Harsh Kashikaal
2003085
C21

Experiment 6

Aim: Implement Singly linked list

Theory:

Linked list can be defined as collection of objects called nodes that are randomly stored in the memory. A node contains two fields

→ Data stored at the address.

→ Pointer which contains the address of the next node in the memory.

The last node of the list contains pointer to the null.

→ Differentiate b/w Array & linked list.

	Array	Linked list
1	Random Access (Fast search)	Fast insertion/deletion time.
2	less memory needed	Dynamic size.
3	Better cache locality	Efficient memory allocation utilization.
4	Slow insertion/deletion time	Slow search time.
5	Fixed size	More memory needed per node as addition of storage required for pointers.
6	Inefficient memory allocation/utilization	

Harsh
2003085

→ Different operations:

1) Insert in beginning: Inserting element at the front of the list

Algorithm: insert_beg (val){
1. Create a new node
2. new node → data = val
3. new node → next = head
4. head = new node
}

2) Insert at End: Insertion at the last of the linked list.

Algorithm: insert at End (val){
1. Create a new node
2. new node → data = val
3. new node → next = null
4. if (head == null)
head = new node
}

3) Insert before a given node: Insertion before the specified node of the linked list

Algorithm: insert_bef (num, val){
Create new node
new nod → data = val
temp 1 = temp 2 = head
while (true){
if (temp 2 → data + 1 == val){
temp 2 = temp 1 ; break
temp 1 = tem 1 → next }
new node → next = temp 1 , temp 2 → next = new node

Harsh
2003085

4> Delete from beginning: Deletation of a node from the start of the list

Algorithm:
```
Del-beg( ){
if (head == null)
    print (underflow)
      return.
temp = head
head = head → next
free (temp)}
```

5> Delete from the End: Deletes the last node of the list.

Algorithm:
```
Del_End (){
if (head == null)
  print (underflow)
else if (head → next == null)
    temp = head.
    head = null
    free (temp) }.
else {
    temp1 = temp2 = head.
  while (temp 1 → next != null){
      temp2 = temp1
      temp1 = temp1 → next. }
   temp 2 → next = null
   free (temp)
}.
```

Harsh
2003085

6) Delete node before a specified location: Deletion of a node before the specified node in the list.

Algorithm:
```
Del-specif (num){
  if (head = null){
  print (underflow)
    return};
  temp1 = temp2 = head.
  while (temp2 → next → data != num){
    temp2 = temp1
    temp1 = temp1 → next}.
  temp2 → next = temp1 → next
  free (temp1)
}
```

7) Forward traversal: Displaying the contents is very easy.

Algorithm:
```
traversal(){
if (head == null){
  print empty.
  return}
temp = head
while (temp != null){
  print (temp → data)
  temp = temp → next
}
}
```

8> Backward traversal: Displaying the content in reverse order

Algorithm: Backtraversal(head){
1. if(head == null)
return.
2. backtraversal(head->next)
3. print(head->data);
}

9> Sorting: Sorting the data of the list using bubble sort.

Algorithm: sort(head){
temp1 = head
while(temp1->next != null){
temp2 = temp1->next;
while(temp2 != null){
if(temp1->data->temp2->data){
swap = temp1->data
temp1 = temp2->data.
temp2 = data == swap.}
temp2 = temp2->next
}
temp1 = temp1->next
}.

10> No. of nodes: Counts the number of nodes present in the list

Algorithm: no. of nodes(start){
if(head == null)
print(0)}

Harsh Kalsiwal.
2023085

```
else {
  int count;
  temp = head.
 while (temp != null)
  { count++
     Temp = Temp → next
  }
```

11> Search an Element: Searching a specified element in the list.

Algorithm:
```
search(val){
  if (head == null)
   print (empty)
  temp = head
  while (temp != null) {
   if (temp → data == val){
     print (Element found!)
     break }
  else
      temp → temp → next.
} }
```

Conclusion: Linked lists are used because of their efficient insertion and deletion. Linked lists are more efficient than arrays in memory allocation. It can grow or shrink at runtime by allocating & deallocating memory.

PROGRAM:

Write a menu driven code to implement Singly LinkedList.

Code:

```
#include <iostream>
#include <conio.h>
using namespace std;
struct node
{
    int data;
    struct node *next;
};
int flag = 0;
int count = 0;
struct node *start = NULL;
struct node *list(struct node *);
struct node *display(struct node *);
struct node *insert_beg(struct node *);
struct node *insert_end(struct node *);
struct node *insert_before(struct node *);
struct node *insert_after(struct node *);
struct node *delete_beg(struct node *);
struct node *delete_end(struct node *);
struct node *delete_node(struct node *);
struct node *delete_after(struct node *);
struct node *delete_list(struct node *);
struct node *sort_list(struct node *);
struct node *backtraversal(struct node *);
struct node *no_of_nodes(struct node *);
struct node *search(struct node *);
struct node *del_specif(struct node *);
int main(int argc, char *argv[])
{
    int option;
    do
    {
        cout << "\n1.LIST";
        cout << "\n2.INSERT IN BEGINNING";
        cout << "\n3.INSERT AT END";
        cout << "\n4.INSERT BEFORE A GIVEN NODE";
        cout << "\n5.DELETE FROM BEGINNING";
        cout << "\n6.DELETE FROM END";
        cout << "\n7.DELETE NODE BEFORE A SPECIFIED LOCATION ";
        cout << "\n8.FORWARD TRAVERSAL";
        cout << "\n9.BACKWARD TRAVERSAL";
        cout << "\n10.SORTING";
        cout << "\n11.COUNT NUMBER OF NODES";
```

```
        cout << "\n12.SEARCH AN ELEMENT";
        cout << "\n13.EXIT";
        cout << "\n\nENTER YOUR OPTION : ";
        cin >> option;
        switch (option)
        {
        case 1:
            start = list(start);
            cout << "\n--LINKED LIST CREATED--";
            break;
        case 2:
            start = insert_beg(start);
            break;
        case 3:
            start = insert_end(start);
            break;
        case 4:
            start = insert_before(start);
            break;
        case 5:
            start = delete_beg(start);
            break;
        case 6:
            start = delete_end(start);
            break;
        case 7:
            start = del_specif(start);
            break;
        case 8:
            start = display(start);
            break;
        case 9:
            start = backtraversal(start);
            break;
        case 10:
            start = sort_list(start);
            break;
        case 11:
            start = no_of_nodes(start);
            cout << "\nNUMBER OF NODES ARE : " << count;
            break;
        case 12:
            start = search(start);
            break;
        }
    } while (option != 13);
    getch();
    return 0;
```

```
}
struct node *list(struct node *start)
{
    struct node *new_node, *ptr;
    int num;
    cout << "ENTER -1 TO END!"<<endl;
    cout << "ENTER THE DATA :";
    cin >> num;
    while (num != -1)
    {
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node->data = num;
        if (start == NULL)
        {
            new_node->next = NULL;
            start = new_node;
        }
        else
        {
            ptr = start;
            while (ptr->next != NULL)
                ptr = ptr->next;
            ptr->next = new_node;
            new_node->next = NULL;
        }
        cout << "\n ENTER THE DATA : ";
        cin >> num;
    }
    return start;
}
struct node *display(struct node *start)
{
    if (flag == 1)
    {
        cout << "\n\n -- EMPTY LIST -- \n\n";
    }
    struct node *ptr;
    ptr = start;
    while (ptr != NULL)
    {
        cout << "\t "<< ptr->data;
        ptr = ptr->next;
    }
    return start;
}
struct node *search(struct node *start)
{
    int num;
```

```
    int flag = 0;
    cout << "ENTERR THE ELEMEMENT TO BE SEARCHED :";
    cin >> num;
    struct node *ptr;
    ptr = start;
    while (ptr != NULL)
    {
        if (ptr->data == num)
        {
            cout << "\n\nNUMBER IS FOUND!\n\n";
            flag = 1;
            break;
        }
        ptr = ptr->next;
    }
    if (flag != 1)
    {
        cout << "\n\nNUMBER NOT FOUND! :(\n\n";
    }
    return start;
}
struct node *no_of_nodes(struct node *start)
{
    struct node *ptr;
    ptr = start;
    while (ptr != NULL)
    {
        count = count + 1;
        ptr = ptr->next;
    }
    return start;
}
struct node *backtraversal(struct node *start)
{
    struct node *prev = NULL;
    struct node *current = start;
    struct node *nextt = NULL;
    while (current != NULL)
    {
        nextt = current->next;
        current->next = prev;
        prev = current;
        current = nextt;
    }
    start = prev;
};
struct node *insert_beg(struct node *start)
{
```

```
    struct node *new_node;
    int num;
    cout << "\n ENTER THE DATA : ";
    cin >> num;
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    new_node->next = start;
    start = new_node;
    return start;
}
struct node *insert_end(struct node *start)
{
    struct node *ptr, *new_node;
    int num;
    cout << "\n ENTER THE DATA ";
    cin >> num;
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    new_node->next = NULL;
    ptr = start;
    while (ptr->next != NULL)
        ptr = ptr->next;
    ptr->next = new_node;
    return start;
}
struct node *insert_before(struct node *start)
{
    struct node *new_node, *ptr, *preptr;
    int num, value;
    cout << "ENTER THE DATA ";
    cin >> num;
    cout << "ENTER THE VALUE BEFORE WHICH THE DATA IS TO BE INSERTED ";
    cin >> value;
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
    while (ptr->data != value)
    {
        preptr = ptr;
        ptr = ptr->next;
    }
    preptr->next = new_node;
    new_node->next = ptr;
    return start;
}
struct node *delete_beg(struct node *start)
{
    struct node *ptr;
```

```
    ptr = start;
    start = start->next;
    free(ptr);
    return start;
}
struct node *delete_end(struct node *start)
{
    struct node *ptr, *preptr;
    ptr = start;
    while (ptr->next != NULL)
    {
        preptr = ptr;
        ptr = ptr->next;
    }
    preptr->next = NULL;
    free(ptr);
    return start;
}
struct node *delete_node(struct node *start)
{
    struct node *ptr, *preptr;
    int val;
    cout << "ENTER THE VALUE TO BE DELETED";
    cin >> val;
    ptr = start;
    if (ptr->data == val)
    {
        start = delete_beg(start);
        return start;
    }
    else
    {
        while (ptr->data != val)
        {
            preptr = ptr;
            ptr = ptr->next;
        }
        preptr->next = ptr->next;
        free(ptr);
        return start;
    }
}
struct node *del_specif(struct node *start)
{
    start = no_of_nodes(start);
    int loc;
    struct node *ptr, *preptr;
    ptr = start;
```

```
    cout << "ENTER THE LOCATION ";
    cin >> loc;
    while (ptr->next != NULL)
    {
        if (loc + 1 == count)
        {
            preptr = ptr;
        }
        ptr = ptr->next;
    }
    preptr->next = ptr->next;
    free(ptr);
    return start;
}
struct node *sort_list(struct node *start)
{
    struct node *ptr1, *ptr2;
    int temp;
    ptr1 = start;
    while (ptr1->next != NULL)
    {
        ptr2 = ptr1->next;
        while (ptr2 != NULL)
        {
            if (ptr1->data > ptr2->data)
            {
                temp = ptr1->data;
                ptr1->data = ptr2->data;
                ptr2->data = temp;
            }
            ptr2 = ptr2->next;
        }
        ptr1 = ptr1->next;
    }
    return start;
}
```

OUTPUT:

```
PS D:\Harsh\SEM 3\DS\CODES> cd "d:\Harsh\SEM 3\DS\CODES\" ; if ($?) { g++ SinglyList.cpp

1.LIST
2.INSERT IN BEGINNING
3.INSERT AT END
4.INSERT BEFORE A GIVEN NODE
5.DELETE FROM BEGINNING
6.DELETE FROM END
7.DELETE NODE BEFORE A SPECIFIED LOCATION
8.FORWARD TRAVERSAL
9.BACKWARD TRAVERSAL
10.SORTING
11.COUNT NUMBER OF NODES
12.SEARCH AN ELEMENT
13.EXIT

ENTER YOUR OPTION : 1
ENTER -1 TO END!
ENTER THE DATA :12

 ENTER THE DATA : 13

 ENTER THE DATA : 14

 ENTER THE DATA : 15

 ENTER THE DATA : 16

 ENTER THE DATA : -1

--LINKED LIST CREATED--
1.LIST
2.INSERT IN BEGINNING
3.INSERT AT END
4.INSERT BEFORE A GIVEN NODE
5.DELETE FROM BEGINNING
6.DELETE FROM END
7.DELETE NODE BEFORE A SPECIFIED LOCATION
8.FORWARD TRAVERSAL
9.BACKWARD TRAVERSAL
10.SORTING
11.COUNT NUMBER OF NODES
12.SEARCH AN ELEMENT
13.EXIT

ENTER YOUR OPTION : 2
```

```
 ENTER THE DATA : 18

1.LIST
2.INSERT IN BEGINNING
3.INSERT AT END
4.INSERT BEFORE A GIVEN NODE
5.DELETE FROM BEGINNING
6.DELETE FROM END
7.DELETE NODE BEFORE A SPECIFIED LOCATION
8.FORWARD TRAVERSAL
9.BACKWARD TRAVERSAL
10.SORTING
11.COUNT NUMBER OF NODES
12.SEARCH AN ELEMENT
13.EXIT

ENTER YOUR OPTION : 3

 ENTER THE DATA 19

1.LIST
2.INSERT IN BEGINNING
3.INSERT AT END
4.INSERT BEFORE A GIVEN NODE
5.DELETE FROM BEGINNING
6.DELETE FROM END
7.DELETE NODE BEFORE A SPECIFIED LOCATION
8.FORWARD TRAVERSAL
9.BACKWARD TRAVERSAL
10.SORTING
11.COUNT NUMBER OF NODES
12.SEARCH AN ELEMENT
13.EXIT

ENTER YOUR OPTION : 4
ENTER THE DATA 20
ENTER THE VALUE BEFORE WHICH THE DATA IS TO BE INSERTED 15

1.LIST
2.INSERT IN BEGINNING
3.INSERT AT END
4.INSERT BEFORE A GIVEN NODE
5.DELETE FROM BEGINNING
6.DELETE FROM END
7.DELETE NODE BEFORE A SPECIFIED LOCATION
8.FORWARD TRAVERSAL
9.BACKWARD TRAVERSAL
```

```
10.SORTING
11.COUNT NUMBER OF NODES
12.SEARCH AN ELEMENT
13.EXIT

ENTER YOUR OPTION : 8
         18      12      13      14      20      15      16      19
1.LIST
2.INSERT IN BEGINNING
3.INSERT AT END
4.INSERT BEFORE A GIVEN NODE
5.DELETE FROM BEGINNING
6.DELETE FROM END
7.DELETE NODE BEFORE A SPECIFIED LOCATION
8.FORWARD TRAVERSAL
9.BACKWARD TRAVERSAL
10.SORTING
11.COUNT NUMBER OF NODES
12.SEARCH AN ELEMENT
13.EXIT

ENTER YOUR OPTION : 5

1.LIST
2.INSERT IN BEGINNING
3.INSERT AT END
4.INSERT BEFORE A GIVEN NODE
5.DELETE FROM BEGINNING
6.DELETE FROM END
7.DELETE NODE BEFORE A SPECIFIED LOCATION
8.FORWARD TRAVERSAL
9.BACKWARD TRAVERSAL
10.SORTING
11.COUNT NUMBER OF NODES
12.SEARCH AN ELEMENT
13.EXIT

ENTER YOUR OPTION : 6

1.LIST
2.INSERT IN BEGINNING
3.INSERT AT END
4.INSERT BEFORE A GIVEN NODE
5.DELETE FROM BEGINNING
6.DELETE FROM END
7.DELETE NODE BEFORE A SPECIFIED LOCATION
8.FORWARD TRAVERSAL
```

```
9.BACKWARD TRAVERSAL
10.SORTING
11.COUNT NUMBER OF NODES
12.SEARCH AN ELEMENT
13.EXIT

ENTER YOUR OPTION : 8
         12      13      14      20      15      16
1.LIST
2.INSERT IN BEGINNING
3.INSERT AT END
4.INSERT BEFORE A GIVEN NODE
5.DELETE FROM BEGINNING
6.DELETE FROM END
7.DELETE NODE BEFORE A SPECIFIED LOCATION
8.FORWARD TRAVERSAL
9.BACKWARD TRAVERSAL
10.SORTING
11.COUNT NUMBER OF NODES
12.SEARCH AN ELEMENT
13.EXIT

ENTER YOUR OPTION : 9

1.LIST
2.INSERT IN BEGINNING
3.INSERT AT END
4.INSERT BEFORE A GIVEN NODE
5.DELETE FROM BEGINNING
6.DELETE FROM END
7.DELETE NODE BEFORE A SPECIFIED LOCATION
8.FORWARD TRAVERSAL
9.BACKWARD TRAVERSAL
10.SORTING
11.COUNT NUMBER OF NODES
12.SEARCH AN ELEMENT
13.EXIT

ENTER YOUR OPTION : 8
         16      15      20      14      13      12
1.LIST
2.INSERT IN BEGINNING
3.INSERT AT END
4.INSERT BEFORE A GIVEN NODE
5.DELETE FROM BEGINNING
6.DELETE FROM END
7.DELETE NODE BEFORE A SPECIFIED LOCATION
8.FORWARD TRAVERSAL
```

```
9.BACKWARD TRAVERSAL
10.SORTING
11.COUNT NUMBER OF NODES
12.SEARCH AN ELEMENT
13.EXIT

ENTER YOUR OPTION : 10

1.LIST
2.INSERT IN BEGINNING
3.INSERT AT END
4.INSERT BEFORE A GIVEN NODE
5.DELETE FROM BEGINNING
6.DELETE FROM END
7.DELETE NODE BEFORE A SPECIFIED LOCATION
8.FORWARD TRAVERSAL
9.BACKWARD TRAVERSAL
10.SORTING
11.COUNT NUMBER OF NODES
12.SEARCH AN ELEMENT
13.EXIT

ENTER YOUR OPTION : 11

NUMBER OF NODES ARE : 6
1.LIST
2.INSERT IN BEGINNING
3.INSERT AT END
4.INSERT BEFORE A GIVEN NODE
5.DELETE FROM BEGINNING
6.DELETE FROM END
7.DELETE NODE BEFORE A SPECIFIED LOCATION
8.FORWARD TRAVERSAL
9.BACKWARD TRAVERSAL
10.SORTING
11.COUNT NUMBER OF NODES
12.SEARCH AN ELEMENT
13.EXIT

ENTER YOUR OPTION : 12
ENTERR THE ELEMEMENT TO BE SEARCHED :16


NUMBER IS FOUND!


1.LIST
2.INSERT IN BEGINNING
```

```
3.INSERT AT END
4.INSERT BEFORE A GIVEN NODE
5.DELETE FROM BEGINNING
6.DELETE FROM END
7.DELETE NODE BEFORE A SPECIFIED LOCATION
8.FORWARD TRAVERSAL
9.BACKWARD TRAVERSAL
10.SORTING
11.COUNT NUMBER OF NODES
12.SEARCH AN ELEMENT
13.EXIT

ENTER YOUR OPTION : 13
```