SYCOA68

Name : Vedant Ghumade

Assignment – 2

Write a C++ program to implement a singly link list and perform operations such as insert, delete, display, search element from it and reverse the list.

Program :

```
#include<iostream>
using namespace std;

struct node
{
   int info;
   struct node *next;
}*start;

class single_llist
{
   public:
      node* create_node(int);
      void insert_begin();
      void insert_pos();
      void insert_last();
```

```
        void delete_pos();
        void search();
        void update();
        void reverse();
        void display();
        single_llist()
        {
            start = NULL;
        }
};


int main()
{
    int choice, nodes, element, position, i;
    single_llist sl;
    start = NULL;
    while (1)
    {
        cout<<endl<<"--------------------------------"<<endl;
        cout<<endl<<"Operations on singly linked list"<<endl;
        cout<<endl<<"--------------------------------"<<endl;
        cout<<"1.Insert Node at beginning"<<endl;
        cout<<"2.Insert node at last"<<endl;
```

```
cout<<"3.Insert node at position"<<endl;
cout<<"4.Delete a Particular Node"<<endl;
cout<<"5.Update Node Value"<<endl;
cout<<"6.Search Element"<<endl;
cout<<"7.Display Linked List"<<endl;
cout<<"8.Reverse Linked List "<<endl;
cout<<"9.Exit "<<endl;
cout<<"Enter your choice : ";
cin>>choice;
switch(choice)
{
case 1:
   cout<<"Inserting Node at Beginning: "<<endl;
   sl.insert_begin();
   cout<<endl;
   break;
case 2:
   cout<<"Inserting Node at Last: "<<endl;
   sl.insert_last();
   cout<<endl;
   break;
case 3:
   cout<<"Inserting Node at a given position:"<<endl;
   sl.insert_pos();
```

```
    cout<<endl;
    break;
case 4:
    cout<<"Delete a particular node: "<<endl;
    sl.delete_pos();
    break;
case 5:
    cout<<"Update Node Value:"<<endl;
    sl.update();
    cout<<endl;
    break;
case 6:
    cout<<"Search element in Link List: "<<endl;
    sl.search();
    cout<<endl;
    break;
case 7:
    cout<<"Display elements of link list"<<endl;
    sl.display();
    cout<<endl;
    break;
case 8:
    cout<<"Reverse elements of Link List"<<endl;
    sl.reverse();
```

```
            cout<<endl;
            break;
        case 9:
            cout<<"Exiting..."<<endl;
            exit(1);
            break;
        default:
            cout<<"Wrong choice"<<endl;
        }
    }
}

node *single_llist::create_node(int value)
{
    struct node *temp, *s;
    temp = new(struct node);
    if (temp == NULL)
    {
        cout<<"Memory not allocated "<<endl;
        return 0;
    }
    else
    {
        temp->info = value;
        temp->next = NULL;
        return temp;
    }
}

void single_llist::insert_begin()
{
    int value;
    cout<<"Enter the value to be inserted: ";
    cin>>value;
    struct node *temp, *p;
    temp = create_node(value);
    if (start == NULL)
    {
        start = temp;
        start->next = NULL;
    }
    else
    {
        p = start;
        start = temp;
        start->next = p;
    }
    cout<<"Element Inserted at beginning"<<endl;
```

```
}

void single_llist::insert_last()
{
    int value;
    cout<<"Enter the value to be inserted: ";
    cin>>value;
    struct node *temp, *s;
    temp = create_node(value);
    s = start;
    while (s->next != NULL)
    {
        s = s->next;
    }
    temp->next = NULL;
    s->next = temp;
    cout<<"Element Inserted at last"<<endl;
}


void single_llist::insert_pos()
{
    int value, pos, counter = 0;
    cout<<"Enter the value to be inserted: ";
```

```
cin>>value;
struct node *temp, *s, *ptr;
temp = create_node(value);
cout<<"Enter the postion at which node to be inserted: ";
cin>>pos;
int i;
s = start;
while (s != NULL)
{
   s = s->next;
   counter++;
}
if (pos == 1)
{
   if (start == NULL)
   {
      start = temp;
      start->next = NULL;
   }
   else
   {
      ptr = start;
      start = temp;
      start->next = ptr;
```

```
        }
    }
    else if (pos > 1  && pos <= counter)
    {
        s = start;
        for (i = 1; i < pos; i++)
        {
            ptr = s;
            s = s->next;
        }
        ptr->next = temp;
        temp->next = s;
    }
    else
    {
        cout<<"Positon out of range"<<endl;
    }
}

void single_llist::delete_pos()
{
    int pos, i, counter = 0;
    if (start == NULL)
    {
```

```
    cout<<"List is empty"<<endl;
    return;
}
cout<<"Enter the position of value to be deleted: ";
cin>>pos;
struct node *s, *ptr;
s = start;
if (pos == 1)
{
    start = s->next;
}
else
{
    while (s != NULL)
    {
        s = s->next;
        counter++;
    }
    if (pos > 0 && pos <= counter)
    {
        s = start;
        for (i = 1;i < pos;i++)
        {
            ptr = s;
            s = s->next;
          }
          ptr->next = s->next;
       }
       else
       {
          cout<<"Position out of range"<<endl;
       }
       free(s);
       cout<<"Element Deleted"<<endl;
    }
}


void single_llist::update()
{
    int value, pos, i;
    if (start == NULL)
    {
       cout<<"List is empty"<<endl;
       return;
    }
    cout<<"Enter the node postion to be updated: ";
    cin>>pos;
    cout<<"Enter the new value: ";
    cin>>value;
    struct node *s, *ptr;
    s = start;
    if (pos == 1)
    {
        start->info = value;
    }
    else
    {
        for (i = 0;i < pos - 1;i++)
        {
            if (s == NULL)
            {
                cout<<"There are less than "<<pos<<" elements";
                return;
            }
            s = s->next;
        }
        s->info = value;
    }
    cout<<"Node Updated"<<endl;
}
```

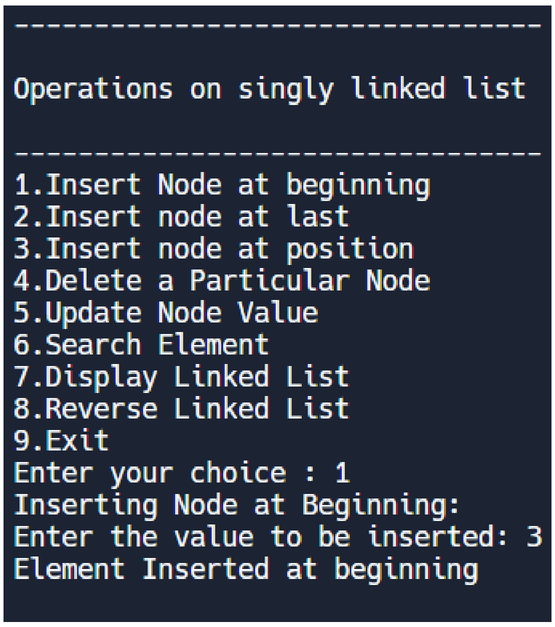
```
void single_llist::search()
{
    int value, pos = 0;
    bool flag = false;
    if (start == NULL)
    {
        cout<<"List is empty"<<endl;
        return;
    }
    cout<<"Enter the value to be searched: ";
    cin>>value;
    struct node *s;
    s = start;
    while (s != NULL)
    {
        pos++;
        if (s->info == value)
        {
            flag = true;
            cout<<"Element "<<value<<" is found at position
"<<pos<<endl;
        }
        s = s->next;
    }
    if (!flag)
        cout<<"Element "<<value<<" not found in the list"<<endl;
}


void single_llist::reverse()
{
    struct node *ptr1, *ptr2, *ptr3;
    if (start == NULL)
    {
        cout<<"List is empty"<<endl;
        return;
    }
    if (start->next == NULL)
    {
        return;
    }
    ptr1 = start;
    ptr2 = ptr1->next;
    ptr3 = ptr2->next;
    ptr1->next = NULL;
    ptr2->next = ptr1;
    while (ptr3 != NULL)
    {
        ptr1 = ptr2;
        ptr2 = ptr3;
        ptr3 = ptr3->next;
        ptr2->next = ptr1;
    }
    start = ptr2;
}


void single_llist::display()
{
    struct node *temp;
    if (start == NULL)
    {
        cout<<"The List is Empty"<<endl;
        return;
    }
    temp = start;
    cout<<"Elements of list are: "<<endl;
    while (temp != NULL)
    {
        cout<<temp->info<<"->";
        temp = temp->next;
```

```
   }
   cout<<"NULL"<<endl;
}
```

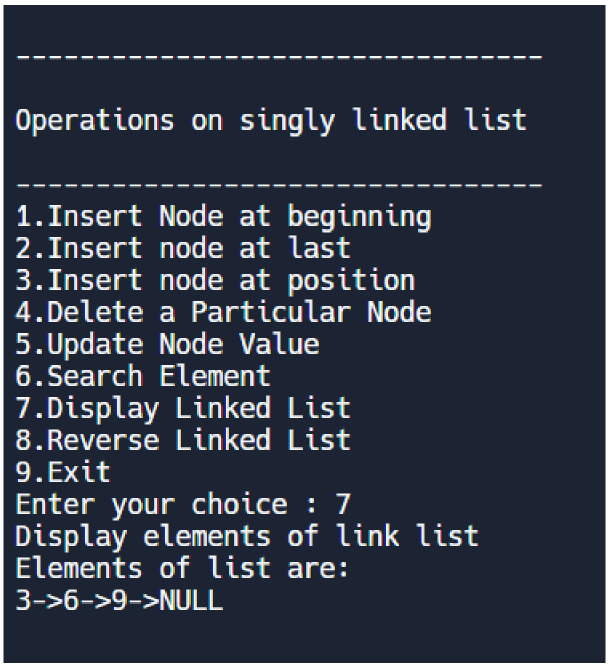
Output :



```
--------------------------------

Operations on singly linked list

--------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Delete a Particular Node
5.Update Node Value
6.Search Element
7.Display Linked List
8.Reverse Linked List
9.Exit
Enter your choice : 1
Inserting Node at Beginning:
Enter the value to be inserted: 3
Element Inserted at beginning
```



```
--------------------------------

Operations on singly linked list

--------------------------------
1.Insert Node at beginning
2.Insert node at last
3.Insert node at position
4.Delete a Particular Node
5.Update Node Value
6.Search Element
7.Display Linked List
8.Reverse Linked List
9.Exit
Enter your choice : 7
Display elements of link list
Elements of list are:
3->6->9->NULL
```