Devanshu Surana
1032210755
PC - 12
Panel - C, Batch - C1

FDS

Lab Assignment - 5

Problem statement

Department of computer Engineering has students club named 'Pinnacle club'. students of second, third and final year of department can be granted membership or request. Similarly, one may cancel the membership of Club. First node is reserved for president of club and last node is reserved for the secretary of the club. Write C program to maintain club members information using ~~simply~~ singly linked list. Store student PRN and Name. Write functions to a) Add and delete the members as well as president and even secretary. b) Compute total number of members of club. c) Display members. d) sorting of two linked list. e) merging of two linked list. f) Reversing using three pointers.

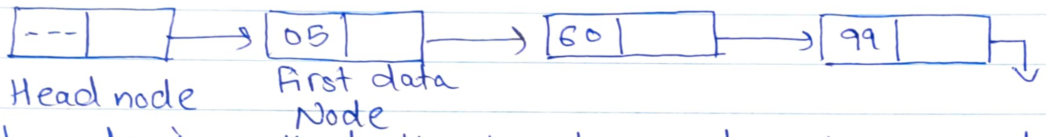
Objective

1. To study data structure : singly linked list
2. To study different operations that could be performed on SLL.
3. To study Applications of singly linked list.

Theory.

- Singly linked list

A linked list in which each node contains only one link field pointing to the next node in the list. It is unidirectional, i.e it can be transversed in only one direction from head to the last node (tail).



First node is called the header node where no data element is stored, but the link field holds the address of the node containing very first data element.

- Purpose of Head Node in singly Linked List.

→ It keeps the whole list by storing a pointer to the first node.

- Various operations on SLL:

Following are different operations that can be performed on singly linked list.

1) Create
2) Insert
3) Delete an element
4) Search an element
5) Display of SLL
6) finding length of SLL.
7) Reverse a SLL
8) Sorting.

Implementation

→ 64-bit open source Linux or its derivatives

→ Open source c programming tool like gcc/Eclipse Editor.

Pseudo Code

Write pseudo code for:

1) Create:

```
Algorithm create (*H)
{
    temp = H;
    repeat until choice = 'y'
    {
      allocate memory to curr;
      accept curr -> data;
       curr -> Next = NULL;
       temp -> next = curr;
       temp = curr;  // temp = temp -> Next
       Read choice;
       }
  }
```

2) Display

```
Algorithm display (*H)
{
  if H -> next == NULL
     print "List is empty"
  else
  {
   // print head node values
     curr = H -> next;
     while (curr != NULL)
     {
       Print curr, curr -> data, curr -> next;
       curr = curr -> next;
       }
  }
```

3) Insert:

```
Algorithm Insert by pos (* H)
{
  i = 1 ; curr = H ;
  // allocate memory for nnode;
   read nnode → data and pos ;
  // accept data & position to be inserted;
    K = len( );
  if (pos > K+1)
      // print "Data can't be inserted";
  else
  {
   while (curr~~y~~ ! = NULL && i < pos)
    {
      i++;
      curr = curr → next;
     }
      nnode → next = curr → next;
      curr → next - nnode;
     }
  }
```

4) Delete

```
Algorithm delpos (* H)
{
  prev = H; ctr = 1;
  curr = H → next;
   read pos;
 // Accept position of data to be deleted;
K= len ();
if (K< pos)
 // display Data can't be deleted;
else
{
 while (ctr < pos && curr! =NULL)
 {
   ctr++;
    prev = curr;
    curr = curr→ next;
  }
    temp = curr;
    prev→ next = curr→ next;
    curr → next = NULL;
    free (temp);
  }
}
```

5) Reverse:-

```
Algorithm reverse (* H)
{
  prev = NULL;
  curr = head→ next;
  while (curr! = NULL)
  {
   future = curr→ next;
   curr→ next = prev;
   prev = curr;
```

```
            curr = future;
        }
        head → next = prev;
      }
```

6) Sort:

```
    Algorithm sort (* H)
    {
      len = len (H);
      for i=1 to len-1
      {
        prev = H;
        curr = H → next;
        for j=0 to < 1-i
        {
            temp = curr → next;
        if (curr → data > temp → data
         {
            temp = curr → next;
         if (curr → data > temp → data
         {
              prev → next = temp;
              curr → next = temp → next;
              temp → next = curr;
              prev = temp;
         }
           else
           {
              prev = curr;
             curr = curr → next;
        }
        }
        }
        }
```

7) Merge

```
Algorithm merge (* H1, * H2)
{
  curr1 = H1 → next;
  curr2 = H2 → next;
  if (curr1 → data < curr2 → data)
  {
    temp = head1;
     Flag = 1;
   }
  else
   { temp = head2;
        flag = 0;
    }
     temp = head2;
      Flag = 0;
     }
  while (curr1 = NULL && curr2 != NULL)
   { if (curr1 → data < curr2 → data)
     {
        temp → next = curr1;
         temp = curr1;
         curr1 = curr1 → next;
        }
      else.
{
 temp → next=curr2;
 temp = curr2;
  curr2 = curr2 → next;
   }
 }
 if (curr1 == NULL)
        temp → next = curr2;
 if (curr2 == NULL)
        temp → next = curr1;
 if (flag == 1)
          display (head1);
 else
       display (head2);
  }
```

Time Complexity:

1) Create : O(n)
2) Display : O(n)
3) Delete : O(n)
4) Insert : O(n)
5) Reverse : O(n)
6) Sort : O(n^2)
7) Merge : ~~O(n log n~~) ~~O(m+n~~) ~~O(n log n~~) O(m+n)

Conclusion : Thus, implemented different operations on SLL.

FAQ's

Ans 1)

```
~~structure~~ struct x Linked list (item)
        declare  CREATE () → linked list
            insert (item, linked list) → linked list
            delete (linked list) → ~~boolean;~~ linked list
            ISEMPS (Linked list) → boolean;
      for all L ∈ Linked list , i E item let
        ISEMPS (CREATE) :: = true
        ISEMPS (insert (i, L)) :: = False
        delete (CREATE) :: = error
        delete (insert (i, L)) :: = L
        end Linked List.
```

Ans 2) i) It requires more space as pointers are also stored with information.
ii) Different amount of time is required to access each elements.
iii) We cannot traverse it from the end
iv) If we want to go to a particular element then we have to go through all those elements that came before that element.

Ans 3) It is used to implement stacks and queues which are like fundamental needs throughout computer Science.

Test Conditions:

1] Input atleast 5 nodes

Input :

PRN	Name
10	Raj
12	Rahul
68	Aishwarya
42	Riya
59	Tanaya

2] Insert an element at all position

Case (i) input : POS = 1 Name = Neha PRN = 35

Output:

35	Neha
10	Raj
12	Rahul
68	Aishwarya
42	Riya
59	Tanaya

Case (ii) Input : POS = 3 Name = Neha PRN = 35

Output :

10	Raj
12	Rahul
35	Neha
68	Aishwarya
42	Riya
59	Tanaya

case (iii) input : POS = 6 Name = Neha PRN = 35

output :

```
10 Raj
12 Rahul
68 Aishwarya
42 Riya
59 Tanaya
35 Neha
```

case (iv) Input : POS = 6 Name = Neha PRN = 35

Output :

```
Data can't be inserted
```

3] Delete an elements from all positions.

case (i) Input : POS = 1

Output :

```
12 Rahul
68 Aishwarya
42 Riya
59 Tanaya
```

case (ii) input = POS = 3

output =

```
10 Raj
12 Rahul
42 Riya
59 Tanaya
```

case (iii) Input :- POS = 5

Output :

```
10 Raj
12 Rahul
68 Aishwarya
42 Riya
```

Case (iv) Input : POS = 6

output :

```
Data can't be deleted
```

4) Reverse.

Output :

```
59 Tanaya
42 Riya
68 Aishwarya
12 Rahul
10 Raj
```

5) Sort :

output :

```
10 Raj
12 Rahul
42 Riya
59 Tanaya
68 Aishwarya
```

6] Merge:

Input:

Linked list 1

PRN	Name
10	Raj
12	Rahul
42	Riya
59	Tanaya

linked list 2

PRN	Name
82	Inayat
70	Ditee
44	Krishnaraj
22	Parth

Output:

82	Inayat
70	Ditee
44	Krishnaraj
22	Parth
10	Raj
12	Rahul
42	Riya
59	Tanaya.