**Name: Riya Jain**

**Section: A (CSE)**

**Roll No: 17**

**EXPERIMENT: 5**

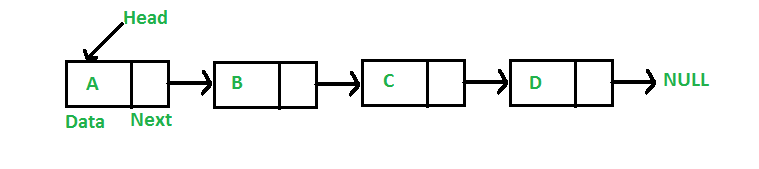
**Aim: To study singly linked linear list and implement various operations on it – Insert, Delete, Reverse, Sorting, Locate, Linked Stack, and Linked Queue.**

**Problem Statement: Create a self-referential structure, Node to represent a node of a singly linked linear list.  
Implement the routines to (1) find length of the list, (2) create a list, (3) insert an element – at the beginning, at the end and at a specified position in the list, insertion in ordered way (4) delete an element from the beginning, end or a specified position at the list, (5) reverse the list, (6) search the list, (7) Sort the list. Create a menu-driven program to test these routines.**

**Use the singly linked linear list routines to implement a linked stack and a linked queue.**

**Theory:**

**A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:**



**In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the 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. Following are the important terms to understand the concept of Linked List.**

* **Link − Each link of a linked list can store a data called an element.**
* **Next − Each link of a linked list contains a link to the next link called Next.**
* **LinkedList − A Linked List contains the connection link to the first link called First.**

## **Linked List Representation**

**Linked list can be visualized as a chain of nodes, where every node points to the next node.**

****

**As per the above illustration, following are the important points to be considered.**

* **Linked List contains a link element called first.**
* **Each link carries a data field(s) and a link field called next.**
* **Each link is linked with its next link using its next link.**
* **Last link carries a link as null to mark the end of the list.**

## **Types of Linked List**

**Following are the various types of linked list.**

* **Simple Linked List − Item navigation is forward only.**
* **Doubly Linked List − Items can be navigated forward and backward.**
* **Circular Linked List − Last item contains link of the first element as next and the first element has a link to the last element as previous**

**Basic Operations on Linked Lists: Following are the basic operations supported by a list.**

**● Insertion − Adds an element at the beginning of the list.**

**● Deletion − Deletes an element at the beginning of the list.**

**● Display − Displays the complete list.**

**● Search − Searches an element using the given key.**

**● Delete − Deletes an element using the given key.**

**Algorithms:**

1. **Insertion at the beginning:**
2. Create a Node

NEWNODE <- Node

1. Is the Node created?

If NEWNODE = NULL

Return START

1. Assign Data value to Node

NewNode → data = DATA

1. Is the List Empty?

If START=NULL

NewNode → next = NULL

Else NewNode → next = START

5. START = NewNode

Return START

1. **Insertion at the end:**

1. Create a Node NEWNODE <- Node

2.Is the Node created? If NEWNODE = NULL Return START

1. Initialize Node

NewNode → DATA = DATA NewNode → Next = NULL

1. Is the List Empty? If START=NULL START = NewNode Else

TEMP = START

While ((TEMP → Next )!=NULL) TEMP = TEMP → Next

1. TEMP → Next = NEWNODE
2. Return START

**3. Insertion at a specific position:**

1. Initialize TEMP = START; and k = 0
2. Is the List Empty?

If START==NULL

InsertAtBeg(START,DATA)

Else Repeat while( k is less than POS-1)

1. TEMP = TEMP →Next
2. If (TEMP == NULL)
3. Display “Node in the list less than the position”
4. Exit
5. k = k + 1
6. Create a Node NEWNODE <- Node
7. Is the Node created? If NEWNODE = NULL Return START
8. Assign values to Node NEWNODE → DATA = DATA NEWNODE → Next = TEMP → Next
9. TEMP → Next = NEWNODE
10. Exit

**4. Ordered Insertion:** 1.Create a Node NEWNODE <- Node

### Is the Node created? If NEWNODE = NULL Return START

1. Assign values to Node NEWNODE → DATA = DATA
2. Is the List Empty? If START=NULL Return NEWNODE
3. If (START → DATA >= NEWNODE → DATA ) NEWNODE → Next=START

Return NEWNODE

1. TEMP = START
2. Check for Proper Position.

While( (TEMP <>NULL) &&(TEMP →Next →DATA

<= NEWNODE → DATA) ) TEMP = TEMP →Next

1. NEWNODE → Next = TEMP →Next
2. TEMP →Next = NEWNODE
3. Return START

**5. Deletion at the beginning:**

1. Is the List Empty?

If START=NULL

Print “ Empty List, Deletion not Possible” Return START

1. Is Only one node in List? If START → Next = NULL Key = START → DATA FREE( START)

Return NULL

1. TEMP = START
2. Key = TEMP → DATA
3. START = START → Next
4. FREE(TEMP)
5. Print “Key deleted is: Key”
6. Return START

**6. Deletion at the end:**

1. Is the List Empty?

If START=NULL

Print “ Empty List, Deletion not Possible” Return START

1. Is Only one node in List? If START → Next = NULL Key = START → DATA FREE( START)

Return NULL

1. TEMP = START
2. Traverse till end of list

While ((TEMP → Next → Next)<>NULL) TEMP = TEMP → Next

1. Delete node & Adjust pointers
2. HOLD= TEMP → Next
3. Key = HOLD → DATA
4. FREE(HOLD)
5. TEMP → Next = NULL
6. Return START
7. **Deletion at specific position:**
8. Initialize TEMP = START; and k = 0
9. Is the List Empty? If TEMP=NULL Return NULL
10. If(POS=0)

START = TEMP → Next FREE(TEMP)

Return START

1. Repeat the step while(k is less than POS-1)
2. TEMP = TEMP →Next
3. If (TEMP = NULL)
4. Display “Node in the list less than the position”
5. Exit
6. k = k + 1
7. HOLD = TEMP → Next → Next
8. Key = Temp → Next → DATA
9. FREE(TEMP → Next)
10. TEMP → Next = HOLD
11. Print “Deleted Key is “, Key
12. Return START
13. Repeat the step while(k is less than POS-1)
14. TEMP = TEMP →Next
15. If (TEMP = NULL)
16. Display “Node in the list less than the position”
17. Exit
18. k = k + 1
19. HOLD = TEMP → Next → Next
20. Key = Temp → Next → DATA
21. FREE(TEMP → Next)
22. TEMP → Next = HOLD
23. Print “Deleted Key is “, Key
24. Return START

**8. Traversal:**

1. If (START is equal to NULL)
2. Display “The list is Empty”
3. Exit
4. Initialize TEMP = START
5. Repeat the step 4 and 5 until (TEMP not equal to NULL)
6. Display TEMP → DATA
7. TEMP = TEMP → Next
8. Exit

**9. Search:**

1. Initialize TEMP = START & POS =0
2. while(TEMP = NULL)
3. If (TEMP → DATA is equal to DATA)
4. Display “The data is found at POS”
5. Exit
6. TEMP = TEMP → Next
7. POS = POS+1
8. If (TEMP is equal to NULL)

(a) Display “The data is not found in the list”

1. Exit

**10. Sorting:**

1.HOLD = START

1. Initialize CNT=0, i=0
2. while(HOLD <> NULL) (a)HOLD=HOLD → next (b)CNT=CNT+1
3. while(i less than CNT-1)
4. TEMP = START
5. while ((TEMP→ NEXT) <> NULL)
6. if ((TEMP →DATA) >(TEMP → NEXT →DATA))
7. k=TEMP →DATA
8. TEMP →DATA = TEMP → NEXT →DATA (iii)TEMP → NEXT →DATA =k
9. TEMP = TEMP →NEXT 7. i=i+1
10. EXIT
    1. **Copy:**

list\*Copy(list\* s)

{

list \*temp;

list \*s1=NULL; temp=s;

while (temp!=NULL)

{

s1 = InsertAtEnd(s1,temp->data); temp=temp->next;

}

return s1;

}

* 1. **Reverse:**

void reverse(list\* Start)

{

list\* prev = NULL; list\* current = Start; list\* next = NULL;

while (current != NULL) {

// Store next

next = current->next;

// Reverse current node's pointer current->next = prev;

// Move pointers one position ahead. prev = current;

current = next;

}

Start = prev; traverse(Start);

}

**CODE:**

#include<stdio.h>

#include<stdlib.h>

struct Node

{

    int data;

    struct Node\*next; //self referencial structure

};

typedef struct Node List;

List\*create\_Node(List\*head);

List\*create\_space();

List\*Insertatbegin(struct Node\*head,int data);

List\*Insertatend(struct Node\*head,int data);

List\*Insertatindex(struct Node\*head,int data,int index);

List\*Deleteatbegin(struct Node\*head);

List\*orderedlist(struct Node\*head,int data);

int length(struct Node\*ptr);

void traversallinkedlist(struct Node\*ptr);

void displaylinkedlist(struct Node\*head);

void search(struct Node\*head,int ele);

List\*DeleteatEnd(struct Node\*head);

List\*Deleteatindex(List\*head,int index);

void concatenation();

void Copy(List\*head);

List\*Reverse(List\*head);

void sorting(List\*head);

int main()

{

List\*head;// abhi list empty hai ek bhi node nhi hai toh head pointer is declared to store the address of first node when it will be created

head=NULL;  //ek bhi node nhi hai wo bhi empty isliye assigining it to null so that there is no garbage in memory assigned to head

 // struct Node\*head=NULL;

//head=create\_Node(head);

//traversallinkedlist(head);

 //if(head!=NULL)

 //printf("Address of Node created:%u",(unsigned)head);

 int ch,n,element,element1,index=0,indexx=0;

while(1)

{

printf("\nDear user please enter operation you want to perform on the array of your choice!!!");

printf("\nPRESS: 1)Insertion Operations\n 2)Deletion Operations\n 3)Concatenation\n 4)length of linked list\n 5) Searching\n 6) Copy\n 7)Reverse\n8)Sorting\n9)Traversing\n10)Stack Operations\n11)Queue Operations\n 12)exit:");

scanf("%d",&ch);

switch(ch)

{

case 1:printf("You opted for Insertion Operations!!");

       int ch1,l=1;

 while(l)

 {

 printf("\nPRESS: 1)Insertatbegin 2)Insertatend 3)Insertatindex 4)orderedlist 5)display:");

 scanf("%d",&ch1);

 switch(ch1)

 {

case 1: printf("You opted for Insertion at Beginning!\n");

        printf("Enter data do you want to insert:");

        scanf("%d",&element);

         head=Insertatbegin(head,element);

         break;

case 2: printf("You opted for Insertion at End!!\n");

        printf("Enter the data do you want to insert:");

        scanf("%d",&element);

        head=Insertatend(head,element);

        break;

case 3: printf("You opted for Insertion at Index!!\n");

        printf("Enter the data do you want to insert:");

        scanf("%d",&element);

        printf("Enter the index position do you want to insert:");

        scanf("%d",&index);

        head=Insertatindex(head,element,index);

        break;

case 4: printf("You opted for Insertion in Ordered List!!\n");

        printf("Enter the data do you want to insert:");

        scanf("%d",&element);

        head=orderedlist(head,element);

        break;

case 5: printf("You opted for Displaying Linkedlist!!");

        printf("\nYour Linked list is:\n");

        traversallinkedlist(head);

        break;

case 6: l=0;

        break;

default: printf("You entered invalid choice");

}

}

break;

case 2: printf("You opted for Deletion Operations!!");

         int ch2,l1=1;

 while(l1)

 {

 printf("\nPRESS: 1)Deletionatbegin 2) Deletionatend 3)Deletionatindex 4)display:");

 scanf("%d",&ch2);

 switch(ch2)

 {

case 1: printf("You opted for Deletion at beginning!\n");

        head=Deleteatbegin(head);

        break;

case 2: printf("You opted for Deletion at End!!\n");

        head=DeleteatEnd(head);

        break;

case 3: printf("You opted for Delete a value from a particular index in your linked list!!\n");

        printf("Enter the index position do you want to delete:");

        scanf("%d",&indexx);

        head=Deleteatindex(head,indexx);

        break;

case 4: printf("You opted for displaying Linked list!!");

        printf("\nYour Linked list is:\n");

        displaylinkedlist(head);

        break;

case 5: l1=0;

        break;

default: printf("You entered invalid choice");

}

}

break;

case 3: printf("You opted for Concatenation Operation!!\n");

        concatenation();

        break;

case 4: printf("You opted for Calculation of length of linked List Operation!!");

        int res=length(head);

        printf("\nYour length of linked list is:%d",res);

        break;

case 5: printf("You opted for searching a given node in Linked list!!");

        printf("Enter data do you want to search:");

        scanf("%d",&element1);

        search(head,element1);

break;

case 6:printf("You opted for Copying Operation in Linked list!!");

       Copy(head);

break;

case 7:printf("You opted for Reversing a Linked list!!");

       head=Reverse(head);

break;

case 8:printf("You opted for Sorting a Linked list!!");

       sorting(head);

       break;

case 9: printf("\nYou opted for Traversing a Linked list!!\n");

        traversallinkedlist(head);

        break;

case 10:printf("You opted for Stack Operations!!");

       int ch3,l2=1;

       List\*head1=NULL;

 while(l2)

 {

 printf("\nPRESS: 1)PUSH 2)POP 3)TOP VALUE 4)display:");

 scanf("%d",&ch3);

 switch(ch3)

 {

case 1: printf("You opted for PUSH OPERATION!\n");

        printf("Enter data do you want to insert:");

        scanf("%d",&element);

        head1=Insertatbegin(head1,element);

        break;

case 2: printf("You opted for POP OPERATION!!\n");

        head1=Deleteatbegin(head1);

        break;

case 3: printf("Your top value is:%d",head1->data);

        break;

case 4: printf("You opted for Displaying Linkedlist!!");

        printf("\nYour Linked list is:\n");

        traversallinkedlist(head1);

        break;

case 5: l2=0;

        break;

default: printf("You entered invalid choice");

}

}

break;

case 11:printf("You opted for Queue Operations!!");

        int ch4,l3=1;

        List\*head2=NULL;

 while(l3)

{

 printf("\nPRESS: 1)Insert in queue 2) Delete from queue 3)display:");

 scanf("%d",&ch4);

 switch(ch4)

{

case 1: printf("You opted for Insertion in Queue!\n");

        printf("Enter data do you want to insert:");

        scanf("%d",&element);

        head2=Insertatend(head2,element);

         break;

case 2: printf("You opted for Deletion in Queue!!\n");

        head2=Deleteatbegin(head2);

        break;

case 3: printf("You opted for Displaying Linkedlist!!");

        printf("\nYour Linked list is:\n");

        traversallinkedlist(head2);

        break;

case 4: l3=0;

        break;

default: printf("You entered invalid choice");

}

}

 break;

case 12: exit(0);

default : printf("You entered invalid choice!!");

}

}

return 0;

}

List\*create\_Node(List\*head)

{

   int n,d;

   printf("Enter the number of nodes to be created:");

   scanf("%d",&n);

   for(int i=0;i<n;i++)

   {       printf("Enter data:");

           scanf("%d",&d);

           head=Insertatbegin(head,d);

   }

 return head;

}

List\*Insertatbegin(List\*head,int data)

{

    List\*ptr=create\_space();

    ptr->data=data;

   /\* if(head==NULL)

    {

      ptr->next=NULL;

      head=ptr;

      return head;

    }\*/

    ptr->next=head; //null ho gya automatically

    head=ptr;

    return head;

}

List\*Insertatend(List\*head,int data)

{

   List\*ptr=create\_space();

   ptr->data=data;

   //ptr->next=NULL;

   if(head==NULL)

   {

   head=ptr;

   ptr->next=NULL;

   return head;

   }

   List\*p=head;

   while(p->next!=NULL)

   {

      p=p->next;

   }

   p->next=ptr;

   ptr->next=NULL;

   return head;

}

List\*Insertatindex(List\*head,int data,int index)

{

   if(head==NULL)  // if list is empty

   {

   head=Insertatbegin(head,data);

   return head;

   }

   int i=0;

   List\*p=head;

   List\*ptr=create\_space();

   while(i!=index-1)

   {

    p=p->next;

     if (p== NULL)

     {

     printf("Nodes in the list are less than the index provided"); //agar nodes kam hai aur index jyaada

     return head;

     }

    i++;

   }

  // printf("%d",p->data);

   ptr->data=data;

   ptr->next=p->next;

   p->next=ptr;

return head;

}

List\*orderedlist(List\*head,int data)

{

 //struct Node\*p=(struct Node\*) malloc(sizeof(struct Node));

 //struct Node\*ptr=(struct Node\*) malloc(sizeof(struct Node));

 List\*ptr=create\_space();

 ptr->data=data;

/\*if(head==NULL)  // if list is empty

   {

   ptr->next=NULL;

   return ptr;

   }\*/

  if(head==NULL)  // if list is empty

   {

   head=Insertatbegin(head,data);

   return head;

   }

 if((head->data)>=(ptr->data))

 {

  ptr->next=head;

  head=ptr;

  return head;

 }

 else{

struct Node\*p=head;

 while(p->next!=NULL)

 {

 if((p->next->data)>=ptr->data)

 {

     break;

 }

 else

 {

     p=p->next;

 }

 }

// printf("%d",p->next->data);

 /\*if((p->next->data)<ptr->data)

 {

     ptr->next=NULL;

     p->next=ptr;

 }\*/

 ptr->next=p->next;

 p->next=ptr;

return head;

 }

}

void traversallinkedlist(List\*ptr)

{

    if(ptr==NULL)

    printf("Node is Empty\n");

    while(ptr!=NULL)

   {

       printf("Element:%d Address:%lu->",ptr->data,(long unsigned)ptr->next);

       ptr=ptr->next;

   }

}

List\*Deleteatbegin(List\*head)

{ int temp;

 if(head==NULL)

 {

 printf("Deletion not possible as your linked list is empty");

 return head;

 }

 if(head->next==NULL)

 {

    temp=head->data;

    free(head);

    return NULL;

 }

 List\*ptr=head;

 temp=ptr->data;

 head=head->next;

 free(ptr);

 printf("Your deleted node was data with data value of %d",temp);

 return head;

}

int length(List\*ptr)

{

   if(ptr==NULL)

   {

    return 0;

   }

   int count=1;

   while(ptr->next!=NULL)

   {

           count++;

           ptr=ptr->next;

   }

   return count;

}

List\*create\_space()

{

   List\*head=(struct Node\*) malloc(sizeof(struct Node));

   if(head==NULL) //agar memory allocate hi nhi huyi

   {

       printf("Node not created\n");

       return NULL; //return head

   }

   else

   {

       printf("Node created\n");

       return head;

   }

}

List\*DeleteatEnd(struct Node\*head)

{

 int temp;

 if(head==NULL)

 {

 printf("Deletion not possible as your linked list is empty");

 return head;

 }

 if(head->next==NULL)

 {  temp=head->data;

    free(head);

    return NULL;

 }

 List\*p=head;

 List\*q=head->next;

 while(q->next!=NULL)

 {

         p=p->next;

         q=q->next;

 }

 p->next=NULL;

 temp=q->data;

 free(q);

 printf("Your deleted node was data with data value of %d",temp);

 return head;

}

void displaylinkedlist(struct Node\*head)

{

if(head==NULL)

 {

 printf("Deletion not possible as your linked list is empty");

 }

 struct Node\*temp=head;

while(temp!=NULL)

   {

       printf("Element:%d Address:%lu->",temp->data,(long unsigned)temp->next);

       temp=temp->next;

   }

}

void search(struct Node\*head,int ele)

{

  int i=0;

 List\*p=head;

 while(p!=NULL)

 {

  if(p->data==ele)

  {

  printf("The element is found at index %d",i);

  break;

  }

  else

  {

  p=p->next;

  i++;

  }

}

if(p==NULL)

printf("\nThe data is not found in the list");

}

List\*Deleteatindex(List\*head,int index)

{    int temp;

     List\*p=head;

     if(head==NULL)

     {

      printf("The list is empty");

      return head;

     }

     if(index==0)

     {

             head=p->next;

             free(p);

             return head;

     }

     int l=length(head);

     if(index>l)

     {

        printf("Deletion not possible\n");

        return head;

     }

     int i=0;

     List \*q=head->next;

     while(i!=index-1)

     {

       p=p->next;

       q=q->next;

       if(p==NULL)

       {

       printf("Nodes in the list are less than the index provided"); //agar nodes kam hai aur index jyaada

       return head;

       }

       i++;

     }

     p->next=q->next;

     temp=q->data;

     free(q);

     printf("Your deleted node was data with data value of %d",temp);

     return head;

 }

 void concatenation()

 {      List \*head1=NULL;

        List \*head2=NULL;

        head1=create\_Node(head1);

        head2=create\_Node(head2);

        List\*p=head1;

         while(p->next!=NULL)

         {

             p=p->next;

         }

         p->next=head2;

         traversallinkedlist(head1);

}

void Copy(List\*head)

{

List \*q=NULL;

List \*p=head;

while (p!=NULL)

{

q=Insertatend(q,p->data); //Insertion at begin se not possible coz previous data ki info kaise rkhte

p=p->next;

}

traversallinkedlist(q);

}

List\*Reverse(List\*head)

{

List\*prev=NULL;

List\*p=head;

List\*next=NULL;

while(p!= NULL)

{

// Store next

next=p->next;

// Reverse current node's pointer

p->next = prev;

// Move pointers one position ahead.

prev=p;

p=next;

}

head= prev;

return head;

//traversallinkedlist(head);

//sorting(head);

//traversallinkedlist(head);

}

void sorting(List\*head)

{

/\*if(head==NULL)

{

printf("List is empty\n");

}\*/

//traversallinkedlist(head);

int i=0;

int n=length(head);

//printf("\nlength:%d",n);

List\*p;

while(i<n-1)

{       p=head;

        while(p->next!=NULL)

        {

                if(p->data>p->next->data)

                {

                int temp=p->data;

                p->data=p->next->data;

                p->next->data=temp;

                }p=p->next;

        }

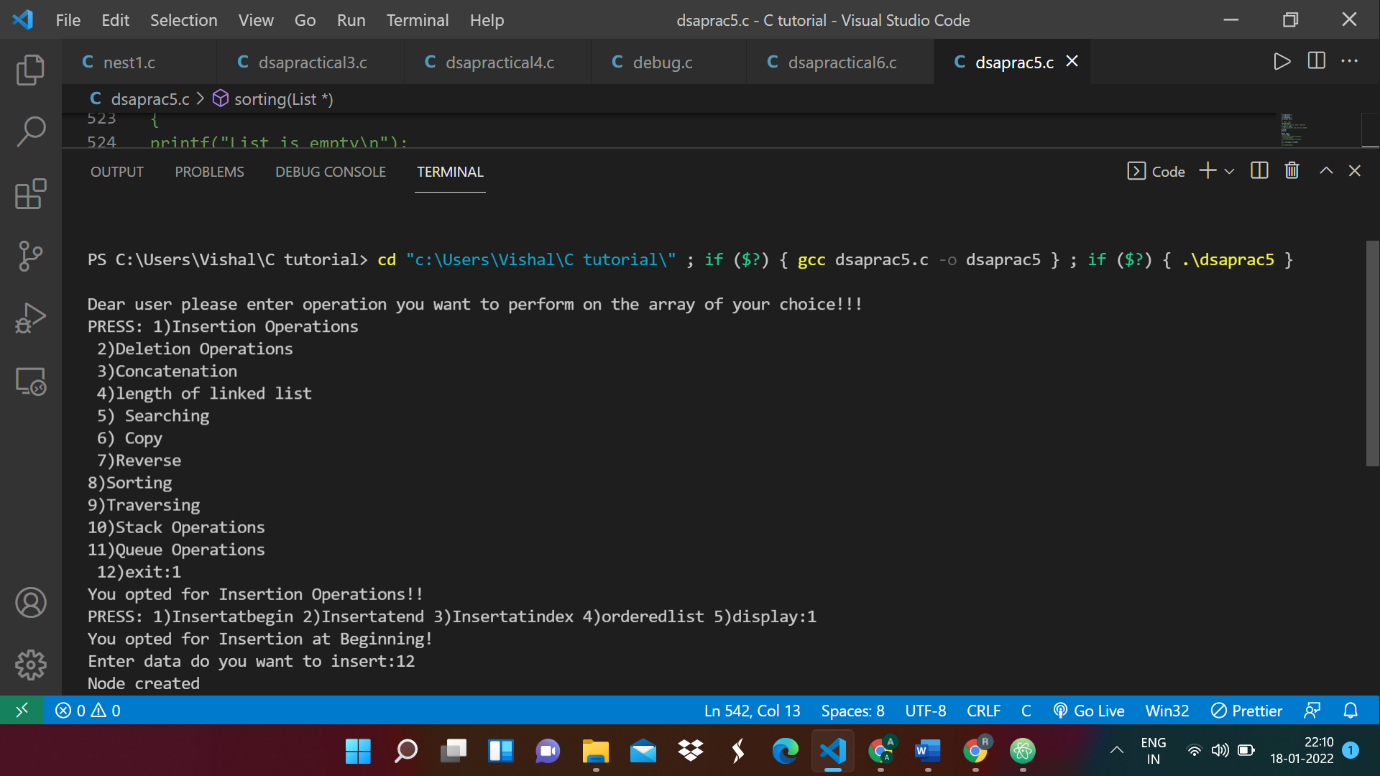
        i++;

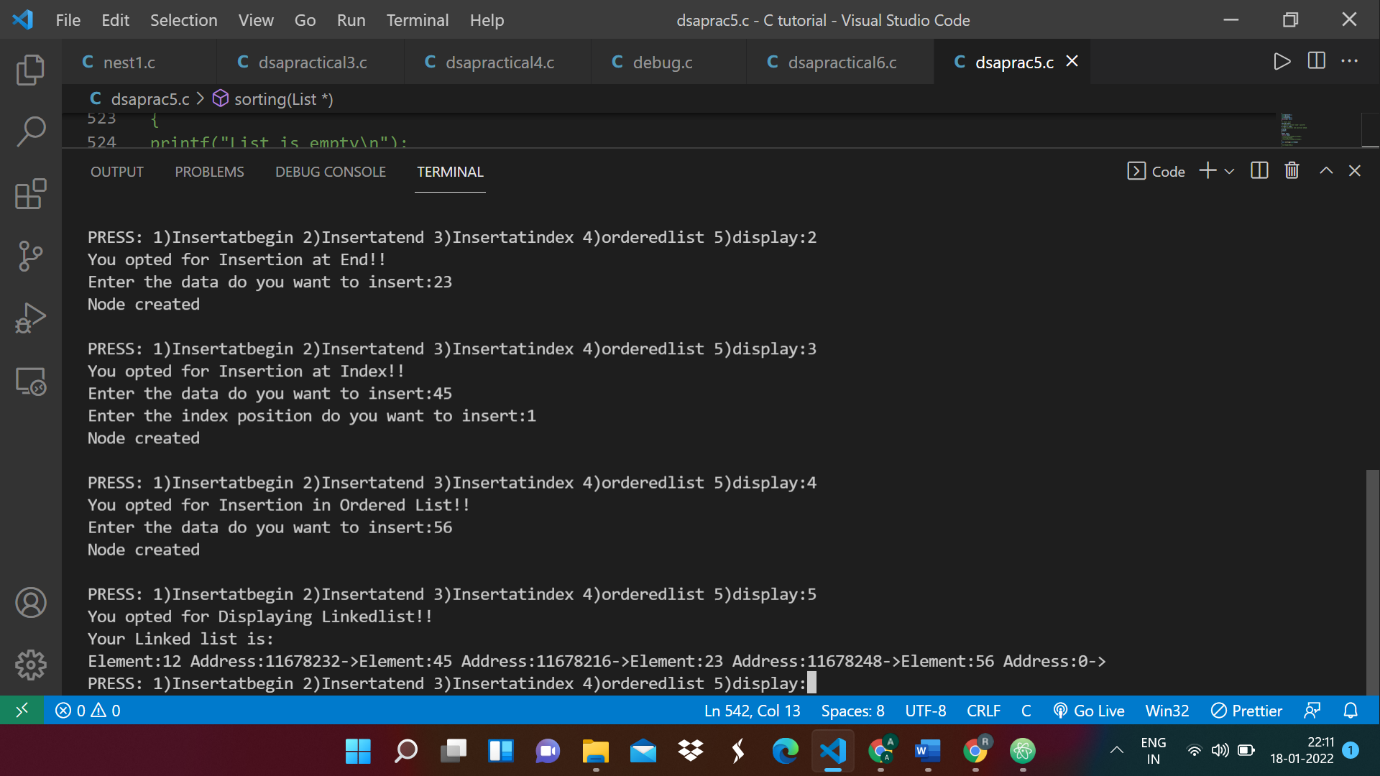
}

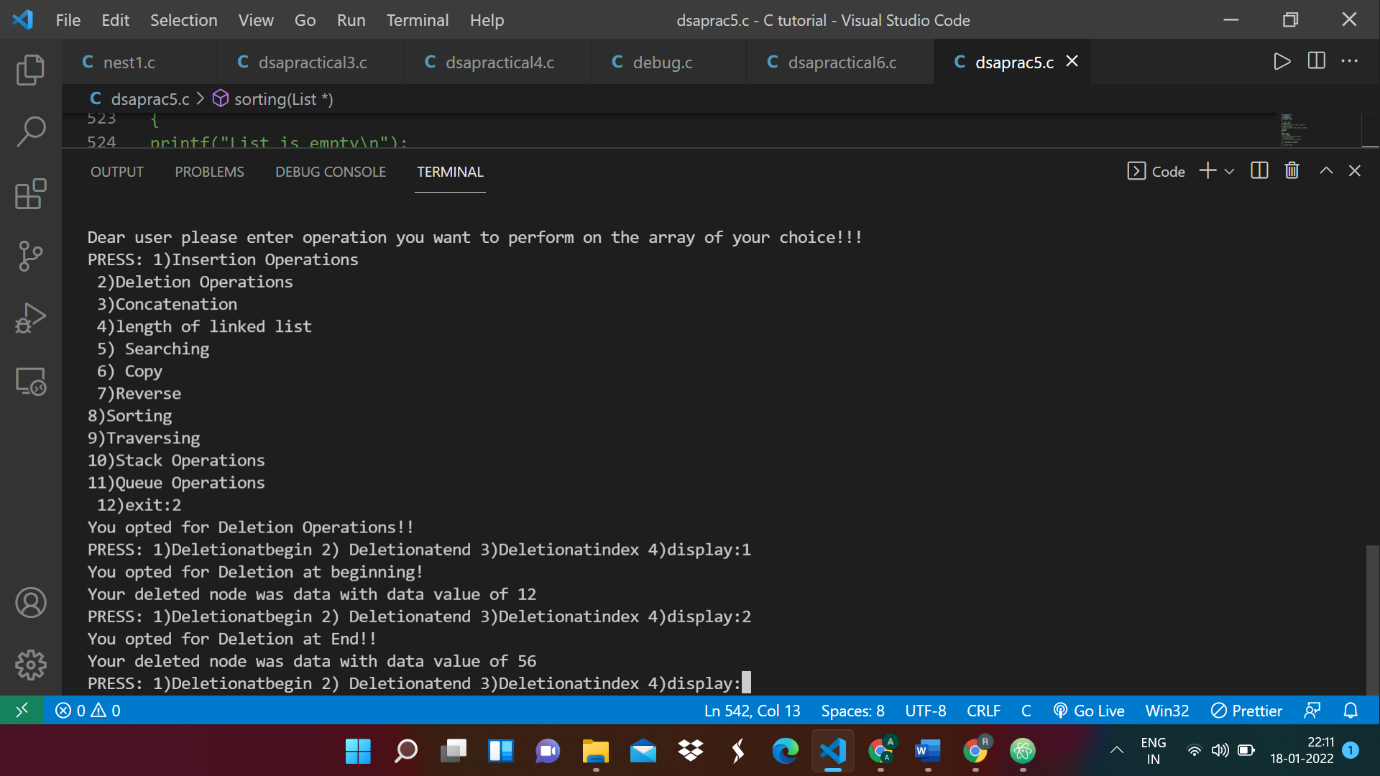
//traversallinkedlist(head);

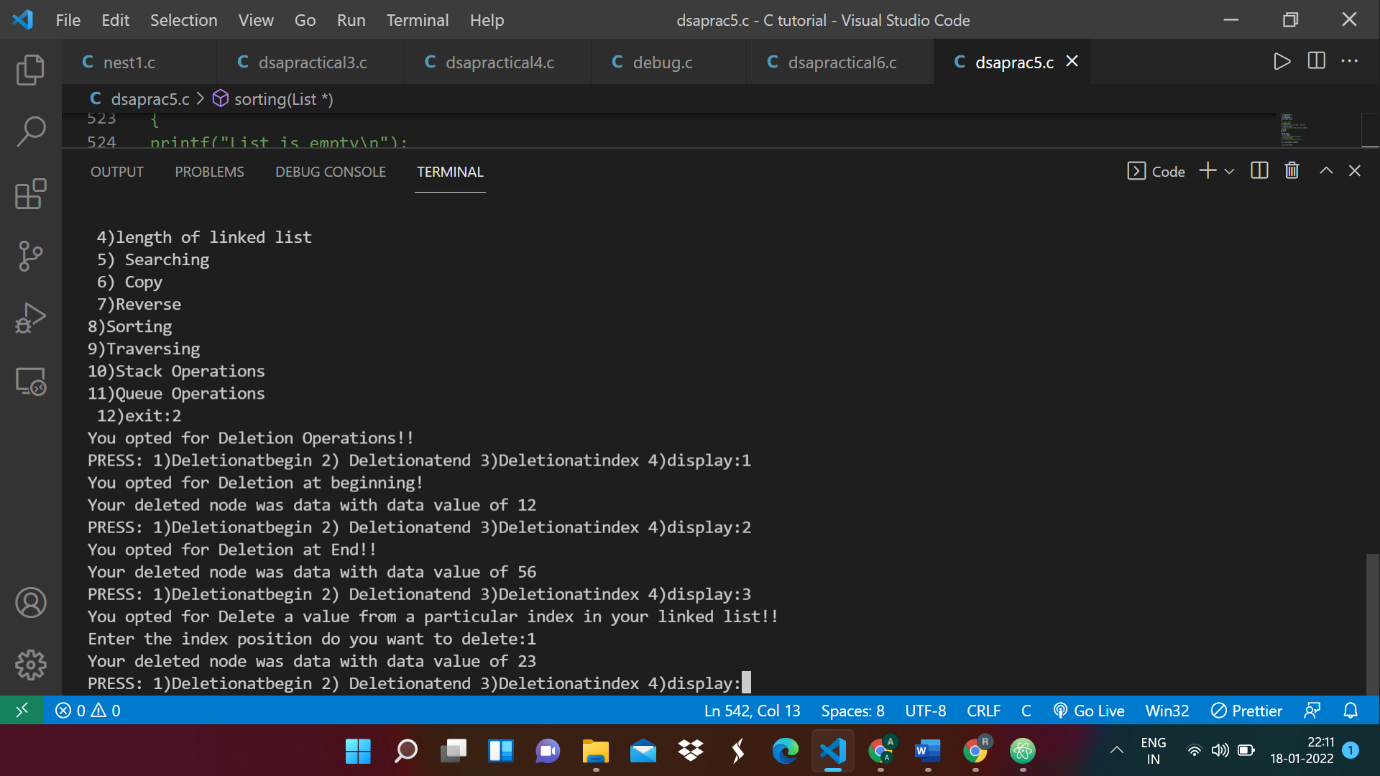
}

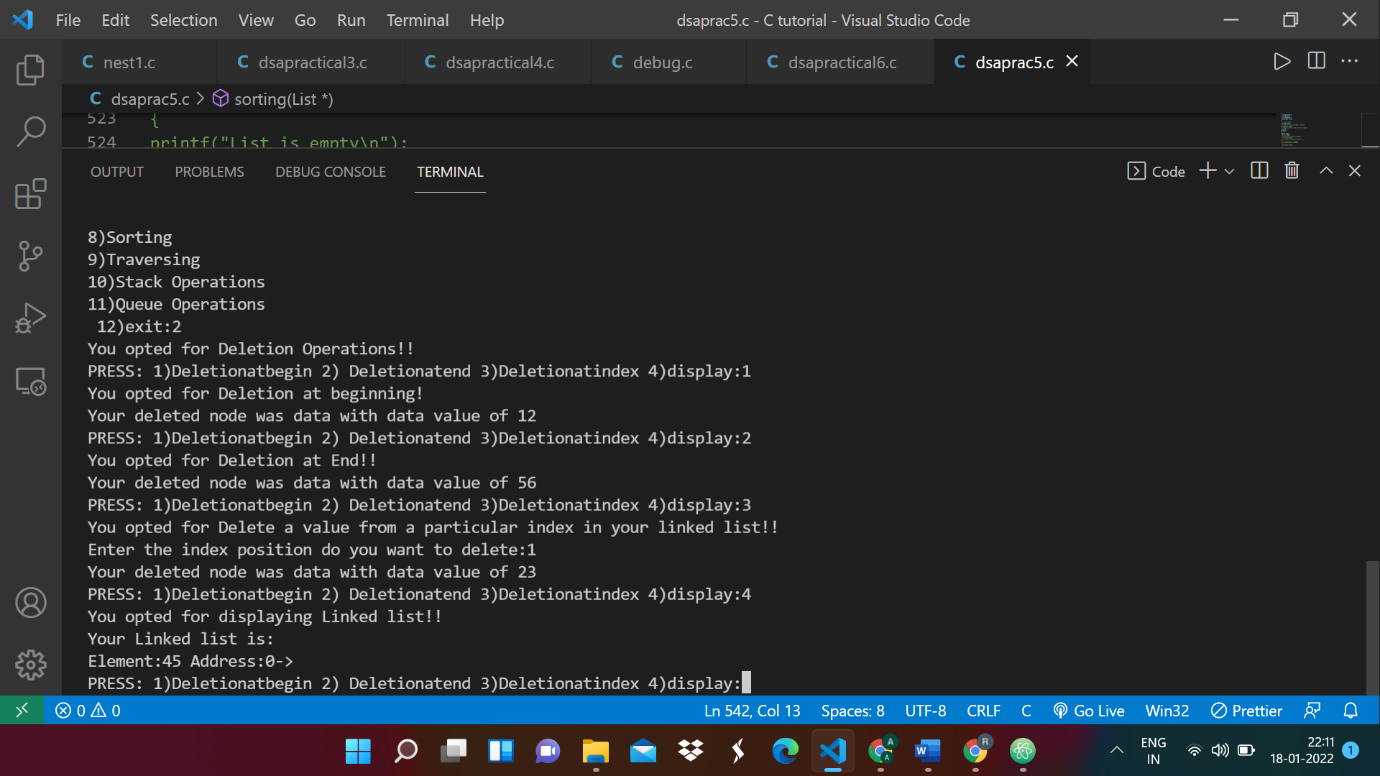
**OUTPUTS:**

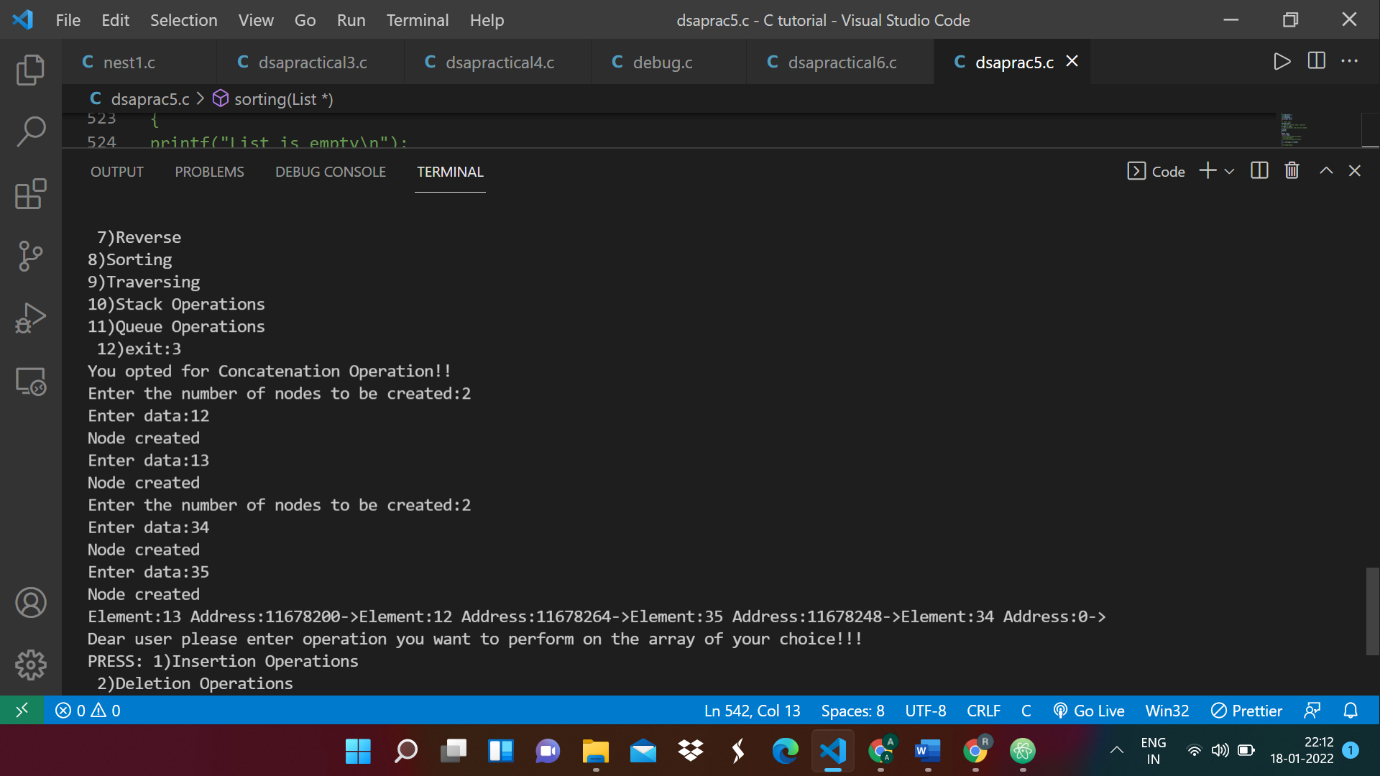


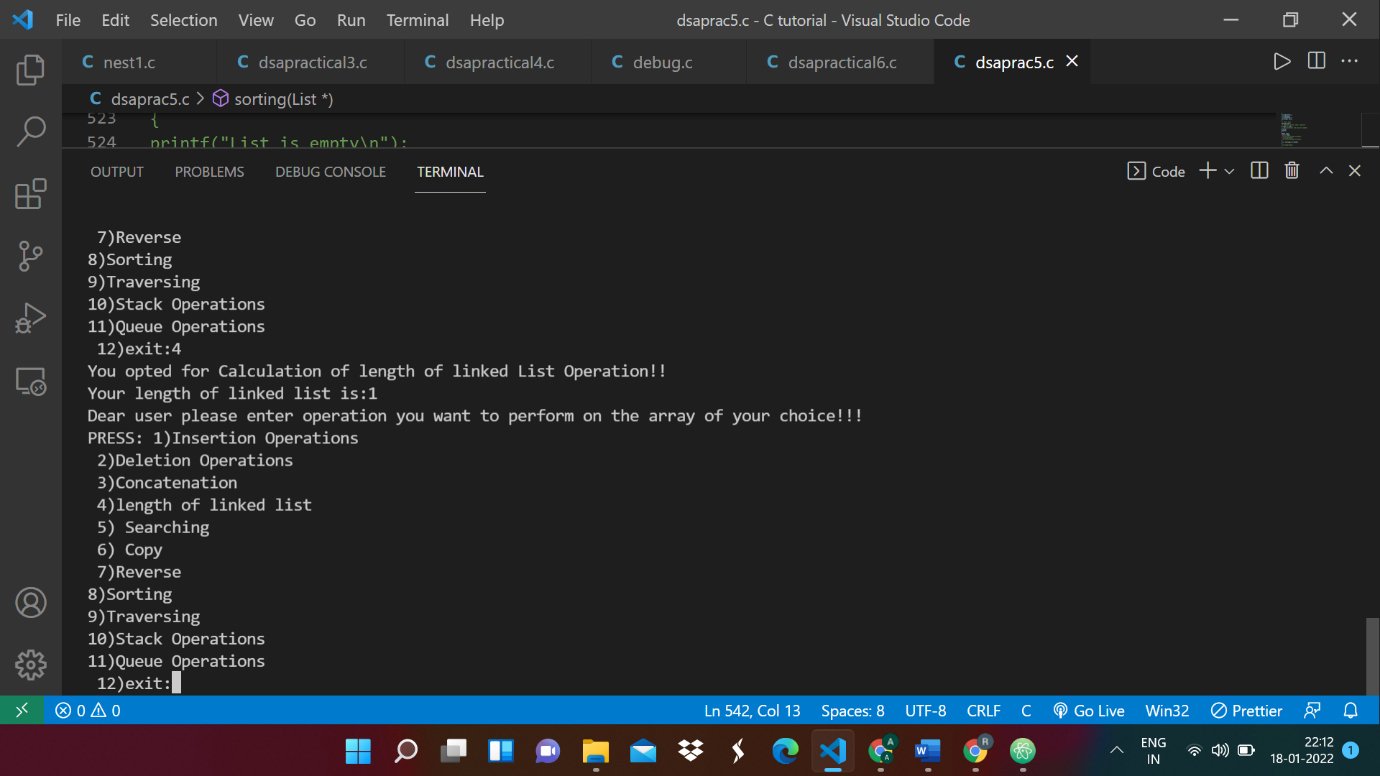


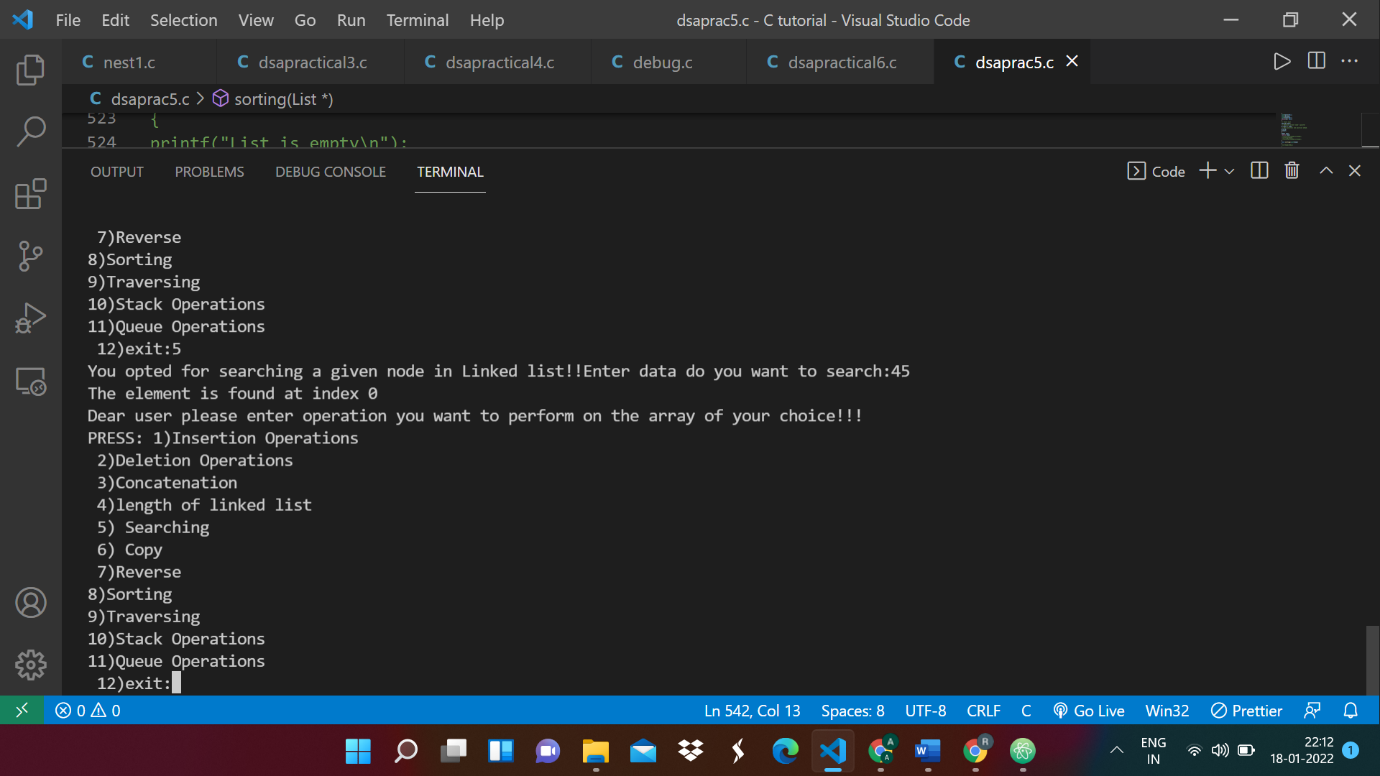


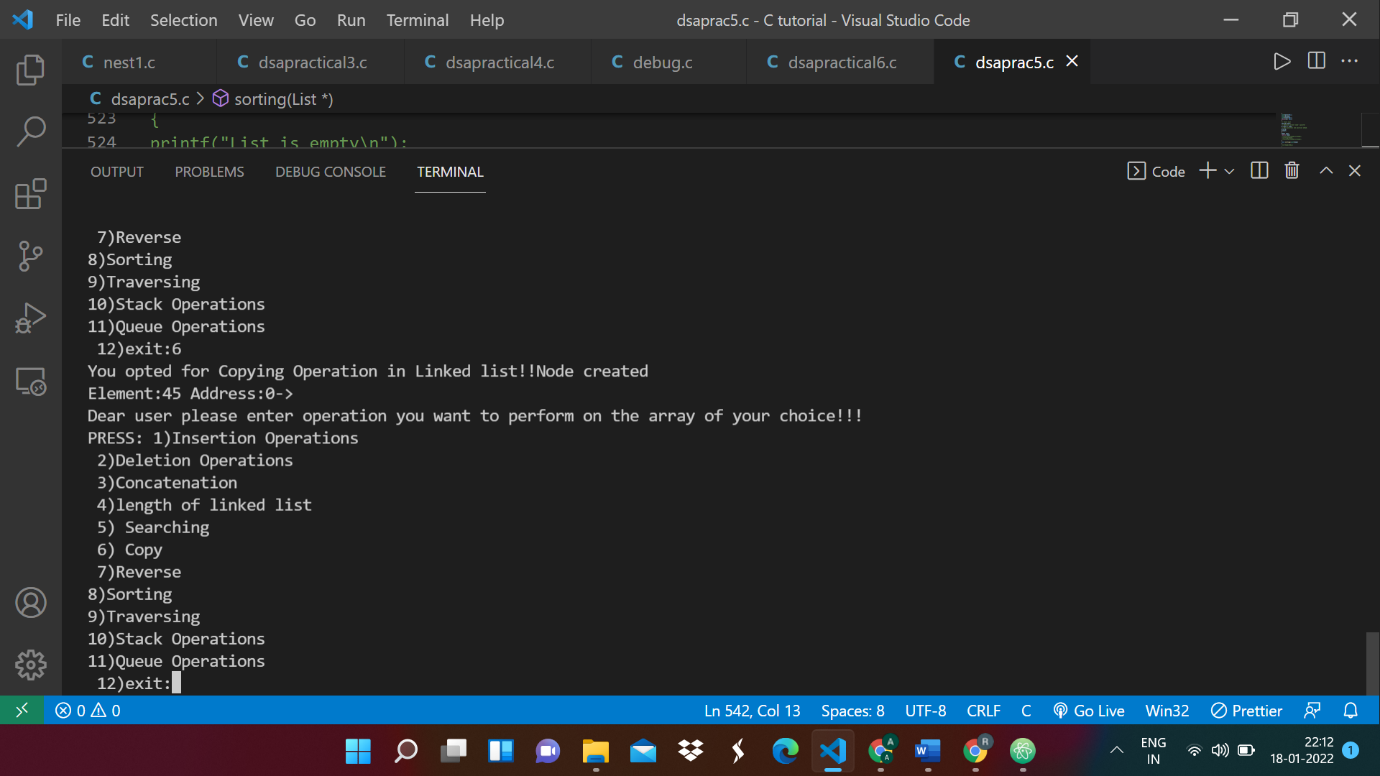


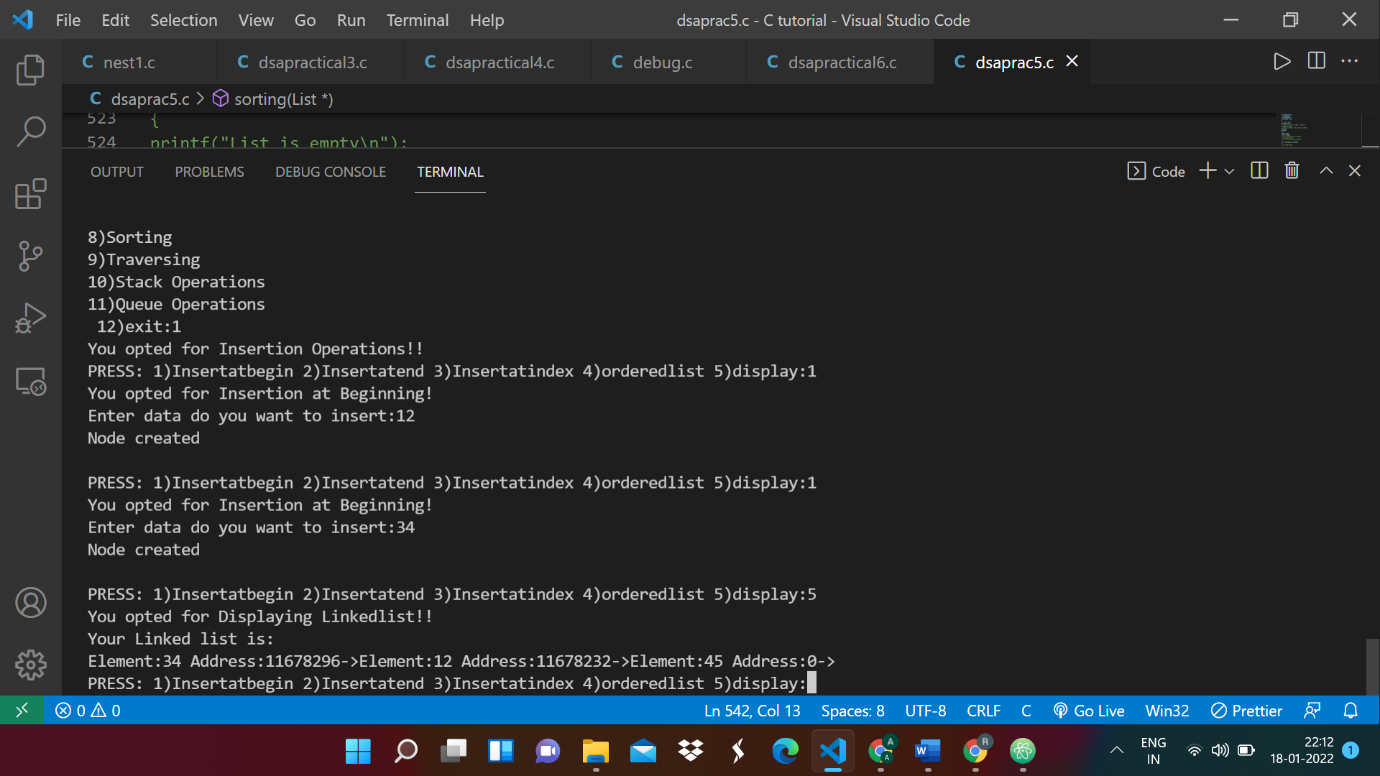


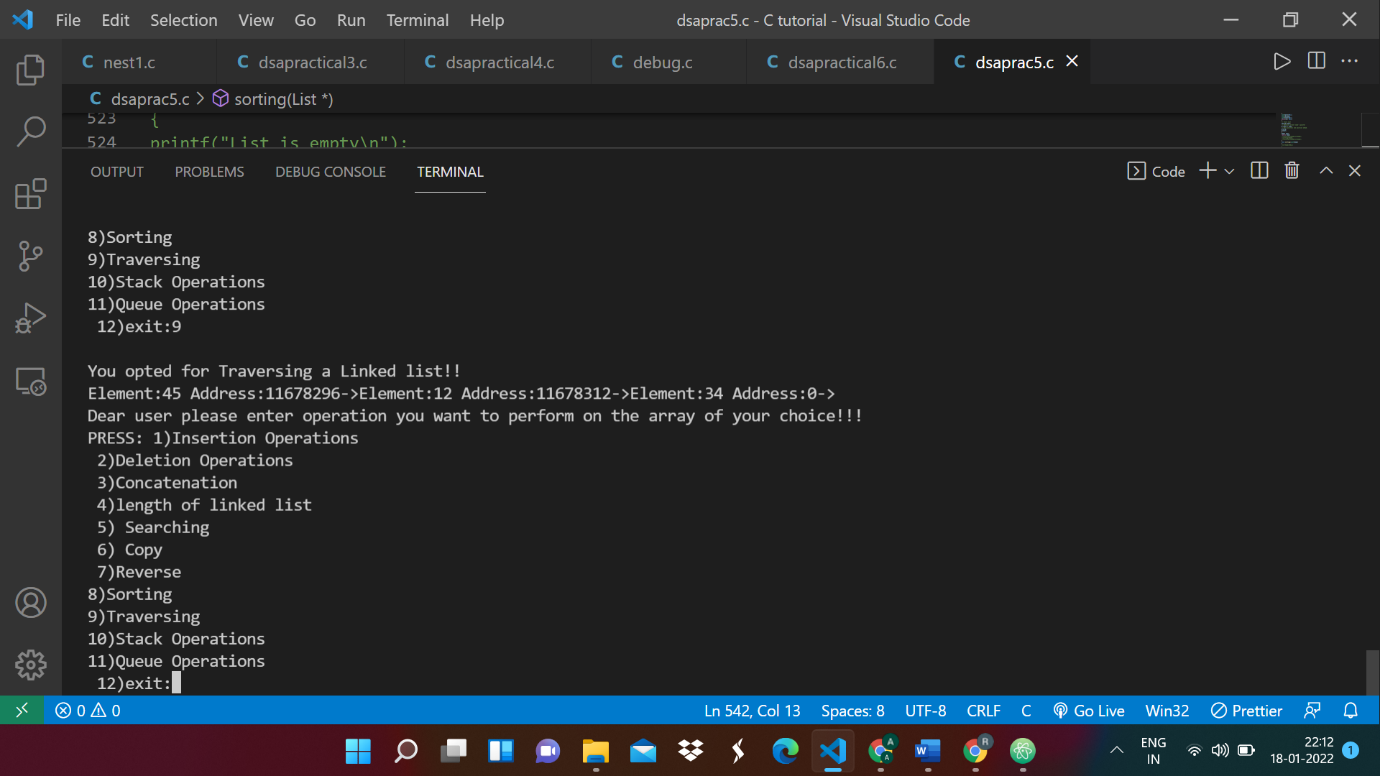


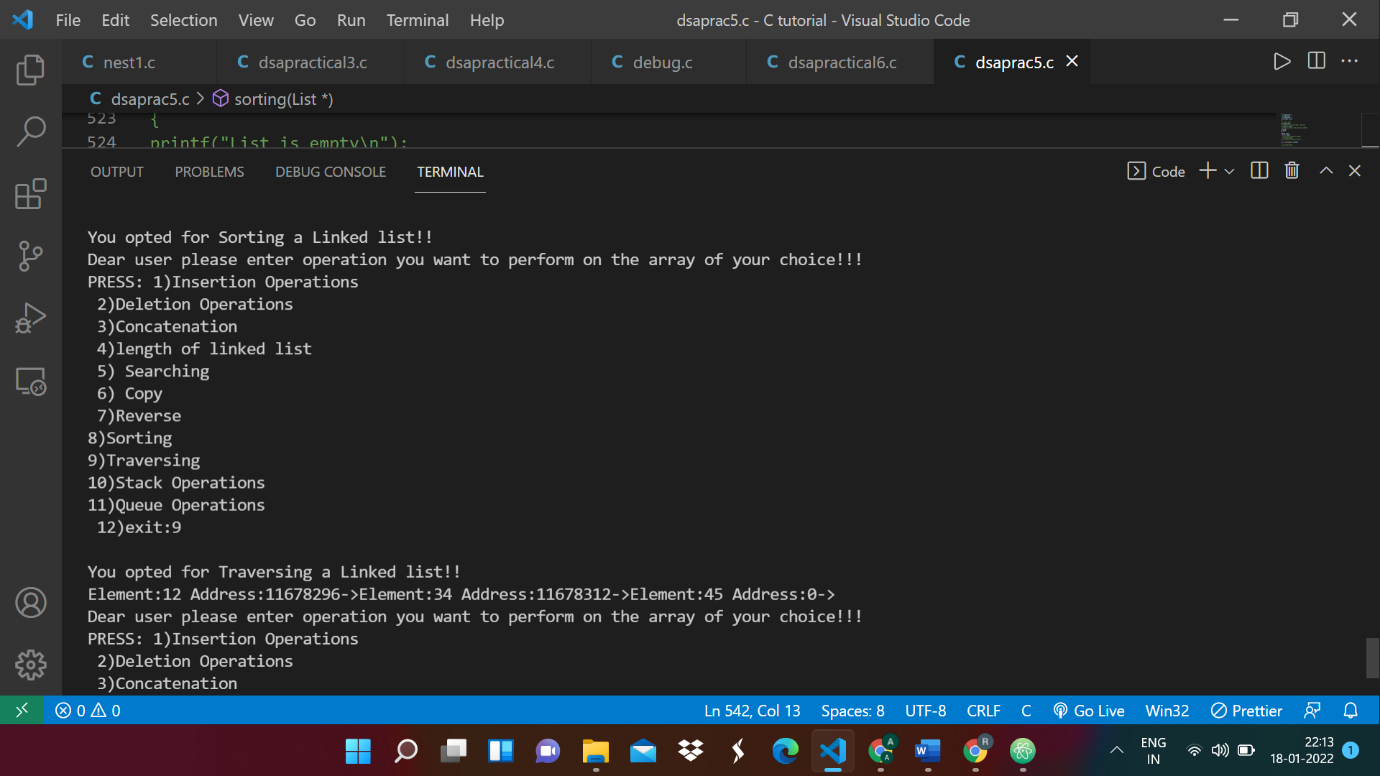


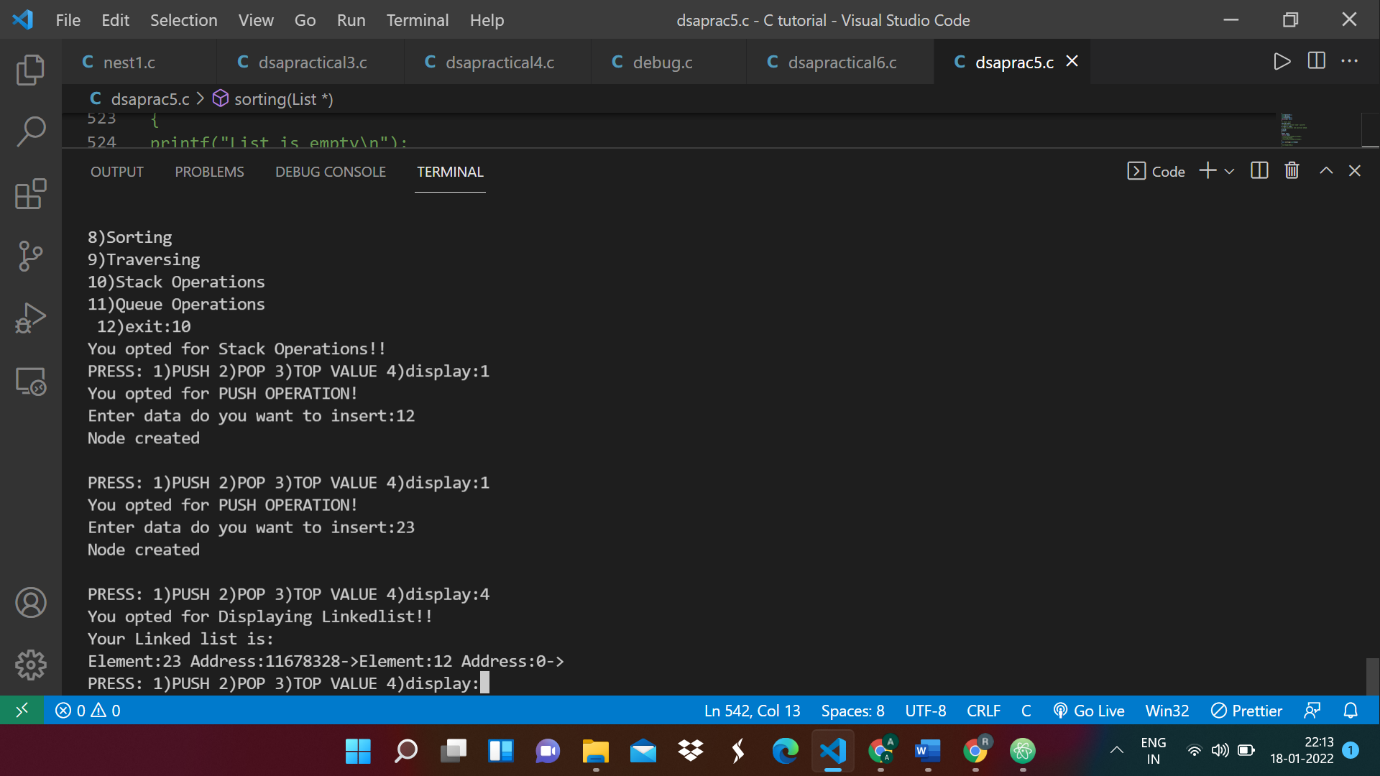


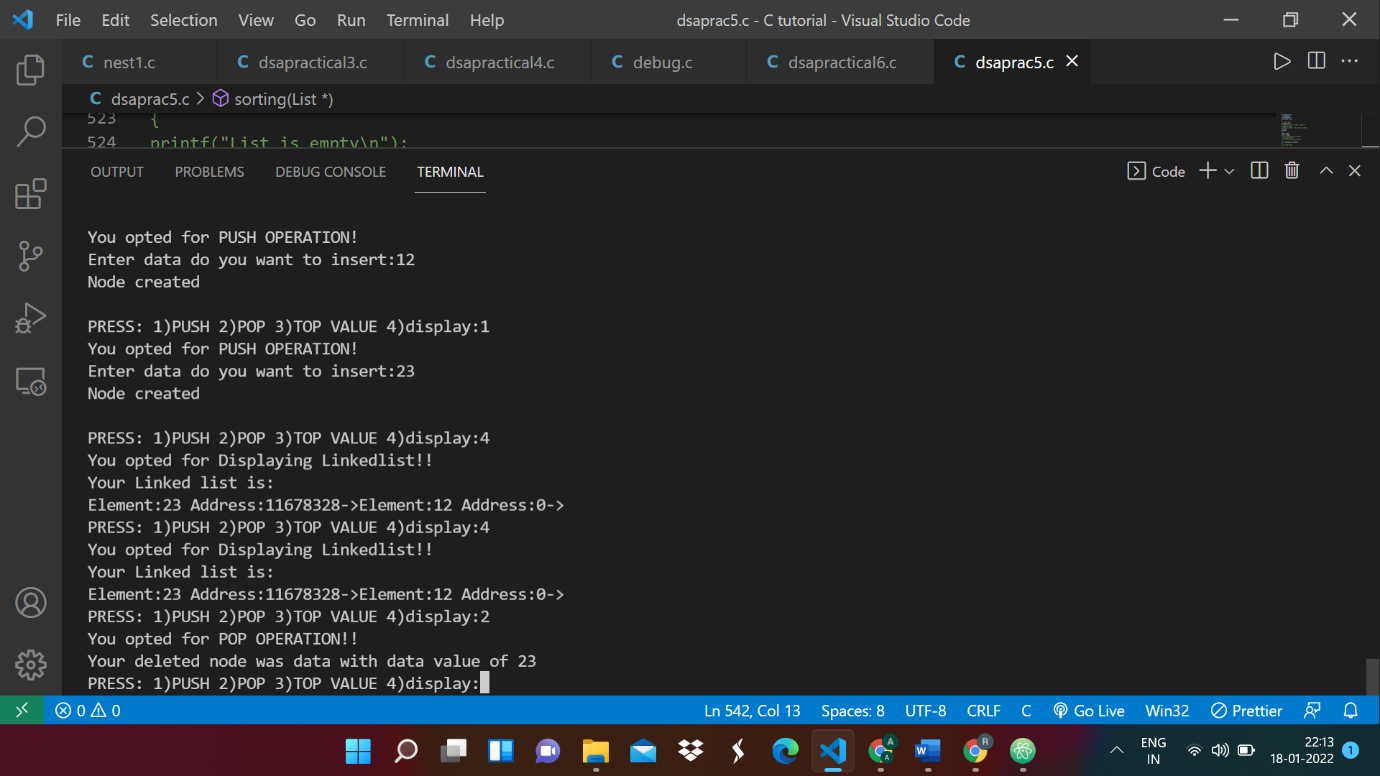


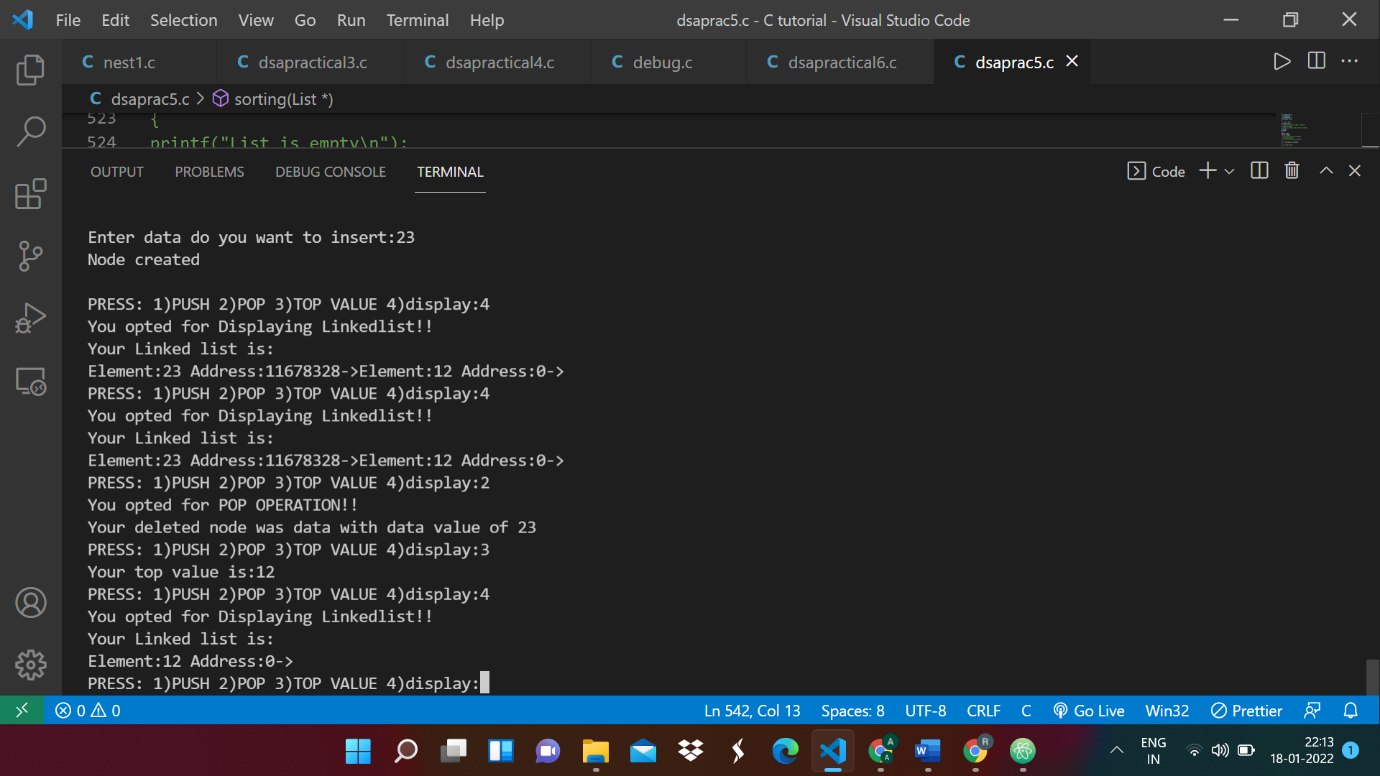


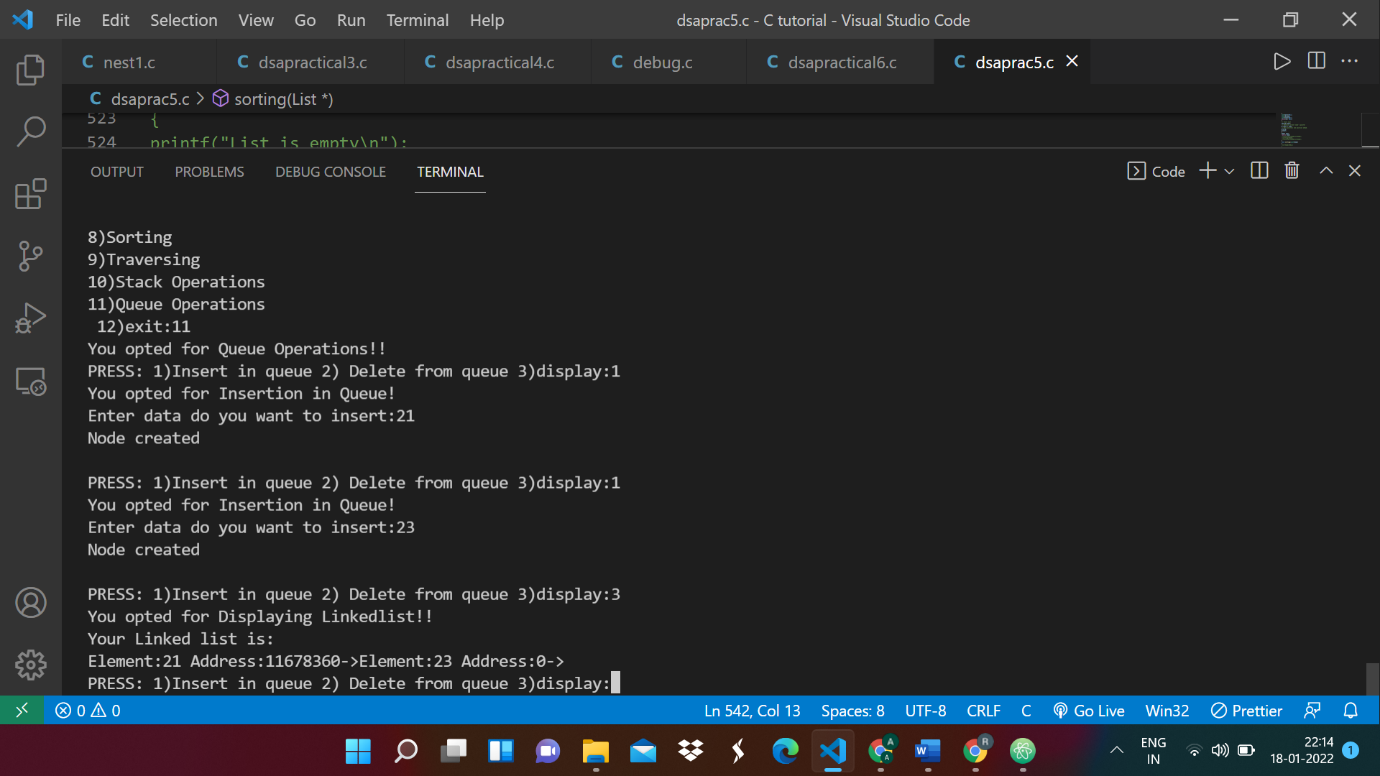












**Conclusion:** In this experiment we learnt about Linked Lists and how it is a linear data structure. Here the elements are not stored at contagious memory locations. The elements here are linked using pointers. So, a linked list consists of nodes where each node contains a data field and a reference (link) to the next node. Linked Lists have a dynamic size and it is easier to insert and delete elements here in comparison to arrays. But, in linked lists you cannot randomly access elements, you have to access elements sequentially from the first node. You also require extra memory space for the pointer for each element of the array. We also implemented linked stacks and linked queues in this experiment.