Assignment no	8
Aim	Write C++ program for storing appointment schedule for day. Appointments are booked randomly using linked list. Set start and end time and min and max duration for visit slot. Write functions for- a) Display free slots b) Book appointment c) Cancel appointment (check validity, time bounds, availability) d) Sort list based on time e) Sort list based on time using pointer manipulation OR Second year Computer Engineering class, set A of students like Vanilla Ice-cream and set B of students like butterscotch ice-cream. Write C++ program to store two sets using linked list. compute and display- a) Set of students who like both vanilla and butterscotch b) Set of students who like either vanilla or butterscotch or not both c) Number of students who like neither vanilla nor butterscotch
Objective	To understand the concept of linked list data structure To understand, implement linked list data structure and its operations
Outcome	After executing this assignment, Students will be able to identify and use linked list data structure for given application. Design and implement linked list data structures and its operations.
OS/Programming tools used	(64-Bit) 64-BIT Fedora 17 or latest 64-BIT Update of Equivalent Open source OS or latest 64-BIT Version and update of Microsoft Windows 7 Operating System onwards Programming Tools (64-Bit), Eclipse

Theory related to assignment:

In this assignment we will implements linked list data structure and its operations

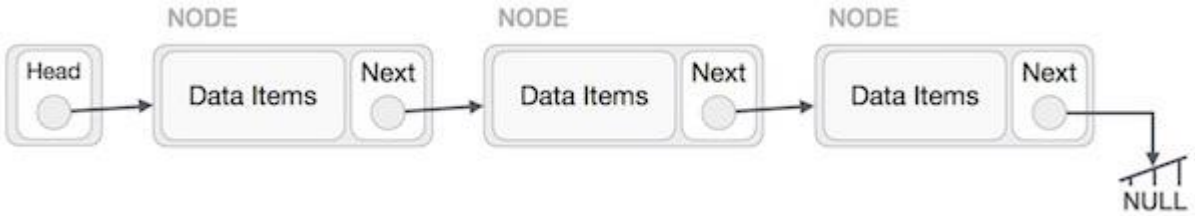
A linked list is a sequence of data structures, which are connected together via links.

Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.

Link − Each link of a linked list can store a data called an element.
Next − Each link of a linked list contains a link to the next link called Next.
LinkedList − A Linked List contains the connection link to the first link called First.

Linked List Representation

Linked list can be visualized as a chain of nodes, where every node points to the next node.



https://www.tutorialspoint.com/data_structures_algorithms/linked_list_algorithms.htm[1]

As per the above illustration, following are the important points to be considered.

1. Linked List contains a link element called first.
2. Each link carries a data field(s) and a link field called next.
3. Each link is linked with its next link using its next link.
4. Last link carries a link as null to mark the end of the list.

Types of Linked List

Following are the various types of linked list.

1. **Simple Linked List** − Item navigation is forward only.
2. **Doubly Linked List** − Items can be navigated forward and backward.
3. **Circular Linked List** − Last item contains link of the first element as next and the first element has a link to the last element as previous.

Basic Operations

Following are the basic operations supported by a list.

Insertion − Adds an element at the beginning of the list.
Deletion − Deletes an element at the beginning of the list.
Display − Displays the complete list.
Search − Searches an element using the given key.
Delete − Deletes an element using the given key.

1.Creation

Step 1 - Define a Node structure with two members data and next

Step 2 - Define a Node pointer 'head' and set it to NULL.

2.Insertion

In a single linked list, the insertion operation can be performed in three ways. They are as follows...

2.1 Inserting at Beginning of the list

2.2 Inserting at End of the list

2.3 Inserting at Specific location in the list

2.1 Inserting at Beginning of the list

We can use the following steps to insert a new node at beginning of the single linked list...

Step 1 - Create a newNode with given value.

Step 2 - Check whether list is Empty (head == NULL)

Step 3 - If it is Empty then, set newNode→next = NULL and head = newNode.

Step 4 - If it is Not Empty then, set newNode→next = head and head = newNode.

2.2 Inserting at End of the list

We can use the following steps to insert a new node at end of the single linked list...

Step 1 - Create a newNode with given value and newNode → next as NULL.

Step 2 - Check whether list is Empty (head == NULL).

Step 3 - If it is Empty then, set head = newNode.

Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.

Step 5 - Keep moving the temp to its next node until it reaches to the last node in the list (until temp → next is equal to NULL).

Step 6 - Set temp → next = newNode.

2.3 Inserting At Specific location in the list (After a Node)

We can use the following steps to insert a new node after a node in the single linked list...

Step 1 - Create a newNode with given value.

Step 2 - Check whether list is Empty (head == NULL)

Step 3 - If it is Empty then, set newNode → next = NULL and head = newNode.

Step 4 - If it is Not Empty then, define a node pointer temp and initialize with head.

Step 5 - Keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode).

Step 6 - Every time check whether temp is reached to last node or not. If it is reached to last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp to next node.

Step 7 - Finally, Set 'newNode → next = temp → next' and 'temp → next = newNode'

3. Deletion

In a single linked list, the deletion operation can be performed in three ways. They are as follows...

3.1 Deleting from Beginning of the list

3.2 Deleting from End of the list

3.3 Deleting a Specific Node

3.1 Deleting from Beginning of the list

We can use the following steps to delete a node from beginning of the single linked list...

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function

Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with head.

Step 4 - Check whether list is having only one node (temp → next == NULL)

Step 5 - If it is TRUE then set head = NULL and delete temp (Setting Empty list conditions)

Step 6 - If it is FALSE then set head = temp → next, and delete temp.

3.2 Deleting from End of the list

We can use the following steps to delete a node from end of the single linked list...

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and

terminate the function.

Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and

initialize 'temp1' with head.

Step 4 - Check whether list has only one Node (temp1 → next == NULL)

Step 5 - If it is TRUE. Then, set head = NULL and delete temp1. And terminate the

function. (Setting Empty list condition)

Step 6 - If it is FALSE. Then, set 'temp2 = temp1 ' and move temp1 to its next node. Repeat the same until it reaches to the last node in the list.

(until temp1 →next == NULL)

Step 7 - Finally, Set temp2 → next = NULL and delete temp1.

3.3 Deleting a Specific Node from the list

We can use the following steps to delete a specific node from the single linked list...

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function.

Step 3 - If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head.

Step 4 - Keep moving the temp1 until it reaches to the exact node to be deleted or to the last node. And every time set 'temp2 = temp1' before moving the 'temp1' to its next node.

Step 5 - If it is reached to the last node then display 'Given node not found in the list! Deletion not possible!!!'. And terminate the function.

Step 6 - If it is reached to the exact node which we want to delete, then check whether list is having only one node or not

Step 7 - If list has only one node and that is the node to be deleted, then set head = NULL and delete temp1 (free(temp1)).

Step 8 - If list contains multiple nodes, then check whether temp1 is the first node in the list (temp1 == head).

Step 9 - If temp1 is the first node then move the head to the next node (head = head → next) and delete temp1.

Step 10 - If temp1 is not first node then check whether it is last node in the list

(temp1 → next == NULL).

Step 11 - If temp1 is last node then set temp2 → next = NULL and

delete temp1 (free(temp1)).

Step 12 - If temp1 is not first node and not last node then set temp2 → next = temp1 →

next and delete temp1 (free(temp1)).

4. Displaying a Single Linked List

We can use the following steps to display the elements of a single linked list...

Step 1 - Check whether list is Empty (head == NULL)

Step 2 - If it is Empty then, display 'List is Empty!!!' and terminate the function.

Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with head.

Step 4 - Keep displaying temp → data with an arrow (--->) until temp reaches to the last

node

Step 5 - Finally display temp → data with arrow pointing to NULL (temp → data --->

NULL)

ADT:

Node class of each Appointment

```
class SLL_Node
{
   int start;
   int end;
   int min;
   int max;
   int flag;
  SLL_Node *next;
public: SLL_Node();//constructor
}
```

Class for SLL class:

```
Class SLL{
SLL_Node *head,*last;
public: SLL(){head=NULL, last=NULL;}//constructor
create_Shedule()   ;
display_Shedule() ;
book_App()          ;
cancel_App()        ;
```

```
            sort_App();
            }
```

Pseudo code:

1. **Function Definition to create Appointment Schedule**

```
Algorithm create_Shedule()    {
   Print("Enter the Appointment Slots:");
   Read(size);
   For i:=0 to size-1    {
     temp = new  SLL_Node;        // Step 1: Dynamic Memory Allocation
     // Step 2: Assign Data & Address
     Read(temp->start);
     Read(temp->end);
     Read(temp->min);
     Read(temp->max);
     temp->flag = 0;
     temp->next = NULL;
      if (head == NULL)      {
        head = temp;
        last = head;
     }
     else {
        last->next = temp;
        last = last->next;
     }
   }
}
```

2.Function Definition to Display Appointment Schedule

```
Algorithm display_Shedule()  {
  cnt:=0
   Print(Appointment Schdule)
```

```
  Print(“ Srno.\tStart\tEnd\tMin_Dur\tMax_Dur\tStatus");
  temp = head;
  while(temp != NULL)
  {
    Print(cnt);
    Print(temp->start);
    Print(temp->end);
    Print (temp->min);
    Print (temp->max)
    if(temp->flag)
      Print (Booked);
    else
     print(Free);
    temp = temp->next;
    cnt++;
  }
}
```

3. Function Definition to Book Appointment

```
Algorithm book_App()    {
 Print(Please enter Appointment time:);
 Read(start);
 temp = head;
 while(temp != NULL)    {
    if(start == temp->start)
    {
      if(temp->flag == 0)
      {
        Print(“Appointment Slot is Booked”);
        temp->flag = 1;
      }
      else
```

```
            print(“ Appointment Slot is not Available!!!";
        }
        temp = temp->next;
    }
}
```

4. Function Definition to Cancel Appointment

```
Algorithm cancel_App()    {
Print(“Please enter Appointment time to Cancel: ");
Read(start);
temp = head;
while(temp != NULL)  {
      if(start == temp->start)   {
         if(temp->flag == 1)     {
            print(“Your Appointment Slot is Canceled!!!");
            temp->flag = 0;
         }
         else
            print(“Your Appointment was not Booked!!!");
      }
      temp = temp->next;
   }
}
```

5.Function Definition to Sort Appointments

```
Algorithm  sort_App() {
For i:=0 to size-1   {
       temp = head;
       while(temp->next != NULL)        {
          if(temp->start > temp->next->start)
          {
              val = temp->start;
                  temp->start = temp->next->start;
```

```
            temp->next->start = val;


        val = temp->end;
            temp->end = temp->next->end;
            temp->next->end = val;


        val = temp->min;
            temp->min = temp->next->min;
            temp->next->min = val;


        val = temp->max;
            temp->max = temp->next->max;
            temp->next->max = val;


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

Time Complexity: O(n)

Space Complexity: O(n)

Conclusion:

The functions for different appointment schedule are implemented successfully using linked list data structure.

Review Questions:

1. What is Singly Linked List ?
2. What is Circular Linked List ?
3. What is Doubly Linked List ?
4. Write a function to delete a linked List ?
5. Why Quick Sort preferred for Arrays and Merge Sort for Linked List?
6. What is a Linked list? 2. Can you represent a Linked list graphically?
7. How many pointers are required to implement a simple Linked list?
8. How many types of Linked lists are there?
9. How to represent a linked list node?
10. Describe the steps to insert data at the starting of a singly linked list.
11. How to insert a node at the end of Linked list?
12. How to delete a node from linked list?
13. How to reverse a singly linked list?

14. What is the difference between singly and doubly linked lists?
15. What are the applications that use Linked lists?
16. What will you prefer to use a singly or a doubly linked lists for traversing through a list of elements?