JASKIRAT SINGH (2020CSC1008)

program to execute various functions on circular linked list

Time Complexities

- Creation of Node - O(1)
- insert_at_beg - O(1)
- insert_at_end – O(1)
- insert_at_pos – O(n)
- delete_from_beg – O(1)
- delete_from_end – O(1)
- delete_from_pos – O(n)
- delete_by_elem – O(n)
- search – O(n)
- display – O(n)
- reverse – O(n)
- merge – O(1)
- union – O(ab)
- intersection – O(ab)
- middle element – O(n/2)

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

/************************** NODE CLASS ***************************/
//a node class to make nodes for linked list
template <class T>
class node
```

```
{
public:
   T data = 0;
   node<T> *next = NULL;
   node<T> *prev = NULL;

   node(int data)
   {
      this->data = data;
   }
};

/************************ Circular Linked List Class ************************/
//a node class to create nodes
template <class T>
class linkedlist
{
private:
   node<T> *head = NULL;
   node<T> *tail = NULL;
   int count = 0;

public:
   linkedlist() {}
   ~linkedlist() {}
   node<T> *createNode(T d);
   void createlinkedlist();
   void printslinkedlist();

   //insertion methods
   void insertion();
```

```
void insert_at_beg(T e);
void insert_at_end(T e);
void insert_at_pos(T e);

//deletion methods
void deletion();
void delete_from_beg();
void delete_from_end();
void delete_from_pos();
void delete_by_elem();

//serach element and return node
node<T> *returnNode();

//reverse the linked list
void reverselinkedlist();

//merges two linked lists
void mergeMenu(linkedlist<T> &l);
void mergefunc(linkedlist<T> l);
linkedlist<T> operator+(linkedlist<T> l);

//method for the union of two linked lists
void unionOfLl(linkedlist<T> l);

//method for the intersection of two linked lists
void intersection(linkedlist<T> l);

//finds middle element of linked list without using count
void middleElem();
```

```
    //main menu for all the methods
    void Menu();
};

//method that creates linked list by taking data from the user
template <class T>
void linkedlist<T>::createlinkedlist()
{
    T element = 0;
    cout << "\nEnter the number of elements in your Linked List: ";
    cin >> count;

    if (count == 0)
        return;

    node<T> *copy = NULL;
    cout << "Enter the Elements in your linked list one by one :-\n";
    for (int i = 0; i < count; i++)
    {
        cin >> element;
        if (i == 0)
        {
            head = new node<T>(element);
            head->prev = tail;
            copy = head;
            tail = head;
        }
        else
        {
            node<T> *tmp = new node<T>(element);
            tmp->prev = copy;
```

```
            copy->next = tmp;
            copy = tmp;
            tail = tmp;
        }
    }
    head->prev = tail;
    tail->next = head;

    //calling print method to print linked list
    printslinkedlist();
}

//method to create node for linked list
template <class T>
node<T> *linkedlist<T>::createNode(T d)
{
    node<T> *temp = new node<T>(d);
    temp->next = NULL;
    temp->prev = NULL;

    return temp;
}

//method to create entire linked list
template <class T>
void linkedlist<T>::printslinkedlist()
{
    if (count == 0)
    {
        cout << "\n|***|  There is nothing to show, linked list is empty |***|\n";
        return;
```

```
    }

    //Printing the Linked list
    node<T> *t = head;
    cout << "\n***| Elements in the linked list are : \n";
    do
    {
        cout << t->data << "\t";
        t = t->next;

    } while (t != head);
    cout << endl;

    //printing the linked list in reverse order
    // node<T> *t1 = tail;
    // cout << "\n***| Printing linked list in Reverse order: \n";
    // while (t1 != NULL)
    // {
    //     cout << t1->data << "\t";
    //     t1 = t1->prev;
    // }
    // cout << endl;

    cout << "| First Element: " << head->data << " | Last Element: " << tail->data << " |\n";
    cout << "| Head->prev->data: " << head->prev->data << " | Tail->next->data: " << tail->next->data
<< " |\n";
}

//method to show menu for insertion
template <class T>
void linkedlist<T>::insertion()
```

```
{
    T e = 0;
    int ch;
    char selc;
    do
    {
        //taking input for element
        cout << "\nEnter the element you want to insert: \n";
        cin >> e;

        //option for the menu
        cout << "\n|><| Insertion Menu |><|\n";
        cout << "Choose any one of the option: \n";
        cout << "1. Insert At Begining\n";
        cout << "2. Insert At End\n";
        cout << "3. Insert At Position\n";
        cin >> ch;

        switch (ch)
        {
        case 1:
            //calling insert_at_beg method to insert element at begining
            insert_at_beg(e);
            cout << "\n<><> After Insertion <><>\n";
            printslinkedlist();

            break;
        case 2:
            //calling insert_at_end method to insert element at end
            insert_at_end(e);
            cout << "\n<><> After Insertion <><>\n";
```

```
            printslinkedlist();


            break;
        case 3:
            //calling insert_at_pos method to insert element at any position
            insert_at_pos(e);
            cout << "\n<><> After Insertion <><>\n";
            printslinkedlist();


            break;
        default:
            cout << "\n!!!Invalid Input!!! Please Try Again\n";
        }
        cout << "\nDo you want to see the Insertion methods menu again.\nPress(Y/y) or press any other key: ";
        cin >> selc;


    } while (selc == 'Y' || selc == 'y');


    cout << "\n||| Exited From Insert Menu |||\n";
}


//method to insert element at the begining
template <class T>
void linkedlist<T>::insert_at_beg(T e)
{
    //insertion at begining
    cout << "\n>>>Inserting element at the Begining<<<" << endl;
    if (head == NULL)
    {
        // cout << "Linked list is empty" << endl;
```

```
        head = new node<T>(e);
        tail = head;
        tail->next = head;
        head->prev = tail;
    }
    else
    {
        // cout << "Linked list has " << count << " elements" << endl;
        node<T> *tmp = new node<T>(e);
        tmp->next = head;
        head->prev = tmp;
        head = tmp;
        head->prev = tail;
        tail->next = head;
    }
    count++;
}

//method to insert element at the end
template <class T>
void linkedlist<T>::insert_at_end(T e)
{
    //insertion at the end
    cout << "\n>>>Inserting element at the End<<<" << endl;
    if (head == NULL)
    {
        head = new node<T>(e);
        tail = head;
        head->prev = tail;
        tail->next = head;
    }
```

```
    else
    {
        node<T> *tmp = new node<T>(e);
        tail->next = tmp;
        tmp->prev = tail;
        tail = tmp;
        tail->next = head;
        head->prev = tail;
    }
    count++;
}

//method to insert element at any position
template <class T>
void linkedlist<T>::insert_at_pos(T e)
{
    if (count == 0)
    {
        cout << "\n***The linked list is empty, there is only one place to add***\n";
        insert_at_beg(e);
        return;
    }
    cout << "\n>>>Inserting element at any Position<<<" << endl;

    int pos = 0;
    cout << "\nEnter the position at which you want to insert the element: ";
    cin >> pos;

    //checking for invalid position
    if (pos > (count + 1) || pos < 1)
    {
```

```
        cout << "\n!!! Invalid Position !!!\n";
        return;
    }

    if (pos == 1)
    {
        insert_at_beg(e);
    }
    else if (pos == count + 1)
    {
        insert_at_end(e);
    }
    else
    {
        int i = 1;
        node<T> *t = head;
        while (i != (pos - 1))
        {
            t = t->next;
            i++;
        }
        node<T> *nw = new node<T>(e);
        nw->next = t->next;
        nw->prev = t;
        t->next->prev = nw;
        t->next = nw;
        count++;
    }
}

//method to show menu for deletion
```

```
template <class T>
void linkedlist<T>::deletion()
{
   int ch;
   char selc;
   do
   {
      //option for the menu
      cout << "\n|><| Deletion Menu |><|\n";
      cout << "Choose any one of the option: \n";
      cout << "1. Delete from Begining\n";
      cout << "2. Delete from End\n";
      cout << "3. Delete from position\n";
      cout << "4. Delete by Element\n";
      cin >> ch;

      switch (ch)
      {
      case 1:
         //calling delete_from_beg method to delete element from the begining
         delete_from_beg();
         cout << "\n<><> After Deletion <><>\n";
         printslinkedlist();

         break;
      case 2:
         //calling delete_from_end method to delete element from the end
         delete_from_end();
         cout << "\n<><> After Deletion <><>\n";
         printslinkedlist();
```

```
            break;
        case 3:
            //calling delete_from_pos method to delete element by it's position
            delete_from_pos();
            cout << "\n<><> After Deletion <><>\n";
            printslinkedlist();

            break;
        case 4:
            //calling delete_by_elem method to delete element by searching the element
            delete_by_elem();
            cout << "\n<><> After Deletion <><>\n";
            printslinkedlist();

            break;
        default:
            cout << "\n!!!Invalid Input!!! Please Try Again\n";
        }
        cout << "\nDo you want to see the Deletion methods menu again.\nPress(Y/y) or press any other key: ";
        cin >> selc;

    } while (selc == 'Y' || selc == 'y');

    cout << "\n||| Exited From Deletion Menu |||\n";
}

//method to delete element from the Begining
template <class T>
void linkedlist<T>::delete_from_beg()
{
```

```
    cout << "\n>>>Deleting element from Begining<<<\n";
    if (head == NULL)
    {
        cout << "\n!!!There is nothing to Delete, Linked List is Empty\n";
        return;
    }
    else if (head->next == head)
    {
        delete (head);
        head = NULL;
        tail = NULL;
        count--;
    }
    else
    {
        node<T> *sec = head->next;
        sec->prev = tail;
        delete (head);
        head = sec;
        tail->next = head;
    }
    count--;
    cout << "\n***| Element Deleted Successfully |***\n";
}

//method to delete element from the end
template <class T>
void linkedlist<T>::delete_from_end()
{
    cout << "\n>>>Deleting element from End<<<\n";
    if (head == NULL)
```

```
    {
        cout << "\n!!!There is nothing to Delete, Linked List is Empty\n";
        return;
    }
    else if (head->next == head)
    {
        delete (head);
        head = NULL;
        tail = NULL;
        count--;
    }
    else
    {
        node<T> *t = tail->prev;
        t->next = head;
        delete (tail);
        tail = t;
        head->prev = tail;
    }
    count--;
    cout << "\n***| Element Deleted Successfully |***\n";
}

//method to delete element from any position
template <class T>
void linkedlist<T>::delete_from_pos()
{
    if (head == NULL)
    {
        cout << "\n!!! There is nothing to Delete, Linked List is Empty !!!\n";
        return;
```

```
}

if (head->next == head)
{
   cout << "!!! There is only one element in the list !!!\n";
   delete (head);
   head = NULL;
   tail = NULL;
   count--;

   cout << "\n***| Element Deleted Successfully |***\n";
   return;
}
cout << "\n>>>Deleting element from Position<<<" << endl;

int pos = 0;
cout << "\nEnter the position at which you want to Delete the element: ";
cin >> pos;

//checking for invalid position
if (pos > count || pos < 1)
{
   cout << "\n!!! Invalid Position !!!\n";
   return;
}

if (pos == 1)
{
   delete_from_beg();
}
else if (pos == count)
```

```
    {
        delete_from_end();
    }
    else
    {
        int i = 1;
        node<T> *t = head;
        while (i != (pos - 1))
        {
            t = t->next;
            i++;
        }
        node<T> *copy = t->next;
        t->next = t->next->next;
        t->next->prev = t;
        delete (copy);
        count--;

        cout << "\n***| Element Deleted Successfully |***\n";
    }
}

//method to delete by searching the element
template <class T>
void linkedlist<T>::delete_by_elem()
{
    if (head == NULL)
    {
        cout << "\n!!! There is nothing to Delete, Linked List is Empty !!!\n";
        return;
    }
```

```
cout << "\n>>>Deleting by searching the element in the linked list<<<\n";


T e;
cout << "\nEnter the element you want to delete: ";
cin >> e;


if (e == head->data)
{
   delete_from_beg();
}
else if (e == tail->data)
{
   delete_from_end();
}
else if (head->next == head)
{
   cout << "\n!!!The element you want to delete is not found in the linked list\n";
}
else
{
   node<T> *t = head;
   while (e != t->next->data)
   {
      t = t->next;
      if (t->next == head)
      {
         cout << "\n!!!The element you want to delete is not found in the linked list\n";
         return;
      }
   }
   node<T> *copy = t->next;
```

```
        t->next = t->next->next;
        t->next->prev = t;
        delete (copy);
        count--;

        cout << "\n***| Element Deleted Successfully |***\n";
    }
}

//method to search for an element and return it's node
template <class T>
node<T> *linkedlist<T>::returnNode()
{
    if (head == NULL)
    {
        cout << "\n!!! There is nothing to search, linked list is empty !!!\n";
        return NULL;
    }

    T e;
    cout << "\n|***| Enter the element you want to search its node: \n";
    cin >> e;

    node<T> *t = head;
    while (e != t->data)
    {
        t = t->next;
        if (t == NULL)
        {
            return t;
        }
```

```
    }

    return t;
}

//method to reverse the whole linked list
template <class T>
void linkedlist<T>::reverselinkedlist()
{
    if (head == NULL || count == 0)
    {
        cout << "\n!!! Linked List is Empty !!!\n";
        return;
    }
    cout << "\n|***| Reversing the linked list |***|\n";

    node<T> *temp = NULL;
    node<T> *current = head;

    // swap next and prev for all nodes of doubly linked list
    do
    {
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;
        current = current->prev;

    } while (current != head);

    node<T> *t = head;
    head = tail;
```

```
    tail = t;

    //connecting head and tail to each other to make the list circular
    head->prev = tail;
    tail->next = head;
}

//method to provide menu of merge methods to the user
template <class T>
void linkedlist<T>::mergeMenu(linkedlist<T> &l1)
{
    int ch;
    char selc;
    do
    {
        //option for the menu
        cout << "\n|><| Merge Menu |><|\n";
        cout << "Choose any one of the option: \n";
        cout << "1. Merge by Function\n";
        cout << "2. Merge by '+' Operator\n";
        cin >> ch;

        switch (ch)
        {
        case 1:
        {
            //calling mergefunc to merge two linked lists by using merge function
            //second linked list
            cout << "\n||| Please Enter the Second linked list |||\n";
            linkedlist<T> l2;
            l2.createlinkedlist();
```

```
        //merged linked list
        l1.mergefunc(l2);
        cout << "\nMerged Linked List: \n";
        l1.printslinkedlist();
    }
    break;
    case 2:
    {
        //calling overloaded + operator to merge two linked lists by using operator overloading

        //second linked list
        cout << "\n||| Please Enter the Second linked list |||\n";
        linkedlist<T> l2;
        l2.createlinkedlist();

        //merged linked list
        linkedlist<T> l3 = l1 + l2;
        cout << "\nMerged Linked List: \n";
        l3.printslinkedlist();
    }
    break;
    default:
        cout << "\n!!!Invalid Input!!! Please Try Again\n";
    }
    cout << "\nDo you want to see the Merge methods menu again.\nPress(Y/y) or press any other key: ";
    cin >> selc;

  } while (selc == 'Y' || selc == 'y');
```

```
    cout << "\n||| Exited From Merge Menu |||\n";
}

//method to merge to linked lists
template <class T>
void linkedlist<T>::mergefunc(linkedlist<T> l2)
{
    cout << "\n>>>Merging by Merge Function<<<\n";
    this->tail->next = l2.head;
    l2.head->prev = this->tail;
    this->tail = l2.tail;
    this->tail->next = this->head;
    this->head->prev = this->tail;
    this->count = this->count + l2.count;
}

//overloading + operator
template <class T>
linkedlist<T> linkedlist<T>::operator+(linkedlist<T> l2)
{
    cout << "\n>>>Merging by Operator Overloading<<<\n";
    this->tail->next = l2.head;
    l2.head->prev = this->tail;
    this->tail = l2.tail;
    this->tail->next = this->head;
    this->head->prev = this->tail;
    this->count = this->count + l2.count;

    return *this;
}
```

```
//method for the union of two linked lists
template <class T>
void linkedlist<T>::unionOfLl(linkedlist<T> l2)
{
   cout << "\n>>>Union Of Two Elements<<<\n";
   linkedlist<T> l1 = *this;
   linkedlist<T> l3;

   int arr[min(l1.count, l2.count)];
   int x = 0;

   node<T> *t1 = l1.head;
   node<T> *t2 = l2.head;

   do
   {
      t2 = l2.head;
      do
      {
         if (t1->data == t2->data)
         {
            arr[x] = t1->data;
            x++;
         }
         t2 = t2->next;

      } while (t2 != l2.head);

      t1 = t1->next;

   } while (t1 != l1.head);
```

```
cout << "\nCommon elements in both the linkedlists are: \n";
if (x == 0)
{
   cout << "\n!!! Nothing is common b/w two linked list !!!\n";
}
else
{
   for (int i = 0; i < x; i++)
   {
      cout << arr[i] << "\t";
   }
   cout << endl;
}

l3.count = l1.count;
node<T> *tmp0 = l1.head;
node<T> *copy = NULL;
//copying 1st linked list elements into 3rd one by one
for (int i = 0; i < l3.count; i++)
{
   if (i == 0)
   {
      l3.head = new node<T>(tmp0->data);
      l3.head->prev = l3.tail;
      copy = l3.head;
      l3.tail = l3.head;
      tmp0 = tmp0->next;
   }
   else
   {
```

```
        node<T> *tmp = new node<T>(tmp0->data);
        tmp->prev = copy;
        copy->next = tmp;
        copy = tmp;
        l3.tail = tmp;
        tmp0 = tmp0->next;
    }
}

//copying elments of 2nd linked list which are not repeated into 3rd
node<T> *tmp1 = l3.tail;
node<T> *tmp2 = l2.head;
do
{
    int c = 0;
    for (int i = 0; i < x; i++)
    {
        if (tmp2->data == arr[i])
        {
            c++;
        }
    }

    if (c == 0)
    {
        node<T> *t = new node<T>(tmp2->data);
        t->prev = tmp1;
        tmp1->next = t;
        tmp1 = t;
    }
    tmp2 = tmp2->next;
```

```
    } while (tmp2 != l2.head);

    l3.tail = tmp1;
    l3.tail->next = l3.head;
    l3.head->prev = l3.tail;
    l3.count = l1.count + l2.count - x;

    cout << "\n***| Union of Two linked lists |***\n";
    l3.printslinkedlist();
}

//method for the intersection of two linked list
template <class T>
void linkedlist<T>::intersection(linkedlist<T> l2)
{
    cout << "\n>>>Intersection Of Two Elements<<<\n";
    linkedlist<T> l1 = *this;

    int arr[min(l1.count, l2.count)];
    int x = 0;

    node<T> *t1 = l1.head;
    node<T> *t2 = l2.head;

    do
    {
        t2 = l2.head;
        do
        {
            if (t1->data == t2->data)
```

```
        {
            arr[x] = t1->data;
            x++;
        }
        t2 = t2->next;

    } while (t2 != l2.head);

    t1 = t1->next;
} while (t1 != l1.head);

cout << "\nCommon elements in both the linkedlists are: \n";
if (x == 0)
{
    cout << "\n!!! Nothing is common b/w two linked list !!!\n";
    return;
}
else
{
    for (int i = 0; i < x; i++)
    {
        cout << arr[i] << "\t";
    }
    cout << endl;
}

linkedlist<T> l3;
l3.count = x;

node<T> *copy = NULL;
for (int i = 0; i < l3.count; i++)
```

```
    {
        if (i == 0)
        {
            l3.head = new node<T>(arr[i]);
            l3.head->prev = l3.tail;
            copy = l3.head;
            l3.tail = l3.head;
        }
        else
        {
            node<T> *tmp = new node<T>(arr[i]);
            tmp->prev = copy;
            copy->next = tmp;
            copy = tmp;
            l3.tail = tmp;
        }
    }
    l3.head->prev = l3.tail;
    l3.tail->next = l3.head;

    cout << "\n***| Interaction of Linked Lists |***\n";
    l3.printslinkedlist();
}

//method to find the middle of the linked list
template <class T>
void linkedlist<T>::middleElem()
{
    cout << "\n>>>Finding the Middle element of the linked list<<<\n";

    node<T> *t1 = head;
```

```
    node<T> *t2 = head;

    while (t2->next != head && t2->next->next != head)
    {
        t2 = t2->next->next;
        t1 = t1->next;
    }

    cout << "\n***| Middle element of the Linked List is : " << t1->data << endl;
}

//main menu for all the functions of linked list
template <class T>
void linkedlist<T>::Menu()
{
    int ch;
    char selc;

    linkedlist<T> l1;
    l1.createlinkedlist();

    do
    {
        //option for the menu
        cout << "\n|><| Main Menu |><|\n";
        cout << "Choose any one of the option: \n";
        cout << "1. Insertion\n";
        cout << "2. Deletion\n";
        cout << "3. Searching\n";
        cout << "4. Display\n";
        cout << "5. Reverse\n";
```

```
cout << "6. Merge\n";
cout << "7. Union\n";
cout << "8. Intersection\n";
cout << "9. Find Middle Element\n";
cin >> ch;

switch (ch)
{
case 1:
{
   //calling insertion method to show insertion methods menu
   l1.insertion();
}
break;
case 2:
{
   //calling deletion method to show deletion menu
   l1.deletion();
}
break;
case 3:
{
   //calling returnNode method to return node if the element is found

   node<T> *s = l1.returnNode();
   if (s == NULL)
   {
      cout << "\n!!! Element not found !!!\n";
   }
   else
   {
```

```
        cout << "\n|**| Data of Node: " << s->data << endl;
        cout << "\n|**| Address of Node: " << s << endl;
    }
}
break;
case 4:
{
    //calling display method to display the linked list
    l1.printslinkedlist();
}
break;
case 5:
{
    //calling reverselinkedlist method to reverse the linked list
    l1.reverselinkedlist();
    cout << "\n<><> After Reversing <><>\n";
    l1.printslinkedlist();
}
break;
case 6:
{
    //calling mergefunc method to show merge methods
    l1.mergeMenu(l1);
}
break;
case 7:
{
    //second linked list
    cout << "\n||| Please Enter the Second linked list |||\n";
    linkedlist<T> l2;
    l2.createlinkedlist();
```

```
        //calling unionOfLl for the union of two linked lists
        l1.unionOfLl(l2);
    }
    break;
    case 8:
    {
        //second linked list
        cout << "\n||| Please Enter the Second linked list |||\n";
        linkedlist<T> l2;
        l2.createlinkedlist();

        //calling intersection for the intersection of two linked lists
        l1.intersection(l2);
    }
    break;
    case 9:
    {
        //calling middleElem methodto find the middle element of the linked list
        l1.middleElem();
    }
    break;
    default:
        cout << "\n!!!Invalid Input!!! Please Try Again\n";
    }
    cout << "\nDo you want to see the menu again.\nPress(Y/y) or press any other key: ";
    cin >> selc;

} while (selc == 'Y' || selc == 'y');

cout << "\n||| Exited From Main Menu |||\n";
```

```
}

//prototype function for taking the type of linkedlist
void llType();

//driver code for the program
int main()
{
   cout << "\n|***|Program Started|***|" << endl;

   //calling type function to take type of linked list user want to give
   llType();

   cout << "\n|***|Program Ended|***|" << endl;

   return 0;
}

//unction for taking the type of linkedlist
void llType()
{
   int choice = 0;

   //options for the user to select the type of array he/she wants
   cout << "\nEnter the type of linked list you want to give\n";
   cout << "1. Integer Array\n";
   cout << "2. Char Array\n";
   cout << "3. Double Array\n";
   cin >> choice;

   switch (choice)
```

```
    {
    case 1:
    {
        linkedlist<int> obj1;
        obj1.Menu();
    }
    break;

    case 2:
    {
        linkedlist<char> obj2;
        obj2.Menu();
    }
    break;
    case 3:
    {
        linkedlist<double> obj3;
        obj3.Menu();
    }
    break;
    default:
        cout << "\nPlease Enter a valid type.";
    }
}

// Output


// Time Complexities

// • Creation of Node - O(1)
```

// • insert_at_beg - O(1)

// • insert_at_end – O(1)

// • insert_at_pos – O(n)

// • delete_from_beg – O(1)

// • delete_from_end – O(1)

// • delete_from_pos – O(n)

// • delete_by_elem – O(n)

// • search – O(n)

// • display – O(n)

// • reverse – O(n)

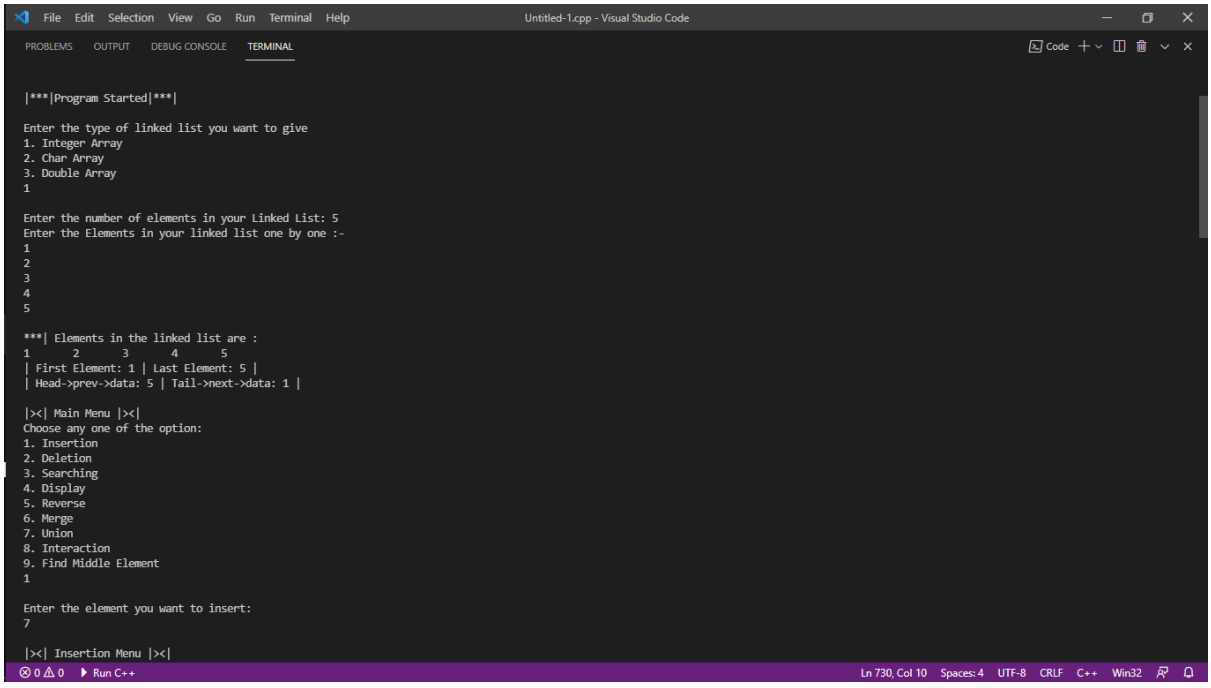
// • merge – O(1)

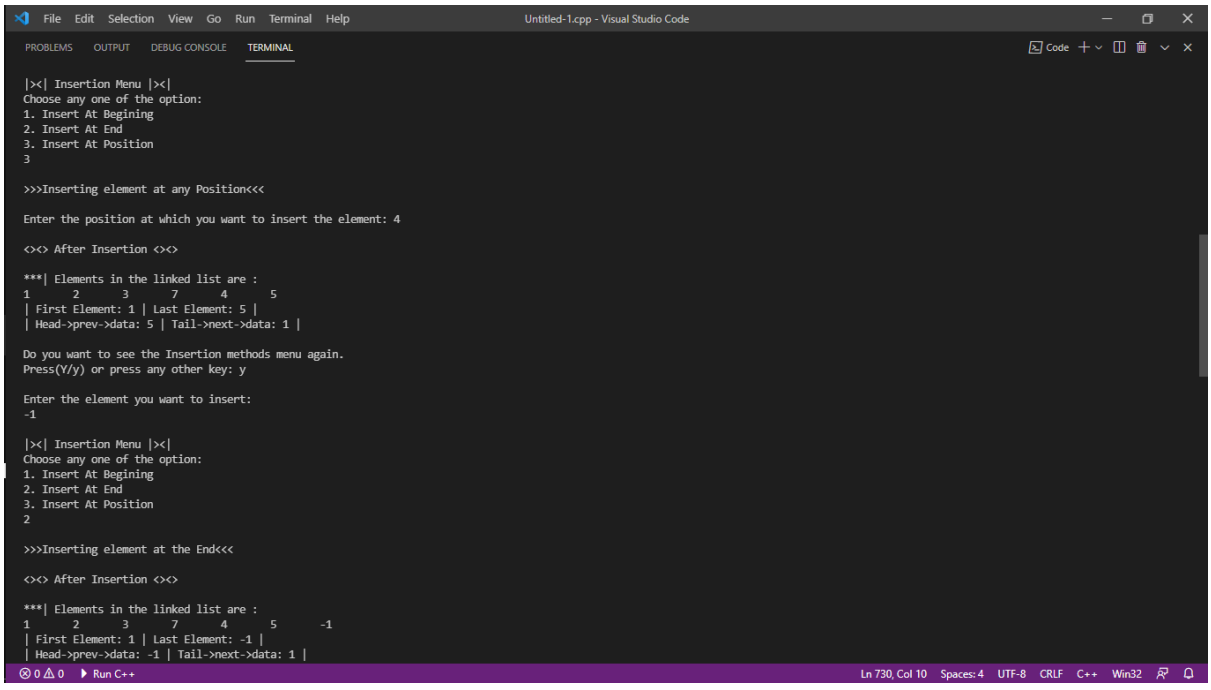
// • union – O(ab)

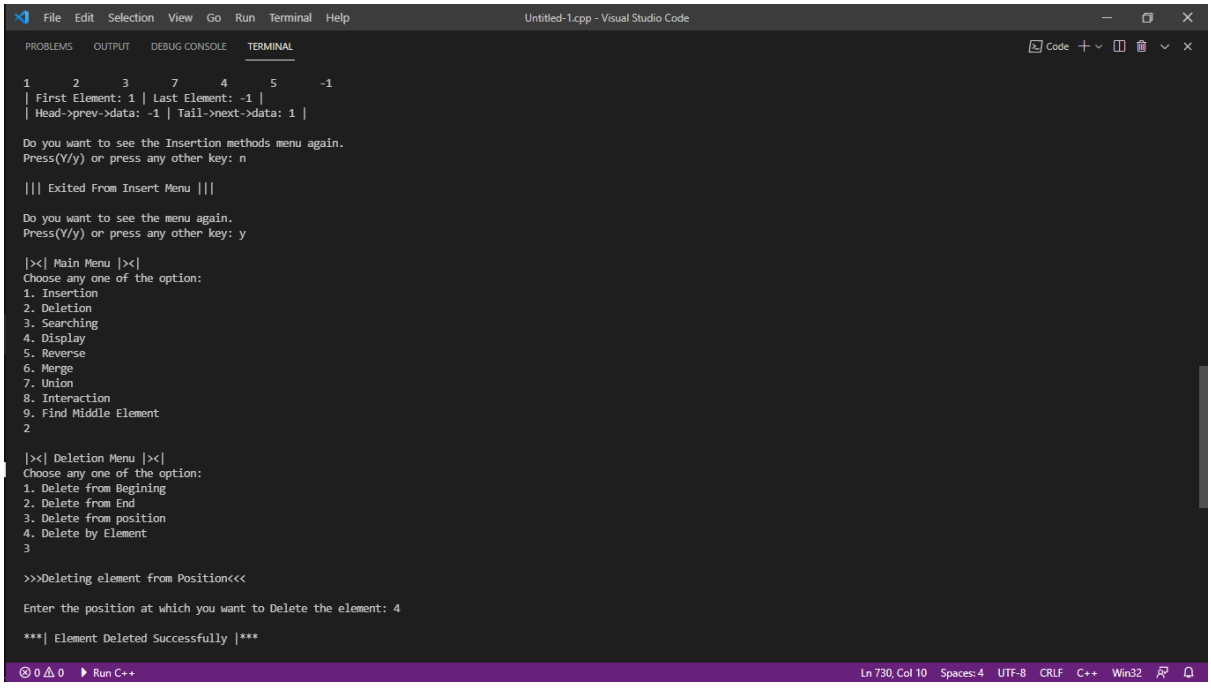
// • intersection – O(ab)

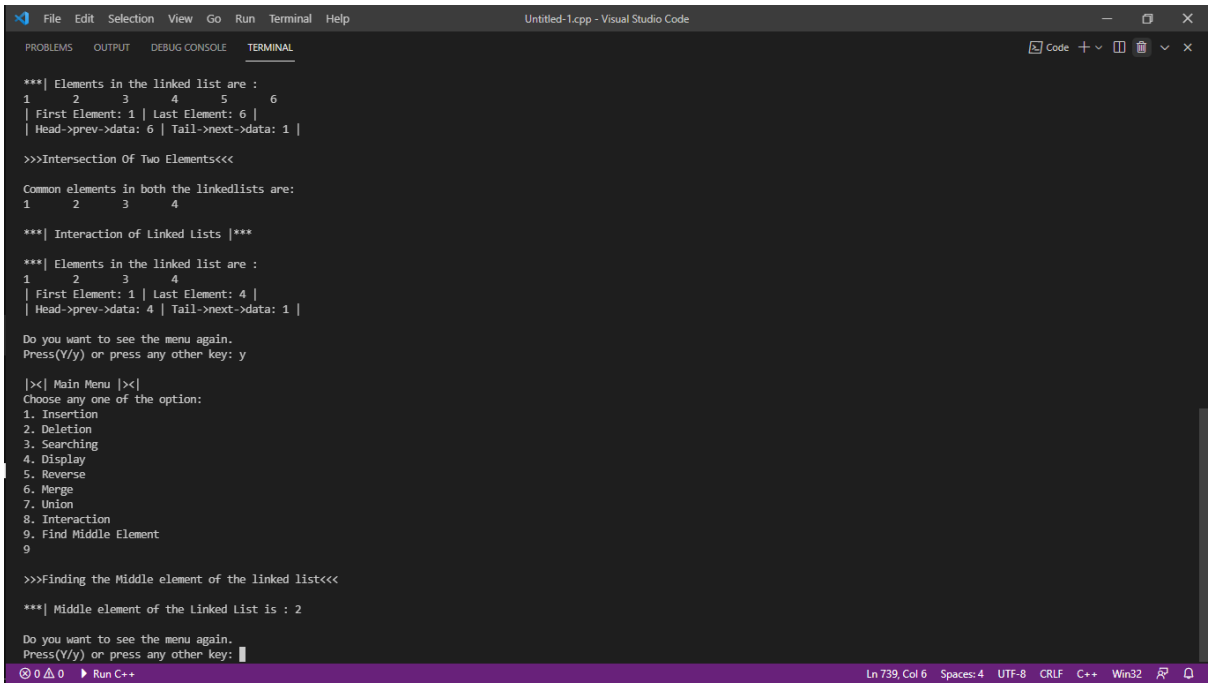
// • middle element – O(n/2)

OUTPUT


File Edit Selection View Go Run Terminal Help
Untitled-1.cpp - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Code
|***|Program Started|***|
Enter the type of linked list you want to give
1. Integer Array
2. Char Array
3. Double Array
1
Enter the number of elements in your Linked List: 5
Enter the Elements in your linked list one by one :-
1
2
3
4
5
***| Elements in the linked list are :
1 2 3 4 5
| First Element: 1 | Last Element: 5 |
| Head->prev->data: 5 | Tail->next->data: 1 |
|><| Main Menu |><|
Choose any one of the option:
1. Insertion
2. Deletion
3. Searching
4. Display
5. Reverse
6. Merge
7. Union
8. Interaction
9. Find Middle Element
1
Enter the element you want to insert:
7
|><| Insertion Menu |><|
0 0 Run C++
Ln 730, Col 10 Spaces: 4 UTF-8 CRLF C++ Win32


File Edit Selection View Go Run Terminal Help
Untitled-1.cpp - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Code
|><| Insertion Menu |><|
Choose any one of the option:
1. Insert At Begining
2. Insert At End
3. Insert At Position
3
>>>Inserting element at any Position<<<
Enter the position at which you want to insert the element: 4
<><> After Insertion <><>
***| Elements in the linked list are :
1 2 3 7 4 5
| First Element: 1 | Last Element: 5 |
| Head->prev->data: 5 | Tail->next->data: 1 |
Do you want to see the Insertion methods menu again.
Press(Y/y) or press any other key: y
Enter the element you want to insert:
-1
|><| Insertion Menu |><|
Choose any one of the option:
1. Insert At Begining
2. Insert At End
3. Insert At Position
2
>>>Inserting element at the End<<<
<><> After Insertion <><>
***| Elements in the linked list are :
1 2 3 7 4 5 -1
| First Element: 1 | Last Element: -1 |
| Head->prev->data: -1 | Tail->next->data: 1 |
0 0 Run C++
Ln 730, Col 10 Spaces: 4 UTF-8 CRLF C++ Win32


File Edit Selection View Go Run Terminal Help
Untitled-1.cpp - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Code
1 2 3 7 4 5 -1
| First Element: 1 | Last Element: -1 |
| Head->prev->data: -1 | Tail->next->data: 1 |
Do you want to see the Insertion methods menu again.
Press(Y/y) or press any other key: n
||| Exited From Insert Menu |||
Do you want to see the menu again.
Press(Y/y) or press any other key: y
|><| Main Menu |><|
Choose any one of the option:
1. Insertion
2. Deletion
3. Searching
4. Display
5. Reverse
6. Merge
7. Union
8. Interaction
9. Find Middle Element
2
|><| Deletion Menu |><|
Choose any one of the option:
1. Delete from Begining
2. Delete from End
3. Delete from position
4. Delete by Element
3
>>>Deleting element from Position<<<
Enter the position at which you want to Delete the element: 4
| Element Deleted Successfully |
0 0 Run C++
Ln 730, Col 10 Spaces: 4 UTF-8 CRLF C++ Win32


File Edit Selection View Go Run Terminal Help
Untitled-1.cpp - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Code
***| Elements in the linked list are :
1 2 3 4 5 6
| First Element: 1 | Last Element: 6 |
| Head->prev->data: 6 | Tail->next->data: 1 |
>>>Intersection Of Two Elements<<<
Common elements in both the linkedlists are:
1 2 3 4
| Interaction of Linked Lists |
***| Elements in the linked list are :
1 2 3 4
| First Element: 1 | Last Element: 4 |
| Head->prev->data: 4 | Tail->next->data: 1 |
Do you want to see the menu again.
Press(Y/y) or press any other key: y
|><| Main Menu |><|
Choose any one of the option:
1. Insertion
2. Deletion
3. Searching
4. Display
5. Reverse
6. Merge
7. Union
8. Interaction
9. Find Middle Element
9
>>>Finding the Middle element of the linked list<<<
***| Middle element of the Linked List is : 2
Do you want to see the menu again.
Press(Y/y) or press any other key:
0 0 Run C++
Ln 739, Col 6 Spaces: 4 UTF-8 CRLF C++ Win32