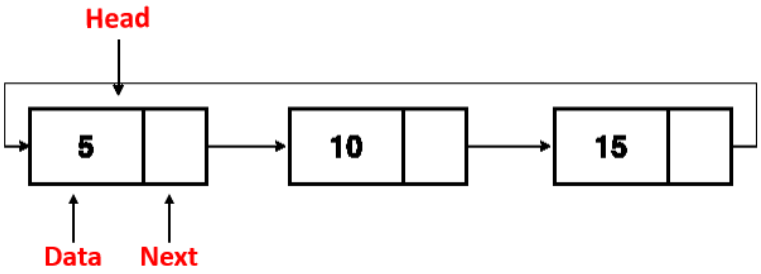


Figure 3: Circular Linked List

2.2 Basic Operations of a Linked List

- **Insert**
  - **At the Beginning Position**
  - **At the Last Position**
  - **At any Specific Position**
- **Delete**
  - **From the Beginning Position**
  - **From the Last Position**
  - **From any Specific Position**
- **Traverse**
- **Display**
- **Search**

2.3 Pointer, Malloc and Struct

- **Pointer - Pointers** are symbolic representation of addresses. They enable programs to simulate call-by-reference as well as to create and manipulate dynamic data structures.
- **Struct** - In C programming, a **struct** (or structure) is a collection of variables (can be of different types) under a single name.
- **Malloc()** - The function **malloc()** is used to allocate the requested size of bytes and it returns a pointer to the first byte of allocated memory. It returns null pointer, if fails. The content of the newly allocated block of memory is not initialized, remaining with indeterminate values.

3 Algorithm

Algorithm 1: Setting up an empty list

```
/* Algorithm for Setting up an empty list                                    */
```

1 Include all the header files which are used in the program.
2 Declare all the user defined functions.
3 Define a Node structure with two members data and next.
4 Define a Node pointer 'head' and set it to NULL.
5 Implement the main method by displaying operations menu and make suitable function calls in the main method to perform user selected operation.

Algorithm 2: Inserting At Beginning of the list

Input: Element

```
/* Algorithm for Inserting At Beginning of the list                          */
```

1 Create a newNode with given value.
2 **if** *head == NULL* **then**
3 | newNode→next = NULL
4 | head = newNode
5 **end**
6 **else**
7 | newNode→next = head
8 | head = newNode.
9 **end**

Algorithm 3: Inserting At End of the list

Input: Element

```
/* Algorithm for Inserting At End of the list                                */
```

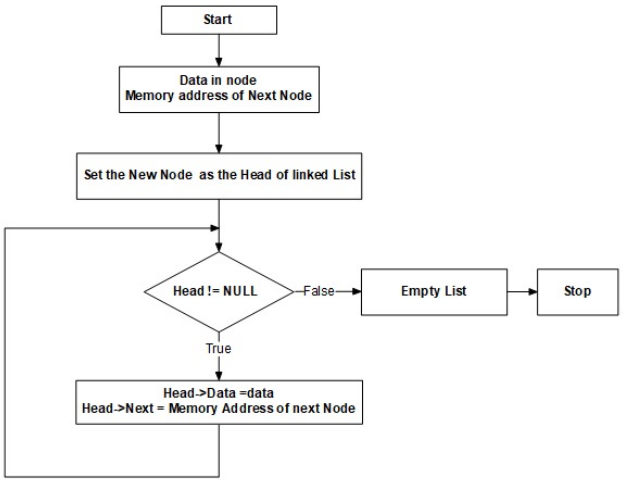
1 Create a newNode with given value and newNode→next as NULL.
2 **if** *head == NULL* **then**
3 | head = newNode
4 **end**
5 **else**
6 | define a node pointer temp
7 | temp = head
8 | **while** *temp->next != NULL* **do**
9 | | temp = temp->next;
10 | **end**
11 | temp->next = newNode;
12 **end**

Algorithm 4: Displaying the list

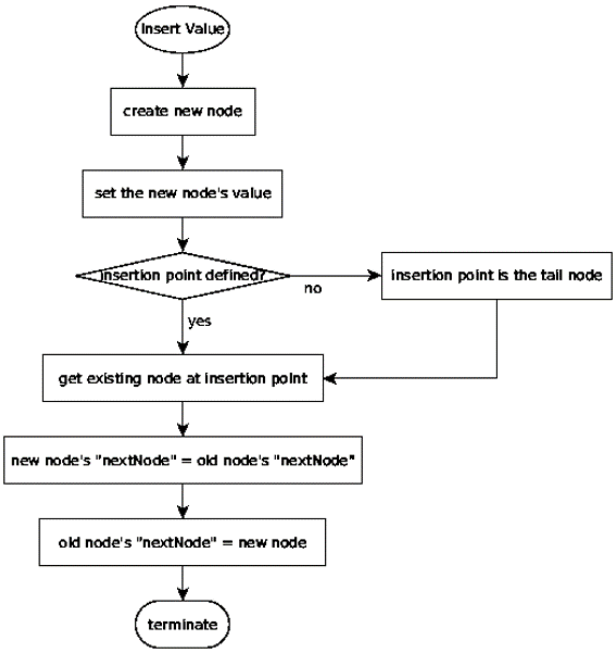
```
/* Algorithm for Displaying the list                                         */
```

1 **if** *head == NULL* **then**
2 | Print "List is Empty!!!"
3 **end**
4 **else**
5 | define a node pointer temp
6 | temp = head
7 | **while** *temp->next != NULL* **do**
8 | | print "%d —> ",temp->data
9 | | temp = temp->next
10 | **end**
11 | print "%d —> ",temp->data
12 **end**

4 Flowchart



(a) Inserting at the beginning



(b) Insert at any position

Figure 4: Insertion in Singly linked list.

5 Implementation in C

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

struct Node
{
    int data;
    struct Node *next;
}*head = NULL;

void insertAtBeginning(int value)
{
    struct Node *newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    if(head == NULL)
    {
        newNode->next = NULL;
        head = newNode;
    }
    else
    {
        newNode->next = head;
        head = newNode;
    }
    printf("\nOne node inserted!!!\n");
```

```
}

void insertAtEnd(int value)
{
    struct Node *newNode;
    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
    if(head == NULL)
    {
        head = newNode;
    }
    else
    {
        struct Node *temp = head;
        while(temp->next != NULL)
        temp = temp->next;
        temp->next = newNode;
    }
    printf("\nOne node inserted!!!\n");
}
void display()
{
    if(head == NULL)
    {
        printf("\nList is Empty\n");
    }
    else
    {
        struct Node *temp = head;
        printf("\n\nList elements are - \n");
        while(temp->next != NULL)
        {
            printf("%d --->",temp->data);
            temp = temp->next;
        }
        printf("%d --->NULL",temp->data);
    }
}

int main()
{
    int choice,value,choice1,loc1,loc2;
    while(1)
    {
        mainMenu: printf("\n\n****** MENU ******\n1. Insert\n2. Display\n3. Exit
            \nEnter your choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                printf("Enter the value to be insert: ");
                scanf("%d",&value);
                while(1)
                {
                    printf("Where you want to insert: \n1. At Beginning\n2. At
                        End\nEnter your choice: ");
                    scanf("%d",&choice1);
```

```
83                      switch(choice1)
84                      {
85                          case 1:
86                              insertAtBeginning(value);
87                              break;
88                          case 2:
89                              insertAtEnd(value);
90                              break;
91                          default:
92                              printf("\nWrong Input!! Try again!!!\n\n");
93                              goto mainMenu;
94                      }
95                      goto subMenuEnd;
96                  }
97              subMenuEnd:
98              break;
99              case 2:
100                 display();
101                 break;
102             case 3:
103                 exit(0);
104             default:
105                 printf("\nWrong input!!! Try again!!\n\n");
106         }
107     }
108     return 0;
109 }
```

6 Input/Output (Compilation, Debugging & Testing)

Input & Output:

****** MENU ******
1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter the value to be insert: 10
Where you want to insert:
1. At Beginning
2. At End
Enter your choice: 1
One node inserted!!!

****** MENU ******
1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter the value to be insert: 50
Where you want to insert:
1. At Beginning
2. At End
Enter your choice: 2
One node inserted!!!

****** MENU ******
1. Insert
2. Display
3. Exit
Enter your choice: 2

List elements are -
10 —>50 —>NULL

****** MENU ******
1. Insert
2. Display
3. Exit
Enter your choice: 3

Process returned 0 (0x0) execution time : 45.455 s
Press any key to continue.

7 Discussion & Conclusion

Based on the focused objective(s) and basic operations of a singly linked list, the additional lab exercise made me more confident to have a clear understanding about singly linked list and ultimately lead me towards the fulfilment of the objectives(s).

8 Lab Task (Please implement yourself and show the output to the instructor)

1. Modify the C program that is able to insert element at any specific position, Delete element from the beginning, last, any specific position as well as display the list after any modification.

8.1 Algorithm

Algorithm 5: Inserting At Any Specific Position of the list

Input: Element

```
/* Algorithm for Inserting Element at Any Specific Position                    */
Create a newNode with given value.
if head == NULL then
    newNode→next = NULL
    head = newNode
end
else
    define a node pointer temp
    temp = head
    while temp->next != location do
        temp = temp->next
        newNode->next = temp->next
        temp->next = newNode
    end
    print "One node inserted!!!"
end
```

9 Lab Exercise (Submit as a report)

- Find the specific node of element that is present or not in the singly linked list.
- Call a function that will generate the size of the singly linked list.
- Insert element between any specific position of the singly linked list

10 Policy

Copying from internet, classmate, seniors, or from any other source is strongly prohibited. 100% marks will be *deducted* if any such copying is detected.