K. J. Somaiya College of Engineering, Mumbai

(A Constituent College of Somaiya Vidyavihar University)

Department of Computer Engineering

Batch:_A3 Roll No.:16010121051

Experiment No.

Grade: AA / AB / BB / BC / CC / CD /DD

Title: Implementation of Linked List

Objective: To understand the use of linked list as data structures for various application.

Expected Outcome of Experiment:

CO	Outcome
CO 2	Apply linear and non-linear data structure in application development.

Books/ Journals/ Websites referred:

Mam's ppt

Introduction:

Define Linked List

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.

Types of linked list:

- Singly linked lists
- Doubly linked lists
- Circular linked lists
- Circular doubly linked lists

Algorithm for creation, insertion, deletion, traversal and searching an element in assigned linked list type:

Implementation of an application using linked list:

Code:

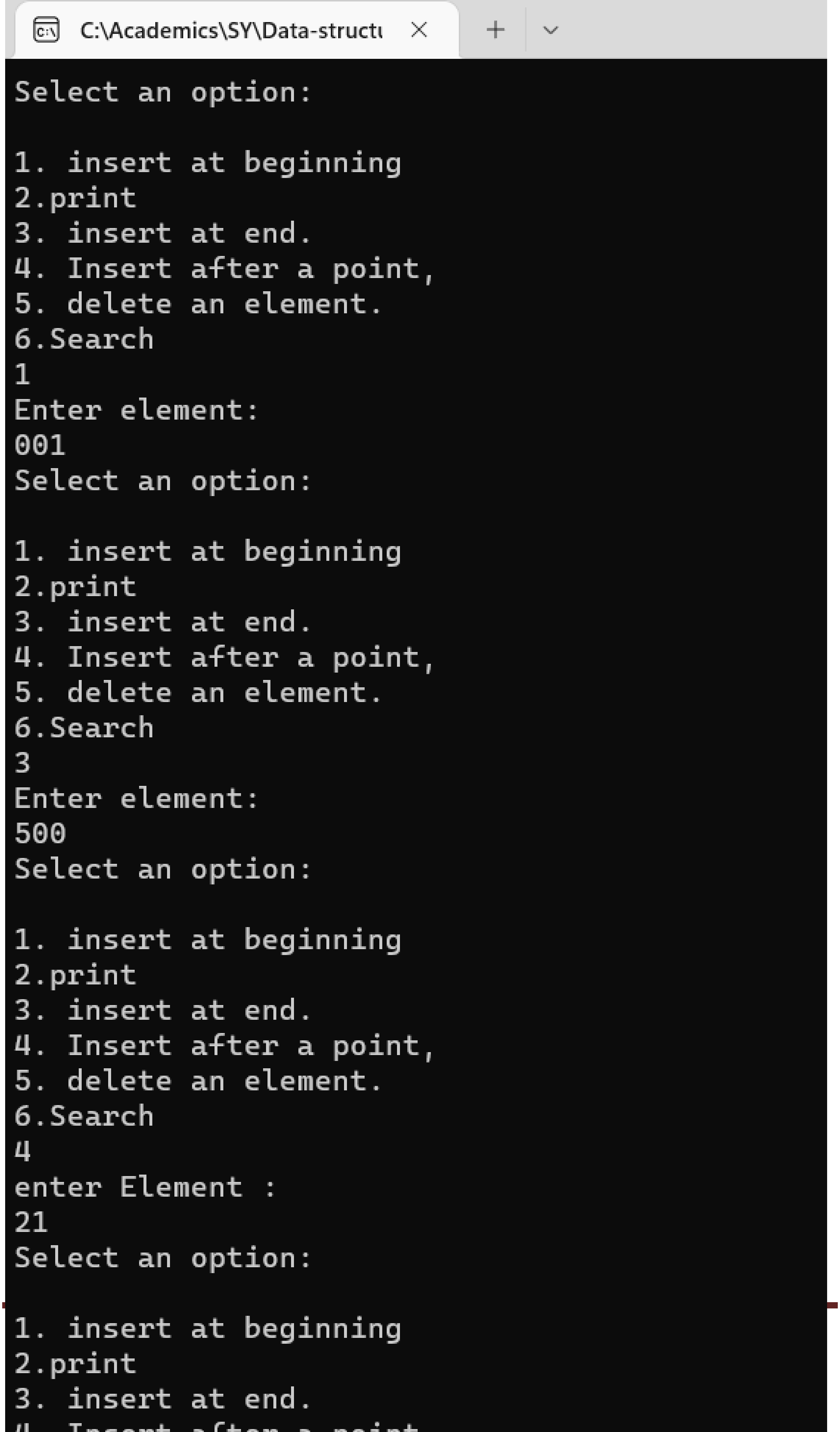
```
#include <stdlib.h>
#include <iostream>
using namespace std;

struct Node {
  int data;
  struct Node* next;
};

void insertAtBeginning(struct Node** head_ref, int new_data) {
  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));

  new_node->data = new_data;
  new_node->next = (*head_ref);

  (*head_ref) = new_node;
}

void insertAfter(struct Node* prev_node, int new_data) {
  if (prev_node == NULL) {
  cout << "the given previous node cannot be NULL";
  return;
  }

  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
  new_node->data = new_data;
  new_node->next = prev_node->next;
  prev_node->next = new_node;
}

void insertAtEnd(struct Node** head_ref, int new_data) {
  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
  struct Node* last = *head_ref; /* used in step 5*/

  new_node->data = new_data;
  new_node->next = NULL;

  if (*head_ref == NULL) {
  *head_ref = new_node;
  return;
  }

  while (last->next != NULL) last = last->next;

  last->next = new_node;
  return;
}

void deleteNode(struct Node** head_ref, int key) {
  struct Node *temp = *head_ref, *prev;

  if (temp != NULL && temp->data == key) {
  *head_ref = temp->next;
  free(temp);
  return;
  }
  while (temp != NULL && temp->data != key) {
  prev = temp;
  temp = temp->next;
  }

  if (temp == NULL) return;
  prev->next = temp->next;
```

```
  free(temp);
}
bool searchNode(struct Node** head_ref, int key) {
  struct Node* current = *head_ref;

  while (current != NULL) {
  if (current->data == key) return true;
  current = current->next;
  }
  return false;
}

void sortLinkedList(struct Node** head_ref) {
  struct Node *current = *head_ref, *index = NULL;
  int temp;

  if (head_ref == NULL) {
  return;
  } else {
  while (current != NULL) {
    index = current->next;

    while (index != NULL) {
    if (current->data > index->data) {
      temp = current->data;
      current->data = index->data;
      index->data = temp;
    }
    index = index->next;
    }
    current = current->next;
  }
  }
```

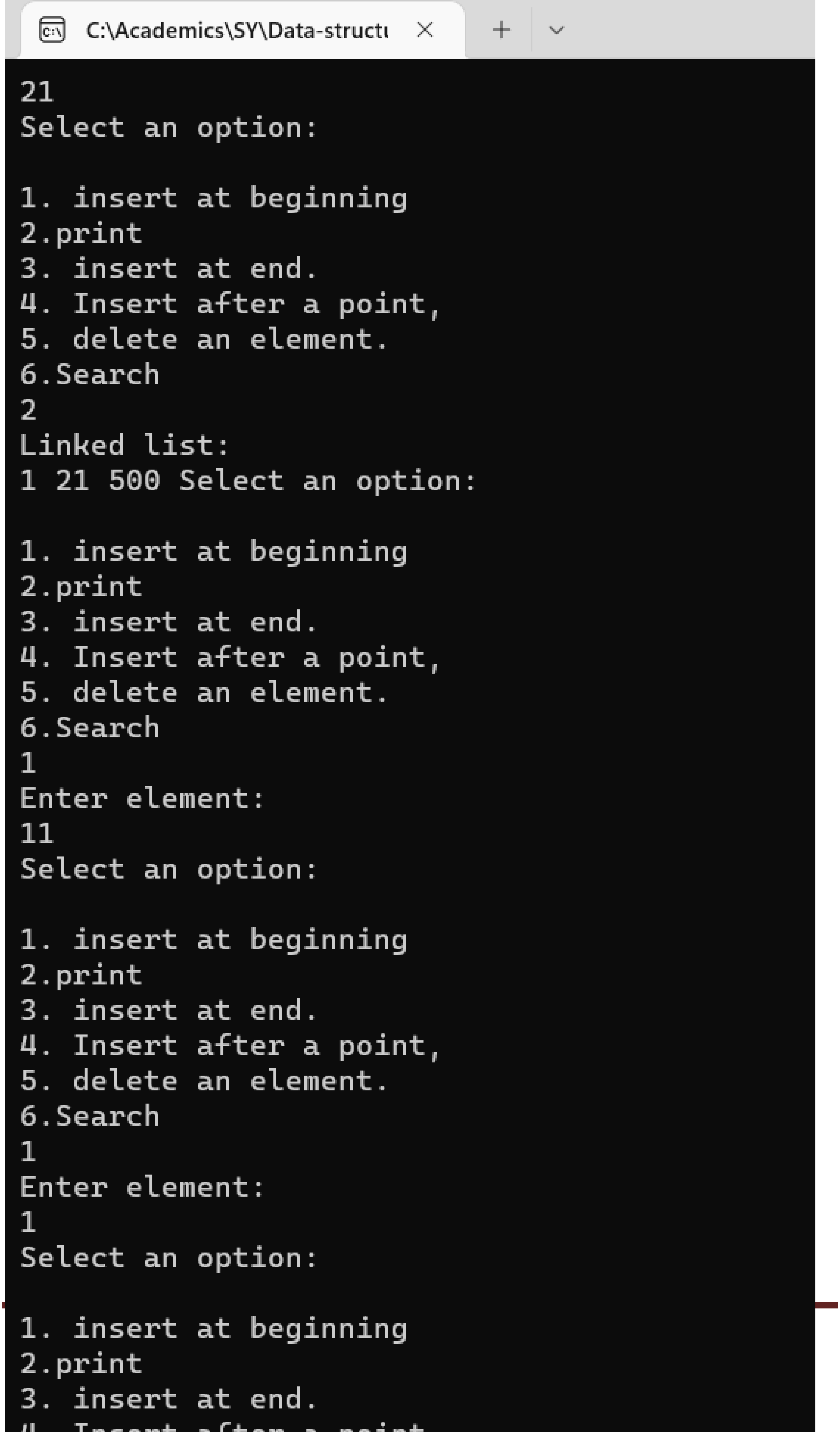
```
}
void printList(struct Node* node) {
  while (node != NULL) {
  cout << node->data << " ";
  node = node->next;
  }
}
int main() {
  struct Node* head = NULL;
  int n,a,b,c,d,e;
while(1){

cout<<"Select an option: "<<endl;

cout<<"\n1. insert at beginning \n";
cout<<"2.print \n";
cout<<"3. insert at end. \n";
cout<<"4. Insert after a point, \n";
cout<<"5. delete an element. \n";
cout<<"6.Search \n";
cin>>n;
switch(n)
{
            case 1 : cout<<"Enter element: \n";
                             cin>>a;
                             insertAtBeginning(&head, a);
                             break;
            case 2 : cout << "Linked list: \n";
                             sortLinkedList(&head);
                             printList(head);
                             break;
        case 3 : cout<<"Enter element: \n";
                     cin>>b;
```

```
                insertAtEnd(&head, b);
                break;
    case 4 : cout<<"enter Element : \n";
                cin>>c;
                insertAfter(head->next, c);
                break;
    case 5 : cout<<"Enter element u wanna delete: \n";
                cin>>d;
                deleteNode(&head, d);
                break;
    case 6 : cout<<"Enter element u wanna search: \n";
                cin>>e;
                if (searchNode(&head, e)) {
                        cout <<  " Found";
                        } else {
                        cout <<  "Not Found";
                        }
                        break;

}}}
```

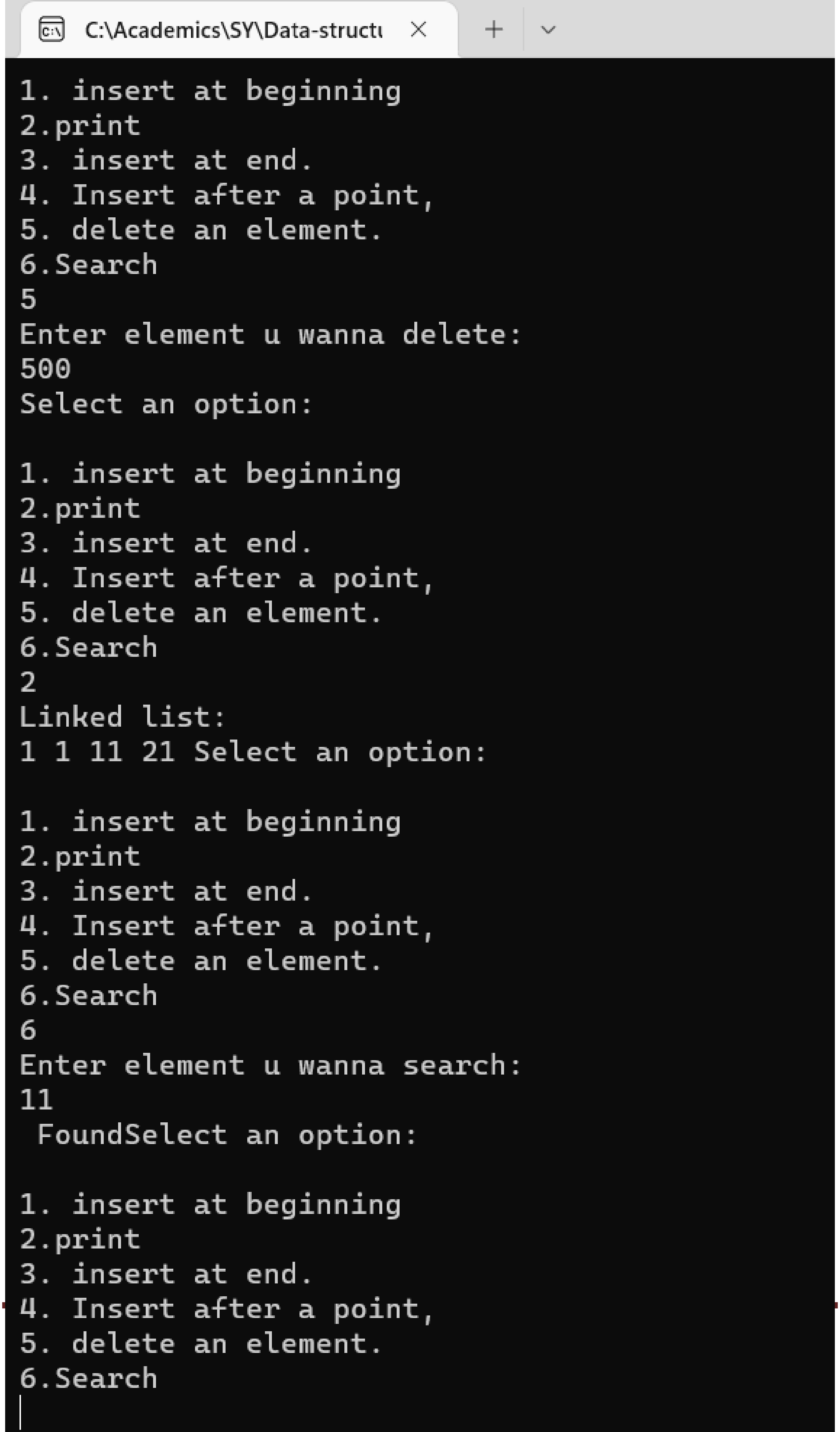
Output:



```
Select an option:

1. insert at beginning
2.print
3. insert at end.
4. Insert after a point,
5. delete an element.
6.Search
1
Enter element:
001
Select an option:

1. insert at beginning
2.print
3. insert at end.
4. Insert after a point,
5. delete an element.
6.Search
3
Enter element:
500
Select an option:

1. insert at beginning
2.print
3. insert at end.
4. Insert after a point,
5. delete an element.
6.Search
4
enter Element :
21
Select an option:

1. insert at beginning
2.print
3. insert at end.
4. Insert after a point,
```



```
21
Select an option:

1. insert at beginning
2.print
3. insert at end.
4. Insert after a point,
5. delete an element.
6.Search
2
Linked list:
1 21 500 Select an option:

1. insert at beginning
2.print
3. insert at end.
4. Insert after a point,
5. delete an element.
6.Search
1
Enter element:
11
Select an option:

1. insert at beginning
2.print
3. insert at end.
4. Insert after a point,
5. delete an element.
6.Search
1
Enter element:
1
Select an option:

1. insert at beginning
2.print
3. insert at end.
```



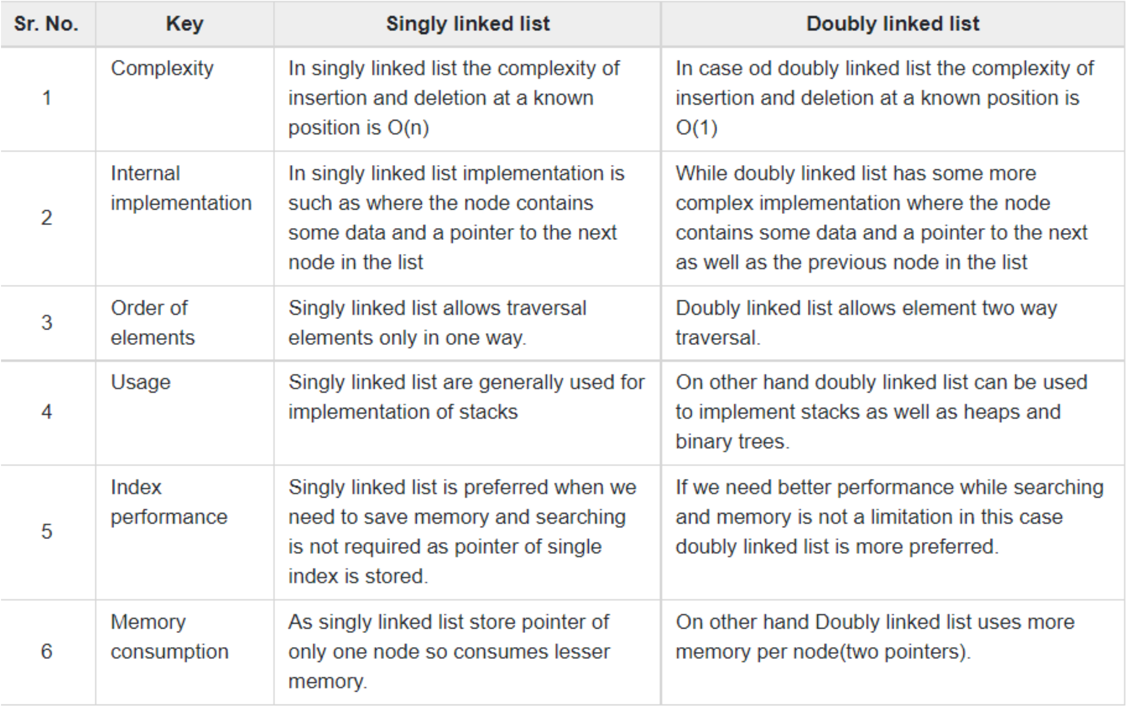
```
1. insert at beginning
2.print
3. insert at end.
4. Insert after a point,
5. delete an element.
6.Search
5
Enter element u wanna delete:
500
Select an option:

1. insert at beginning
2.print
3. insert at end.
4. Insert after a point,
5. delete an element.
6.Search
2
Linked list:
1 1 11 21 Select an option:

1. insert at beginning
2.print
3. insert at end.
4. Insert after a point,
5. delete an element.
6.Search
6
Enter element u wanna search:
11
 FoundSelect an option:

1. insert at beginning
2.print
3. insert at end.
4. Insert after a point,
5. delete an element.
6.Search
```

Conclusion:-

Linked list was implemented successfully and various operations in linked list were also performed.

Post lab questions:

1. Compare and contrast SLL and DLL



Sr. No.	Key	Singly linked list	Doubly linked list
1	Complexity	In singly linked list the complexity of insertion and deletion at a known position is O(n)	In case od doubly linked list the complexity of insertion and deletion at a known position is O(1)
2	Internal implementation	In singly linked list implementation is such as where the node contains some data and a pointer to the next node in the list	While doubly linked list has some more complex implementation where the node contains some data and a pointer to the next as well as the previous node in the list
3	Order of elements	Singly linked list allows traversal elements only in one way.	Doubly linked list allows element two way traversal.
4	Usage	Singly linked list are generally used for implementation of stacks	On other hand doubly linked list can be used to implement stacks as well as heaps and binary trees.
5	Index performance	Singly linked list is preferred when we need to save memory and searching is not required as pointer of single index is stored.	If we need better performance while searching and memory is not a limitation in this case doubly linked list is more preferred.
6	Memory consumption	As singly linked list store pointer of only one node so consumes lesser memory.	On other hand Doubly linked list uses more memory per node(two pointers).