SET213 – Data Structures & Algorithms

Experiment # 10

Experiment Title

Linked List - II

Assessment of CLO(s): IV

Performed on 13-12-2024

Student Name			
Roll No.		**Group**	
Semester		**Session**	

Total (Max)	**Criteria 1 (2.5)**	**Criteria 2 (2.5)**	**Criteria 3 (2.5)**	**Criteria 4 (2.5)**	**Total (10)**
Marks Obtained					
Remarks (if any)					

Experiment evaluated by

Instructor's Name	**Engr. Muhammad Asad Husain**		
Date		**Signature**	

Department of Engineering Technology
(UIT University)

Course Code: SET213 Course Title: Data Structures & Algorithms Course Credits: 3+1

Session: Fall 2024

Rubric for assessment criteria to perform experiment number 10.

Level / Criteria	UNSATISFACTORY 1	COMPETENT 2	PROFICIENT 3	DISTINGUISHED 4
Capability of writing algorithm/ Procedure	None of the steps are implemented of an algorithm.	Few steps are implemented correctly of an algorithm.	Most of the steps are implemented correctly of an algorithm.	All the steps are implemented correctly of an algorithm.
Capability of writing Program	Programs not completed.	Completeness of code, consistent variable naming and unformatted.	Completeness of code, inconsistent variable naming and well formatted.	Completeness of code, consistent variable naming and well formatted.
Completion of target in Lab	25% target has been completed	50% target has been completed	75% target has been completed	100% target has been completed
Output	None of the outputs are correct.	Few outputs have been found correctly.	Some of the outputs are correct and well formatted.	Most of the outputs are correct and well formatted.

Practical Objective(s):

i. Insertion and deletion in a linked list
ii. Searching an item in a linked list

Theory

In this practical, we will learn to implement three basic operations on a linked list i.e. insertion, deletion and searching. Before that, we present a review of how to create a linked list.

Creating a Linked List

First of all, we define a structure named as node, which has two fields: an integer variable named as data and a pointer variable named as next. The next pointer holds the address of the next node in the linked list. If there is only a single node in the linked list, the next pointer should point to NULL.

```
struct node
{
int data;
struct node* next;
};
```

After defining the structure **node**, we need to declare a pointer object of **node** structure. This pointer object **rootnode** serves as the root/start node of the linked list.

```
node *rootnode;
```

Now, we will create an object of structure node using the **new** operator. The new operator creates a **node** object and allocates it a memory space from heap memory. It then returns the memory address of the allocated memory space. We assign that memory space into the pointer variable **rootnode**. Remember, we are using **new** command for dynamic memory allocation as discussed in practical 8.

```
rootnode=new node;
```

The fields of a pointer object are accessed using **->** operator.

```
rootnode->data=50;
rootnode->next=NULL;
```

Similarly, we can create more nodes if we want to insert them into the linked list.

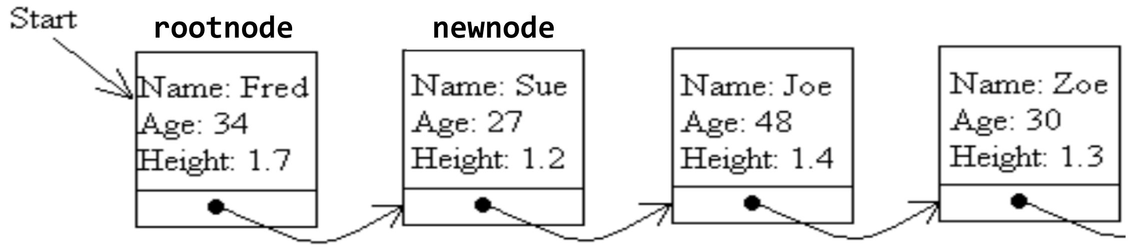
```
node *newnode; newnode = new node; newnode->data=100;
```

Insertion in a Linked List

A new node can be inserted at the front of a linked list as well as at the end of a linked list. We will discuss both of these cases.

I. Insertion at the end of a Linked List

A new node can be inserted at the end of a linked list so that it becomes the last node.



a) How to reach the end of a Linked List?

We know that the next pointer of the last node in a linked list points to NULL. This is how we identify the end of a linked list. In order to reach the end of a linked list, we need to start from **rootnode** and check each subsequent node for having a NULL pointer in its next field. For this we need to create a pointer object and assign **rootnode** to it.

```
node *currentnode;
currentnode = rootnode;
```

Unless we don't find a node whose next is pointing to NULL, we keep moving to the next node. After the completion of this while loop, we will get the address of last node of the linked list saved in **currentnode**.

```
while (currentnode->next != NULL)
{
currentnode = currentnode->next;
}
```

b) Insert the new node at the end of a Linked List

Once we have found the address of the last node, we will link the new node to it. We do this by setting the next field of the **currentnode** to **newnode**.

```
currentnode->next = newnode;
```

Since we have inserted this new node at the end of the linked list, it means that now **newnode** is the last node. Therefore, we need to set the next pointer field of the **newnode** to **NULL**.

```
newnode->next = NULL;
```

II. Insertion at the front of the Linked List

A new node can be inserted at the front of a linked list so that it becomes the first/root node. In this case, the initial **rootnode** will become the second node i.e. right next to the **newnode**.

```
newnode->next=rootnode;
rootnode=newode;
```

Deletion in a Linked List

We can delete a node from anywhere in a linked list. We discuss here two cases from the front of the linked list or from the end of a linked list.

I. Deleting a node from the end of a Linked List

In order to delete a node from the end of a linked list, we need to designate the second last node as the last node and then delete the previously known last node.

(a) **How to find the second last node in a Linked List?**

We know how to find the last node i.e. the node whose next pointer has a NULL value. As we said that we need to designate the second last node as the last node. Below we discuss, how to find the second last node.

```
node *prev=rootnode;
node *temp;
temp=prev->next;
while(temp->next!=NULL)
```

```
{
temp=temp->next;
prev=prev->next;
}
```

We keep track of two consecutive nodes at a time so that when we have the address of first node in **prev** and that of second node in temp. Then the address of second node will be saved in **prev** and that of third node in temp. Finally, the address of second last node and the last node will be saved in **prev** and temp, respectively.

Once we find that the address of second last node in **prev**, we set its next pointer to NULL, thus making it the last node. After that we delete temp which has the address of previously known last node.

```
prev->next=NULL;
delete temp;
```

(b) **Deleting a node from the front of a Linked List**

In order to delete a node from the front of a linked list, we need to delete the **rootnode** and designate the node next to **rootnode** as the new **rootnode**. It is important to note the order in which these steps will take place. First, the node next to the **rootnode** will be saved in an intermediate pointer e.g. save. After that, we will delete the **rootnode**. Lastly, the saved node will be assigned to **rootnode**. Thus, now the second node will become the **rootnode**.

```
node *save;
save=rootnode->next;
delete rootnode;
rootnode=save;
```

Searching an item in a Linked List

Once a linked list is created, we can search for a particular item in it. We start from the **rootnode** and keep moving to the next node unless either we find the desired item or the linked list ends i.e. node is **NULL**. In the former case, search is successful i.e. the item is found, whereas in the latter case search is unsuccessful.

```
node *counter=rootnode;
while(counter!=NULL && counter->data!=item)
{
counter=counter->next;
}
if(counter==NULL)
cout<<"Item not found!\n";
else
cout<<"Item found at\n”;
```

Do It Yourself:

1. (a) Define a structure named as “computer”, which has these fields:
   - Category (Workstation, Laptop, Tablet, etc.)
   - Brand name (e.g. Intel, HP, Lenovo, etc.)
   - RAM size in GB (1, 2, 4, 8, etc.)

   (b) Add another field which will point to the address of the next node.

2. Write a program which creates a linked list each of whose nodes is a structure of type computer (created in task 1).
   (a) Insert three computer nodes at the front of this linked list.
   (b) Delete a node from the end of the linked list.