

Lab 02

Singly Linked List

Objective:

The purpose of this lab session is to acquire skills in working with singly linked lists. lab will give you an overview of C++ language.

Activity Outcomes:

This lab teaches you the following topics:

- Creation of singly linked list
- Insertion in singly linked list
- Deletion from singly linked list
- Traversal of all nodes

Instructor Note:

As a pre-lab activity, read Chapter 14 from the text book “A Common-Sense Guide to Data Structures and Algorithms, Jay Wengrow, Pragmatic Bookshelf, 2020.”.

1) Useful Concepts

A *list* is a finite ordered set of elements of a certain type. The elements of the list are called *cells* or *nodes*. A list can be represented *statically*, using arrays or, more often, *dynamically*, by allocating and releasing memory as needed. In the case of static lists, the ordering is given implicitly by the one-dimension array. In the case of dynamic lists, the order of nodes is set by *pointers*. In this case, the cells are allocated dynamically in the heap of the program. Dynamic lists are typically called *linked lists*, and they can be singly- or doubly-linked.

The structure of a node may be:

struct *Nodetype*

```
{  
    int data; /* an optional field */  
    ... /* other useful data fields */  
    Nodetype *next=NULL; /* link to next node, assigned NULL so that should not point garbage*/  
};  
Nodetype *first=NULL, *last=NULL; /* first and last pointers are global and point first and last node */
```

A singly-linked list may be depicted as in Figure 1.1.

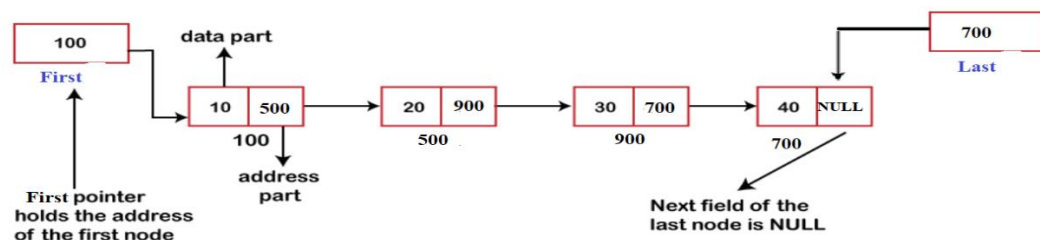


Figure 1.1.: A singly-linked list model.

2) Solved Lab Activities

<i>Sr.No</i>	<i>Allocated Time</i>	<i>Level of Complexity</i>	<i>CLO Mapping</i>
<i>Activity 1</i>	<i>10 mins</i>	<i>Medium</i>	<i>CLO-4</i>
<i>Activity 2</i>	<i>10 mins</i>	<i>Medium</i>	<i>CLO-4</i>
<i>Activity 3</i>	<i>5 mins</i>	<i>Medium</i>	<i>CLO-4</i>
<i>Activity 4</i>	<i>10 mins</i>	<i>Medium</i>	<i>CLO-4</i>
<i>Activity 5</i>	<i>5 mins</i>	<i>Medium</i>	<i>CLO-4</i>
<i>Activity 6</i>	<i>10 mins</i>	<i>Medium</i>	<i>CLO-4</i>

Activity 7	5 mins	Medium	CLO-4
Activity 8	5 mins	Medium	CLO-4

Activity 1:

Insertion at the end of the linked list

Consider the following steps which apply to dynamic lists:

```
Void insert_end()
{
    //take the pointer to hold the address of nodetype record
    Nodetype *p;
    //allocate runtime memory for new record of nodetype using new operator
    p = new Nodetype ;
    cout<<"Enter the data in node:  ";
    cin>>p->data;
    /* l i s t i s   empty */
    i f  (  first == NULL )
    /* p becomes first node and first and last pointer will point to same node */
        first=last = p;
    else {
    /* link the last pointer with newly created node (p) */
        last->next  = p;
        last = p;           /* assign p to last node */
    }
}
```

Output

Will insert the new node at the end of the linked list.

Activity 2:

Insertion at the start of the linked list

Consider the following steps which apply to dynamic lists:

```

Void insert_start()
{
    //take the pointer to hold the address of nodetype record
    Nodetype *p;
    //allocate runtime memory for new record of nodetype using new operator
    p = new Nodetype ;
    cout<<"Enter the data in node:  ";
    cin>>p->data;
    i f ( first == NULL ) /* l i s t i s   empty */
    /* p becomes first node and first and last pointer will point to same node */
        first=last = p;
    else {
        p->next=first; /* link the new node with first node */
        fisrt = p;      /* assign p to last node */
    }
}

```

Output

Will insert the new node at the end of the linked list.

Activity 3:

Accessing nodes of linked list

```

Nodetype search (int key){
    Nodetype *p = NULL;
    p = f i r s t ;
    while ( p != NULL  && p->data != key)
    {
        p = p->next ;
    }
    return p; /* if p is NULL then value not found */
}

```

Output

Return the pointer of Node if the required element is found.

Activity 4:

Inserting after specific value

```
void insert_after (int key){
    Nodetype *p = NULL;
    p=search(key);
    if(p==NULL)
        cout<<"value not found";
    else
    {
        Nodetype *Newnode;
        Newnode= new Nodetype;
        if(p==last)
        {
            last->next=p;
            last=p;
        }
        else
        {
            Newnode->next=p->next;
            p->next=Newnode;
        }
        Cout<<"New node linked successfully";
    }
}
```

Output

Will insert the new node after specified node

Activity 5:

Deleting first node from single linked list

```
Void delete_first()
{
    Nodetype *p;
```

```

i f ( f i r s t == NULL ) /* l i s t i s empty */
    cout<<"\n Linked List is empty";
else
{ /* non-empty l i s t */
    p = f i r s t ;
    f i r s t = f i r s t ->next ;
    delete ( p ) ; /* free up memory */
}
}

```

Output

Will delete the first node of the singly linked list.

Activity 6:

Deleting last node from single linked list

```

Void delete_last()
{
NodeT *q, *q1;
q1 = NULL; /* i n i t i a l i z e */
q = f i r s t ;
if(q==NULL)
{
    Cout<<"\n Linked List is empty";
}
else
{ /* non-empty l i s t */
while ( q != last )
{ /* advance towards end */
q1 = q; /*q1 will follow the q pointer */
q = q->next ;
}
i f ( q == f i r s t )
{ /* only one node */

```

```

    f i r s t = last = NULL;
}
else
{ /* more than one node */
    q1->next = NULL;
    last = q1;
}
delete q; } }

```

Activity 7:

Deleting a node given by user

```

void remove_spec(int key)
{
    Nodetype *q, *q1;
    q1 = NULL; /* i n i t i a l i z e */
    q = f i r s t ;
    /* search node */
    while ( q != NULL && q->data != key )
    {
        q1 = q;
        q = q->next ;
    }
    i f ( q == NULL )
    {
        cout<<" Not found supplied key";
    }
    else if(q==first && q==last)
    {
        delete q;
        first=last=NULL;
    }
    else if (q==last)
    {

```

```

q1->next=NULL;

last=q1;    /* make 2nd last node as last node */

delete q;

}

else      /* other than f i r s t node and last */

{

q1->next = q->next ;

delete q; } }

```

Output

Will delete the user given node of the singly linked list.

Activity 8:

Complete deletion of single linked list

```

while ( f i r s t != NULL )

{

p = f i r s t ;

f i r s t = f i r s t ->next ;

free ( p ) ;

}

last = NULL;

```

Output

All the nodes of Linked List will be deleted.

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab

Lab Task 1

Add functionality to display the Link List in reverse (both using loops and using recursive approach).

Lab Task 2

Add a function that merges two linked lists passed as parameter into a third new linked list.

Lab Task 3

Add a function to find multiple occurrences of data element in the list..

Lab 03

Doubly Linked List

Objective:

This lab session is intended to help you develop the operations on doubly-linked lists.

Activity Outcomes:

This lab teaches you the following topics:

- Creation of doubly linked list
- Insertion in doubly linked list
- Deletion from doubly linked list
- Traversal of all nodes

Instructor Note

As a pre-lab activity, read Chapter 14 from the text book “A Common-Sense Guide to Data Structures and Algorithms, Jay Wengrow, Pragmatic Bookshelf, 2020.”.