Assignment number -14 .**Name** Roshan Chongder . **Roll Number** – CSE/17/65

Problem-
Write a program to implement single linked_list using new and delete operator .

2.**Program Code**
It's the header file containing the linked_list and the node class(to implement the linked list)
[named as template_linked_list.h]

```
#include<malloc.h>
#include<iostream>
using namespace std ;
//class node
template<typename type>
class node{
  public :
  type data;
  node *next ;

  node(){ cout<<"In the do nothing constructor of node."<<endl;}
  //paramterized
  node(type x){
    cout<<"In paramterixed constructor of node for"<<this<<endl;
    data=x; next=NULL;  }
  //destructor for node
  ~node(){cout<<"In destructor call for node"<<this<<endl;}
   //new operator overloaded for node
  void* operator new (size_t size){
    cout<<"In overloaded new for node"<<endl;
    void *ptr=malloc(size);
    return ptr;}
  //delete operator overload for node
  void operator delete(void *ptr){
    cout<<"In overloaded delete for node "<<endl;
    free(ptr);}
} ;



//class linked_list
template<typename type>
class linked_list{
  public :
  node<type> *first,*end ;

  //constructor of the linked_list
  linked_list(int x=0,type *list=NULL){
    //x is the number of nodes  and list to initialize the linked_list
    first=end=NULL;
    for(int i=1;i<=x;i++){
      node<type> *temp;
      if(list!=NULL)
      temp=new node<type>(list[i-1]);
      else
      temp=new node<type>();
```

```
        if(first==NULL)
        first=end=temp;
        else{
           end->next=temp;
           end=temp;}
      }
   }
    //destrctor of the linked_list
   ~linked_list(){
    cout<<"In destrucotor call of Linked_list for "<<this<<endl;
    while(len()!=0){
       node<type> *d=first;
       first=first->next;
       delete  d ;   }
   }
   //to get the length of the linked_list
   int len();
   //overloading the new operotor for linked_list
   void* operator new(size_t size){
      void* ptr=malloc(size);
      return ptr; }
   //overloading delete[] for linked_list
   void operator delete(void *ptr){
      cout<<"In overloaded delete of Linked_list"<<endl;
      free(ptr);  }

   //operator overloading of []
   type& operator[](int x){
     // cout<<"Ok i am in"<<endl;
      if(len()<x){
         cout<<"Index out of bound ..."<<endl;
         return first->data ;    }
      else{
      //    cout<<"In the else block "<<endl;
         node<type>* temp=first ;
         while(x){
            temp=temp->next ;
            x--;    }
         return temp->data;  }
   }

    //function to delete a node (depending on index or data)
   void del_index(int x) ;
   void del_data(type x);

   //function to show the linked_list
   void show() ;

   //appending some nodes in the linked list
   void append(type data) ;
   void append(type data,int index) ;
   void append(linked_list<type> *ptr);
   //function to sort the linked_list
   void sort() ;
};

template<typename type>
void linked_list<type>:: append(type data,int index){
      if(index>len())
      cout<<"Such index does not exist"<<endl;
      else{
         node<type> *t=new node<type>(data);
```

```
        if(index==len())//insertion at the end
        end->next=t;
        else{
           node<type> *temp=first;
           node<type> *pre=NULL;
           while(index){
              pre=temp ;
              temp=temp->next;
              index--;    }
           pre->next=t ;
           t->next=temp;   }
     }
}
template<typename type>
void linked_list<type> :: sort(){
     //applying inertion sort
     for(int i=1;i<=len()-1;i++){
        type temp=(*this)[i];
        int j=i-1;
        while(j>=0 && temp<(*this)[j]){
           (*this)[j+1]=(*this)[j];
           j--;    }
        (*this)[j+1]=temp;  }
}

template<typename type>
void linked_list<type> :: append(linked_list<type> *ptr){
        if(first==NULL)
        first=end=ptr->first;
        else{
           end->next=ptr->first; //last node is pointing to the newly created node n
           while(end->next!=NULL)
           end=end->next ;    }
}

template<typename type>
void linked_list<type> :: append(type data)
{
     node<type> *n=new node<type>(data);
     if(n!=NULL){
        if(first==NULL)
        first=end=n;
        else{
           end->next=n; //last node is pointing to the newly created node n
           end=n; //now end is pointing to  n     }
   }
     else
     cout<<"Allocation failed ."<<endl;
}
template<typename type>
void linked_list<type> ::  show()
{
     cout<<endl;
     node<type> *temp=first;
     if(temp==NULL){
        cout<<"Linked_list is empty "<<endl;
        return ;    }
     while(temp->next!=NULL){
        cout<<temp->data<<"-->";
        temp=temp->next;    }
     cout<<temp->data<<endl;
     cout<<endl;}
```

```
template<typename type>
void linked_list<type> :: del_data(type x){
      //finding the data
      node<type>* temp=first ;
      node<type> *pre=NULL;
      while(temp!=NULL){
        if(temp->data==x)
        break;
        else{
          pre=temp;
          temp=temp->next ;   }
      }
      if(temp==NULL)
      cout<<"data not found"<<endl;
      else{
        //means the data is found
        if(pre==NULL)//found at the first node{
          cout<<"Foud "<<x<<" at the first node."<<endl;
          first=temp->next ;}
        else if(temp->next==NULL){
          cout<<"Foudn"<<x<<" at the last node "<<endl;
          pre->next=NULL;}
        else{
          cout<<"Found "<<x<<" in some where inter-midiate node."<<endl;
          pre->next=temp->next ;  }
        delete temp ;   }
}

template<typename type>
void linked_list<type> :: del_index(int x){
        if(x>=len())
        cout<<"Such index does not exist"<<endl;
        else{
          node<type> *temp=first;
          node<type> *pre=NULL;
          while(x){
            pre=temp ;
            temp=temp->next;
            x--;}
          pre->next=temp->next ;
          delete temp;    }
}
template<typename type>
int linked_list<type> :: len(){
      node<type>* temp=first;
      int count=0;
      while(temp!=NULL){
        count++ ;
        temp=temp->next ;   }
      return count ;
}
```

MAIN .CPP FILE TO EXECUTE

```
#include"template_linked_list.h"
//using namespace std ;

int main()
{
   char arr[]={'a','c','b'}; //to initialize the linked_list
   linked_list<char> *ptr=new linked_list<char>(3,arr);
   ptr->show();
   ptr->append('x');
   ptr->show();
   ptr->append('z',1);
  ptr->append('q',5);
   ptr->show();
   ptr->sort();
   ptr->show();
   delete ptr;

return 0;
}
```

Output -

```
In overloaded new for node
In paramterixed constructor of node for0x564916d902a0
In overloaded new for node
In paramterixed constructor of node for0x564916d902c0
In overloaded new for node
In paramterixed constructor of node for0x564916d902e0
a-->c-->b
In overloaded new for node
In paramterixed constructor of node for0x564916d90300
a-->c-->b-->x
In overloaded new for node
In paramterixed constructor of node for0x564916d90320
In overloaded new for node
In paramterixed constructor of node for0x564916d90340
```

a-->z-->c-->b-->x-->q

a-->b-->c-->q-->x-->z

In destrucotor call of Linked_list for 0x564916d8fe70

In destructor call for node0x564916d902a0

In overloaded delete for node

In destructor call for node0x564916d90320

In overloaded delete for node

In destructor call for node0x564916d902c0

In overloaded delete for node

In destructor call for node0x564916d902e0

In overloaded delete for node

In destructor call for node0x564916d90300

In overloaded delete for node

In destructor call for node0x564916d90340

In overloaded delete for node

In overloaded delete of Linked_list