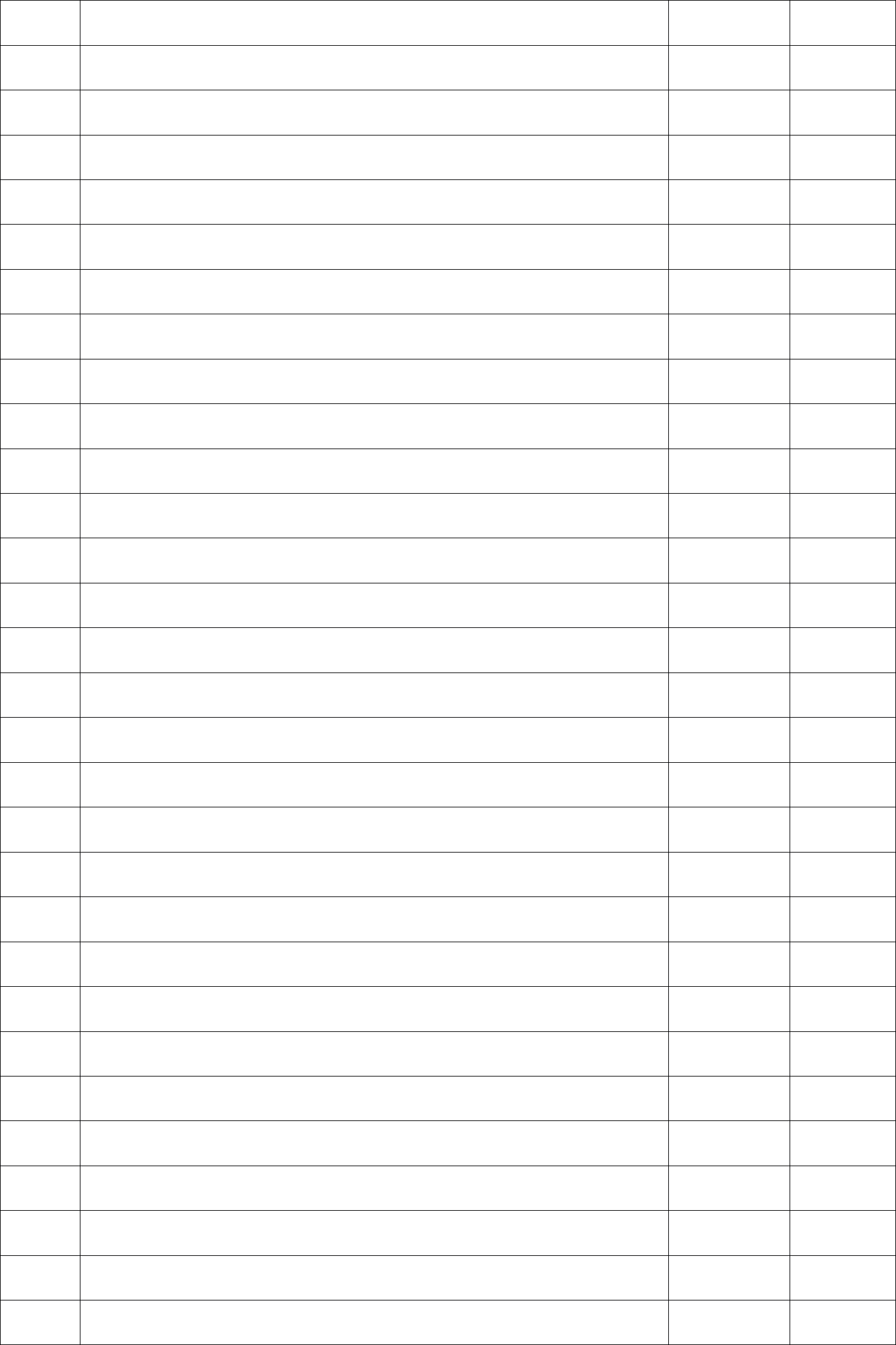
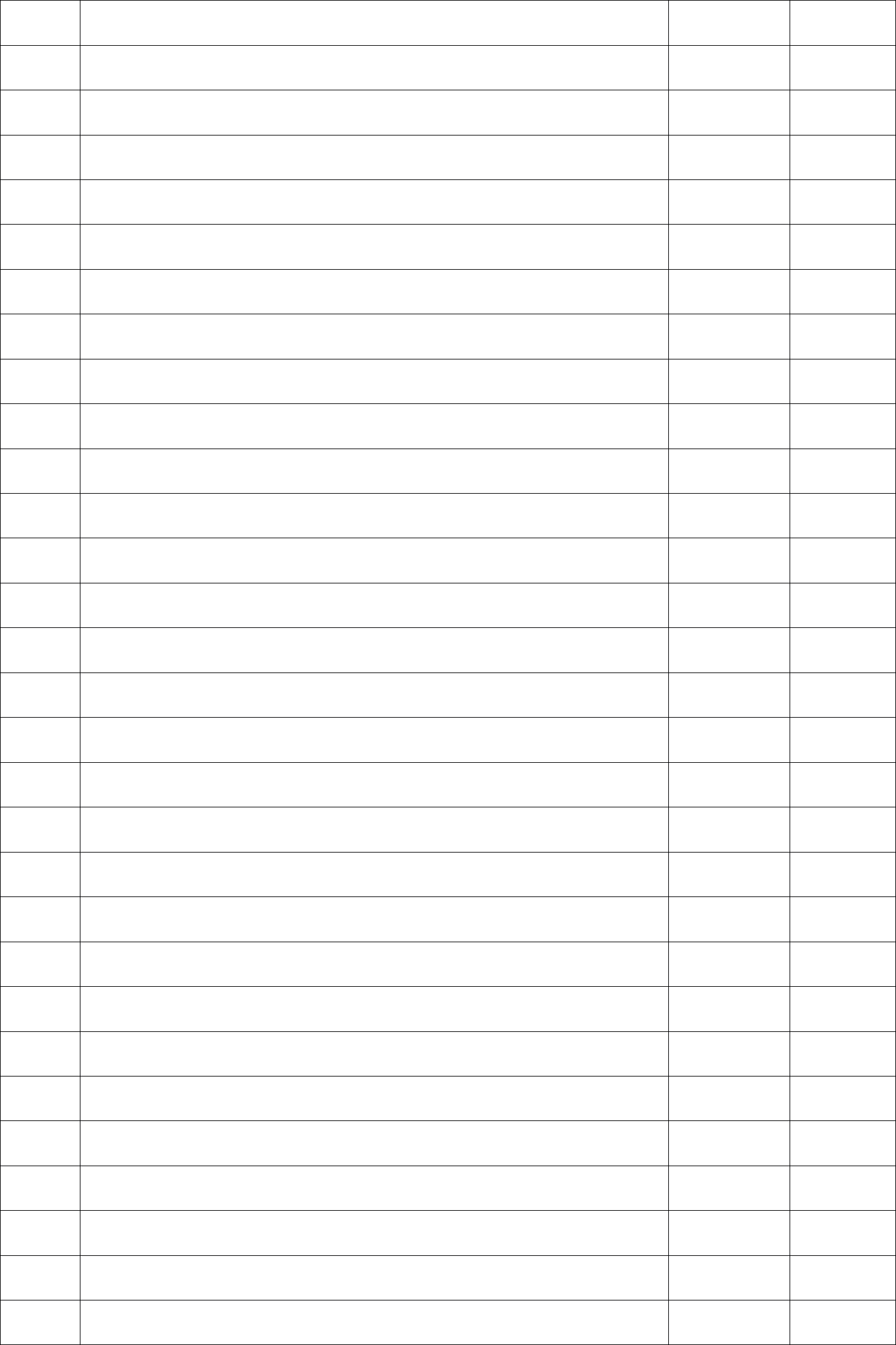
**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **PRACTICALS** | **DATE** | **SIGN** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |





**Q1. Write a program to insert an element in a 1 dimensional array.**

#include<stdio.h>

int main()

{

int arr[30], ele , num , i , loc;

printf("\n Enter the number of elements:"); scanf("%d", &num);

for(i=0 ; i<num ; i++)

{

scanf("%d", &arr[i]);

}

printf("\n Enter the element to be inserted:"); scanf("%d" , &ele);

printf("\n Enter the location");

scanf("%d" , &loc);

**//Create space at the specified location** for(i=num ; i>=loc ; i--)

{

arr[i]=arr[i-1];

}

num++;

arr[loc-1]=ele;

**//Print out the result of insertion**

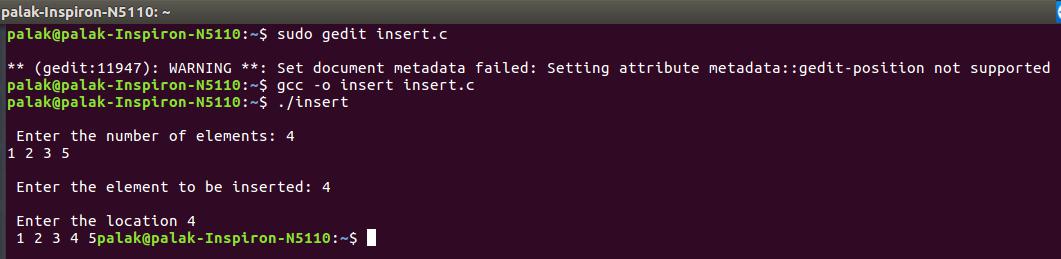
for(i=0 ; i<num ; i++)

printf(" %d" , arr[i]);

return (0);

}

**Output:**



**Q2. Write a program to delete an element from a 1 dimensional array.**

#include<stdio.h>

int main()

{

int arr[30] , num , i , loc;

printf("\n Enter the number of elements:"); scanf ("%d" , &num);

**//Read elements in an array** printf("\n Enter %d elements: " , num);

for(i=0;i<num;i++)

{

scanf("%d" , &arr[i]);

}

**//Read the location**

printf("\n Location of the element to be deleted: "); scanf("%d" , &loc);

**//Loop for the deletion**

while(loc < num)

{

arr[loc - 1] = arr[loc];

loc++;

}

num--; **//number of elements reduced by 1**

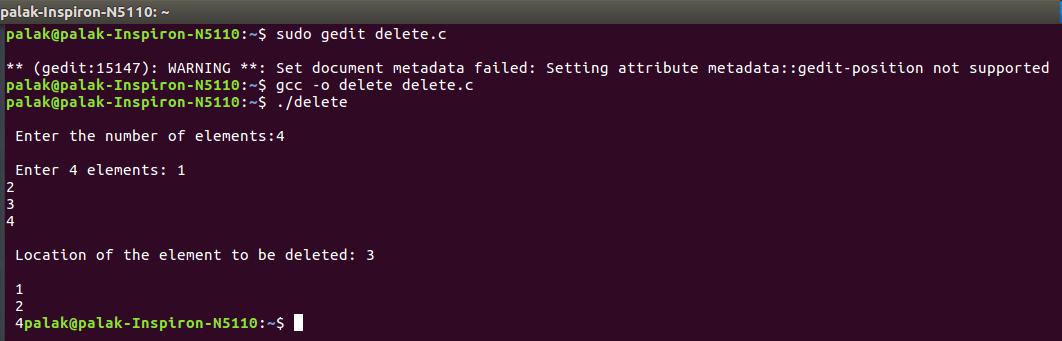
for(i=0;i<num;i++)

printf("\n %d" , arr[i]);

return (0);

}

**Output:**



**Q3. Write a program to search an element in a 1 dimensional array using linear search.**

#include <stdio.h>

int main()

{

int arr[100], search, c, n;

printf("Enter the number of elements in array\n"); scanf("%d",&n);

printf("Enter %d integer(s)\n", n);

for (c = 0; c < n; c++)

scanf("%d", &arr[c]);

printf("Enter the number to search\n"); scanf("%d", &search);

for (c = 0; c < n; c++)

{

if (arr[c] == search) **/\* if required element found \*/**

{

printf("%d is present at location %d.\n", search, c+1);

break;

}

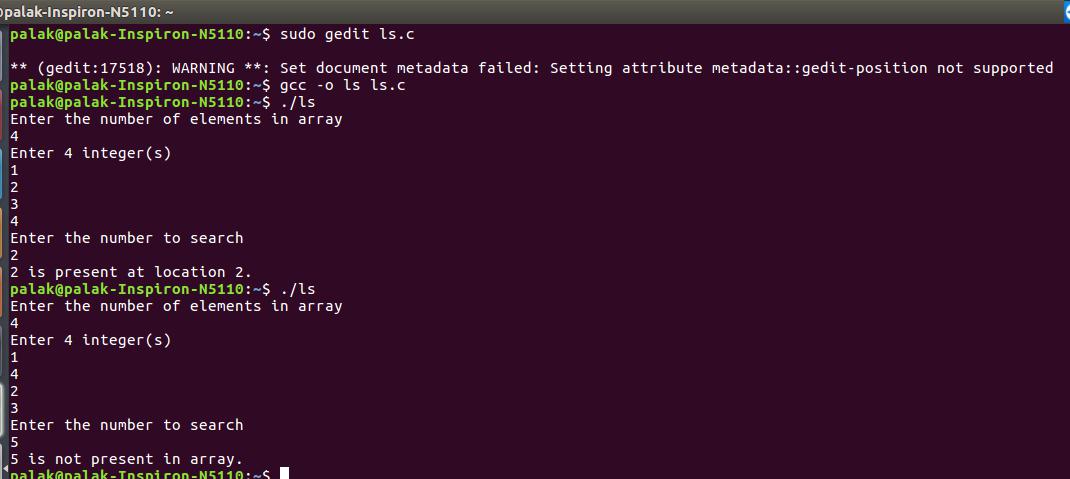
}

if (c == n)

printf("%d is not present in array.\n", search); return 0;

}

**Output:**



**Q4. Write a program to search an element in a 1 dimensional array using binary search.**

#include <stdio.h>

int main()

{

int c, first, last, middle, n, search, arr[100];

printf("Enter number of elements\n"); scanf("%d",&n);

printf("Enter %d integers\n", n);

for (c = 0; c < n; c++)

scanf("%d",&arr[c]);

printf("Enter value to find\n");

scanf("%d", &search);

first = 0;

last = n - 1;

middle = (first+last)/2;

while (first <= last)

{

if (arr[middle] < search)

first = middle + 1;

else if (arr[middle] == search)

{

printf("%d found at location %d.\n", search, middle+1);

break;

}

else

last = middle - 1;

middle = (first + last)/2;

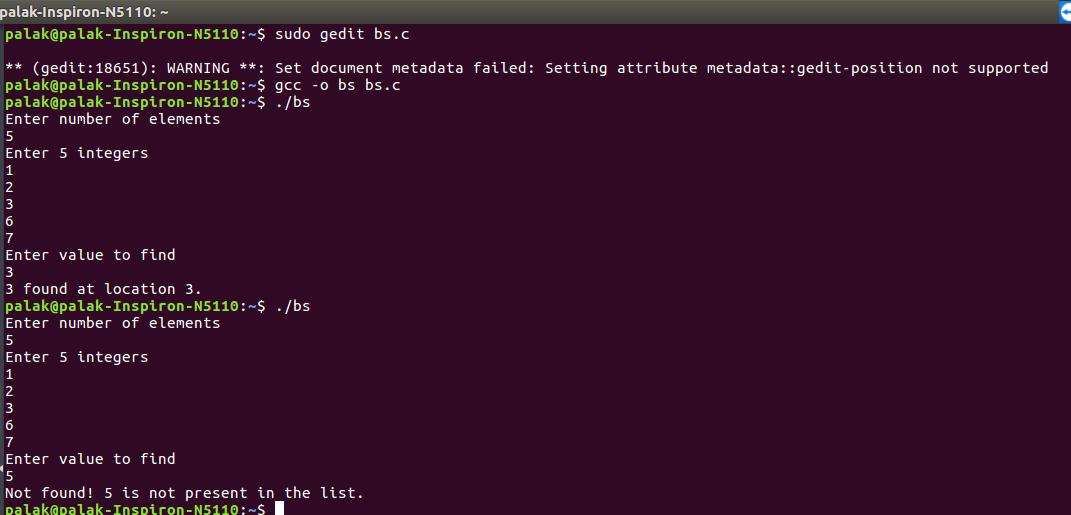
}

if (first > last)

printf("Not found! %d is not present in the list.\n", search); return 0;

}

**Output:**



**Q5. Write a program to sort a 1 dimensional array using bubble sort.**

#include <stdio.h>

int main()

{

int arr[100], n, c, d, swap;

printf("Enter number of elements\n"); scanf("%d", &n);

printf("Enter %d integers\n", n);

for (c = 0; c < n; c++)

scanf("%d", &arr[c]);

for (c = 0 ; c < ( n - 1 ); c++)

{

for (d = 0 ; d < n - c - 1; d++)

{

if (arr[d] > arr[d+1])

{

swap = arr[d];

arr[d] = arr[d+1];

arr[d+1] = swap;

}

}

}

printf("Sorted list in ascending order:\n");

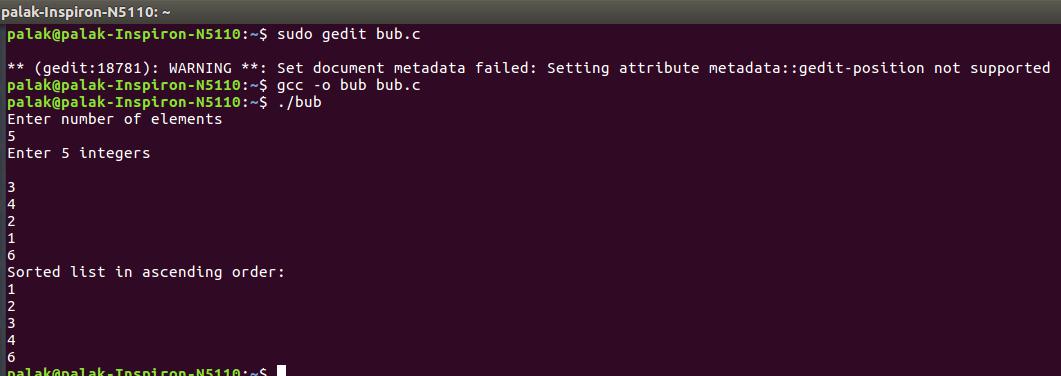
for ( c = 0 ; c < n ; c++ )

printf("%d\n", arr[c]);

return 0;

}

**Output:**



**Q6. Write a program to sort a 1 dimensional array using insertion sort.**

#include <stdio.h>

int main()

{

int n, arr[1000], c, d, t;

printf("Enter number of elements\n"); scanf("%d", &n);

printf("Enter %d integers\n", n);

for (c = 0; c < n; c++)

{

scanf("%d", &arr[c]);

}

for (c = 1 ; c <= n - 1; c++)

{

d = c;

while ( d > 0 && arr[d-1] > arr[d])

{

t = arr[d];

arr[d] = arr[d-1];

arr[d-1] = t;

d--;

}

}

printf("Sorted list in ascending order:\n"); for (c = 0; c <= n - 1; c++)

{

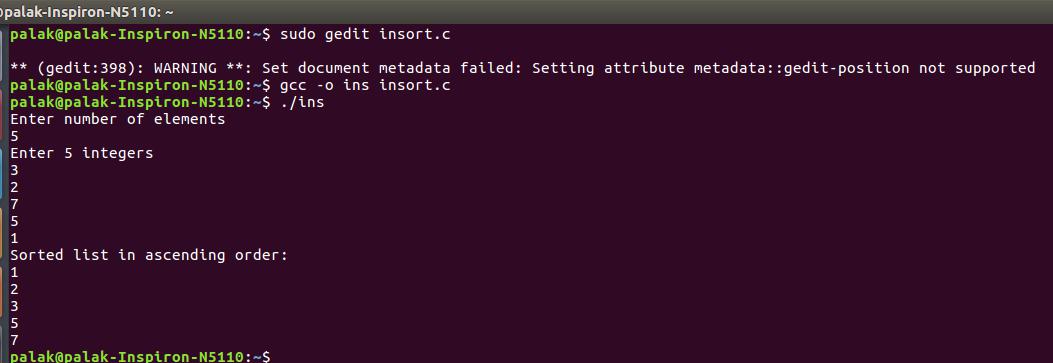
printf("%d\n", arr[c]);

}

return 0;

}

**Output:**



**Q7. Write a program to sort a 1 dimensional array using selection sort.**

#include <stdio.h>

int main()

{

int arr[100], n, c, d, pos, swap;

printf("Enter number of elements\n"); scanf("%d", &n);

printf("Enter %d integers\n", n);

for ( c = 0 ; c < n ; c++ )

scanf("%d", &arr[c]);

for ( c = 0 ; c < ( n - 1 ) ; c++ )

{

pos = c;

for ( d = c + 1 ; d < n ; d++ )

{

if ( arr[pos] > arr[d] )

pos = d;

}

if ( pos != c )

{

swap = arr[c];

arr[c] = arr[pos];

arr[pos] = swap;

}

}

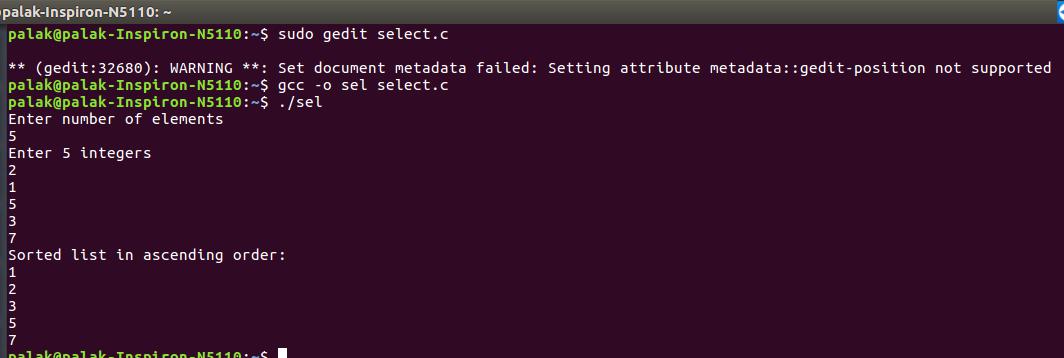
printf("Sorted list in ascending order:\n"); for ( c = 0 ; c < n ; c++ )

printf("%d\n", arr[c]);

return 0;

}

**Output:**



**Q8. Write a program to merge two 1 dimensional sorted arrays.**

#include <stdio.h>

void main()

{

int arr1[50], arr2[50], arr3[100], m, n, i, j, k = 0;

printf("\n Enter size of array Array 1: "); scanf("%d", &m);

printf("\n Enter sorted elements of array 1: \n"); for (i = 0; i < m; i++)

{

scanf("%d", &arr1[i]);

}

printf("\n Enter size of array 2: ");

scanf("%d", &n);

printf("\n Enter sorted elements of array 2: \n"); for (i = 0; i < n; i++)

{

scanf("%d", &arr2[i]);

}

i = 0;

j = 0;

while (i < m && j < n)

{

if (arr1[i] < arr2[j])

{

arr3[k] = arr1[i];

i++;

}

else

{

arr3[k] = arr2[j];

j++;

}

k++;

}

if (i >= m)

{

while (j < n)

{

arr3[k] = arr2[j];

j++;

k++;

}

}

if (j >= n)

{

while (i < m)

{

arr3[k] = arr1[i];

i++;

k++;

}

}

printf("\n After merging: \n");

for (i = 0; i < m + n; i++)

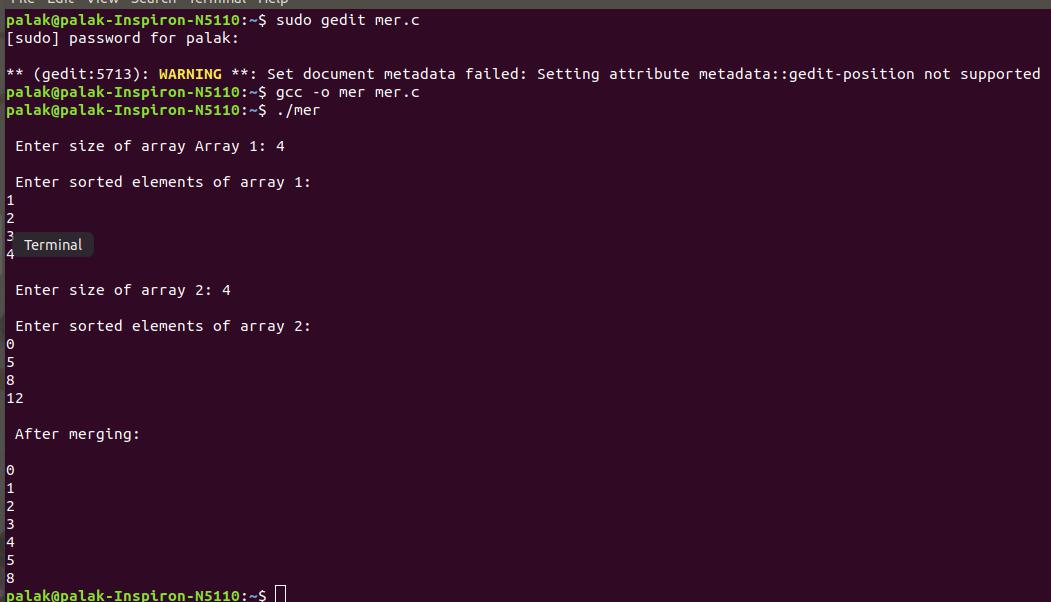
{

printf("\n%d", arr3[i]);

}

}

**Output:**



**Q9. Write a program to concatenate two 1 dimensional arrays.**

#include <stdio.h>

int main()

{

int array[100],even[50],odd[50];

int loop, index, e\_len, o\_len , i ;

printf("\n Enter size of even array: "); scanf("%d", &e\_len);

printf("\n Enter elements of even array: \n");

for (i = 0; i < e\_len; i++)

{

scanf("%d", &even[i]);

}

printf("\n Enter size of odd array: "); scanf("%d", &o\_len);

printf("\n Enter elements of odd array : \n");

for (i = 0; i < o\_len; i++)

{

scanf("%d", &odd[i]);

}

index = 0;

for(loop = 0; loop < e\_len; loop++)

{

array[index] = even[loop];

index++;

}

for(loop = 0; loop < o\_len; loop++)

{

array[index] = odd[loop];

index++;

}

printf("\nEven -> ");

for(loop = 0; loop < e\_len; loop++)

printf(" %d", even[loop]);

printf("\nOdd -> ");

for(loop = 0; loop < o\_len; loop++)

printf(" %d", odd[loop]);

printf("\nConcat -> ");

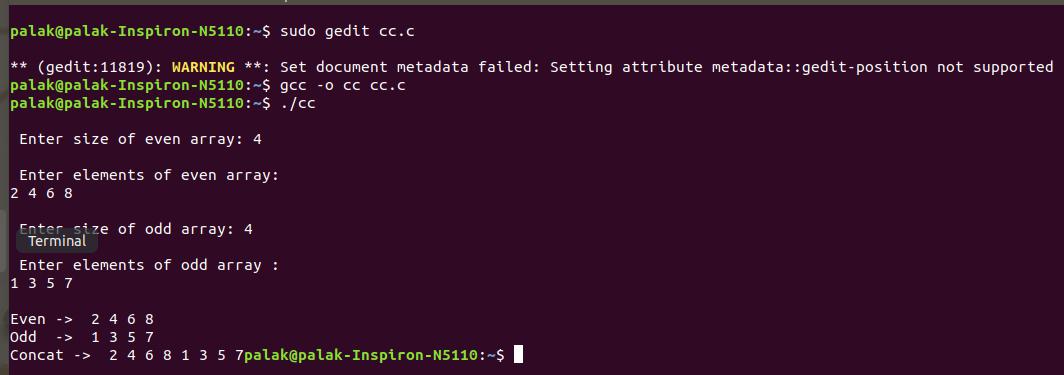
for(loop = 0; loop < e\_len+o\_len; loop++)

printf(" %d", array[loop]);

return 0;

}

**Output:**



**Q10. Write a program to copy elements of one 1 dimensional array to another.**

#include<stdio.h>

int main()

{

int arr1[30], arr2[30], i, num;

printf("\nEnter no of elements :");

scanf("%d", &num);

**//Accepting values into Array**

printf("\nEnter the values :");

for (i = 0; i < num; i++)

{

scanf("%d", &arr1[i]);

}

**/\* Copying data from array 'a' to array 'b \*/** for (i = 0; i < num; i++)

{

arr2[i] = arr1[i];

}

**//Printing of all elements of array**

printf("The copied array is :");

for (i = 0; i < num; i++)

printf("\narr2[%d] = %d", i, arr2[i]);

return (0);

}

**Output:**



**Q11. Write a program to search an element in a 2D array .**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int n,i,j,s,c=0;

printf("\nEnter the size of Matrix.\n"); scanf("%d",&n);

int \*\*a;

a = malloc(n\*sizeof(int\*));

printf("Enter the elements in the Matrix.\n"); for(i=0;i<n;i++)

{

a[i] = malloc(n\*sizeof(int));

for(j=0;j<n;j++)

{

printf("[%d,%d] : ",i+1,j+1);

scanf("%d",&a[i][j]);

}

}

printf("Enter the element to be searched.\n");

scanf("%d",&s);

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

if(s == a[i][j])

{

printf("Element %d at position Number [%d,

%d].\n",s,i+1,j+1);

c++;

}

}

}

if(c==0)

{

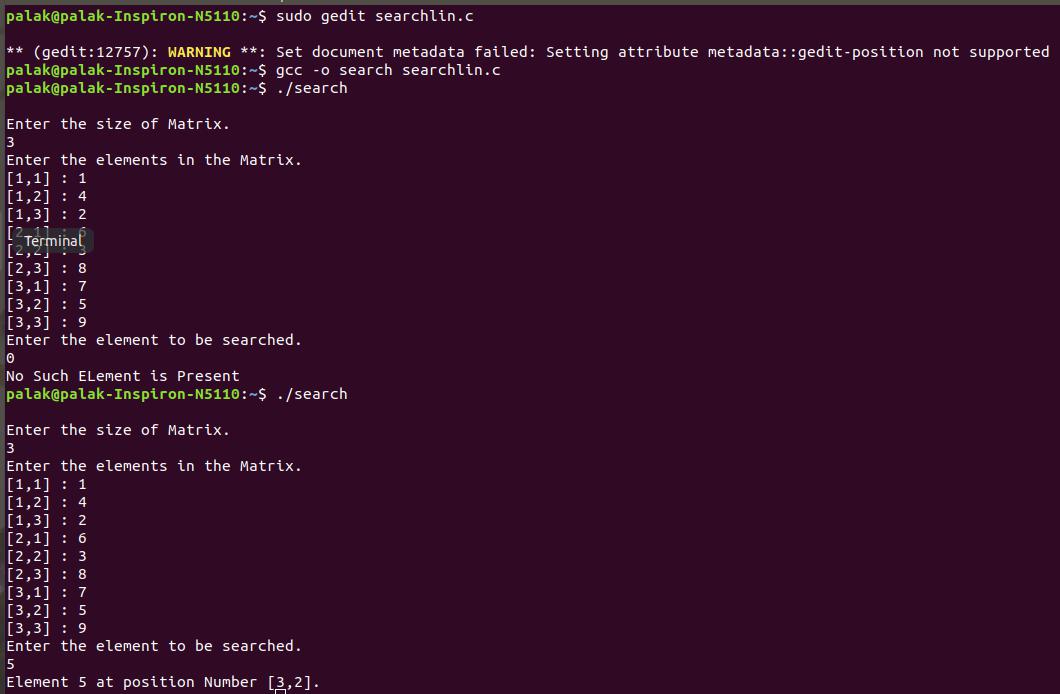
printf("No Such ELement is Present\n");

}

return 0;

}

**Output:**



**Q12. Write a program to add two 2 dimensional arrays.**

#include<stdio.h>

void main()

{

int a[3][3],b[3][3],c[3][3],i,j;

printf("Enter the First matrix->");

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

scanf("%d",&a[i][j]);

}

}

printf("\nEnter the Second matrix->");

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

scanf("%d",&b[i][j]);

}

}

printf("\nThe First matrix is\n");

for(i=0;i<3;i++)

{

printf("\n");

for(j=0;j<3;j++)

{

printf("%d\t",a[i][j]);

}

}

printf("\nThe Second matrix is\n");

for(i=0;i<3;i++)

{

printf("\n");

for(j=0;j<3;j++)

{

printf("%d\t",b[i][j]);

}

}

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

c[i][j]=a[i][j]+b[i][j];

}

printf("\nThe Addition of two matrix is\n"); for(i=0;i<3;i++)

{

printf("\n");

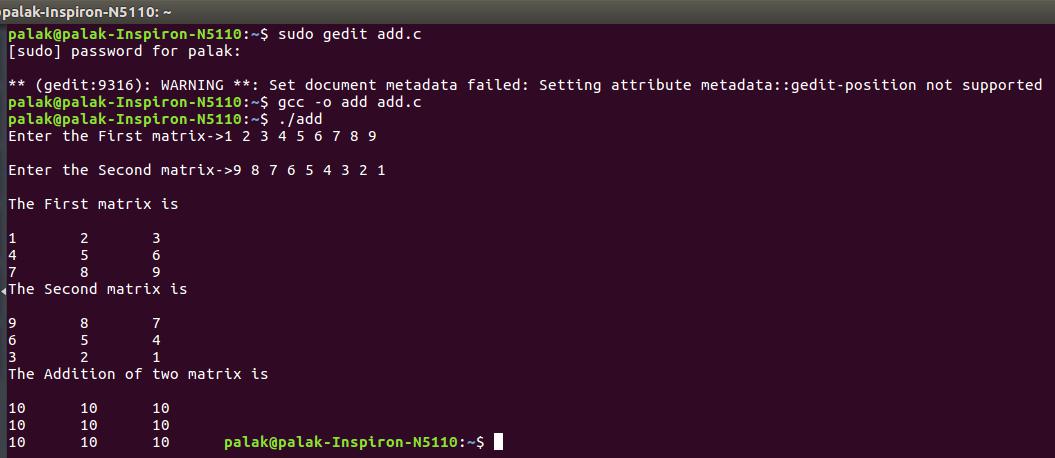
for(j=0;j<3;j++)

printf("%d\t",c[i][j]);

}

}

**Output:**



**Q13. Write a program to subtract two 2 dimensional arrays.**

#include<stdio.h>

void main()

{

int a[3][3],b[3][3],c[3][3],i,j;

printf("Enter the First matrix->");

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

scanf("%d",&a[i][j]);

}

}

printf("\nEnter the Second matrix->");

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

{

scanf("%d",&b[i][j]);

}

}

printf("\nThe First matrix is\n");

for(i=0;i<3;i++)

{

printf("\n");

for(j=0;j<3;j++)

{

printf("%d\t",a[i][j]);

}

}

printf("\nThe Second matrix is\n");

for(i=0;i<3;i++)

{

printf("\n");

for(j=0;j<3;j++)

{

printf("%d\t",b[i][j]);

}

}

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

c[i][j]=a[i][j]-b[i][j];

}

printf("\nThe subtraction of two matrix is\n"); for(i=0;i<3;i++)

{

printf("\n");

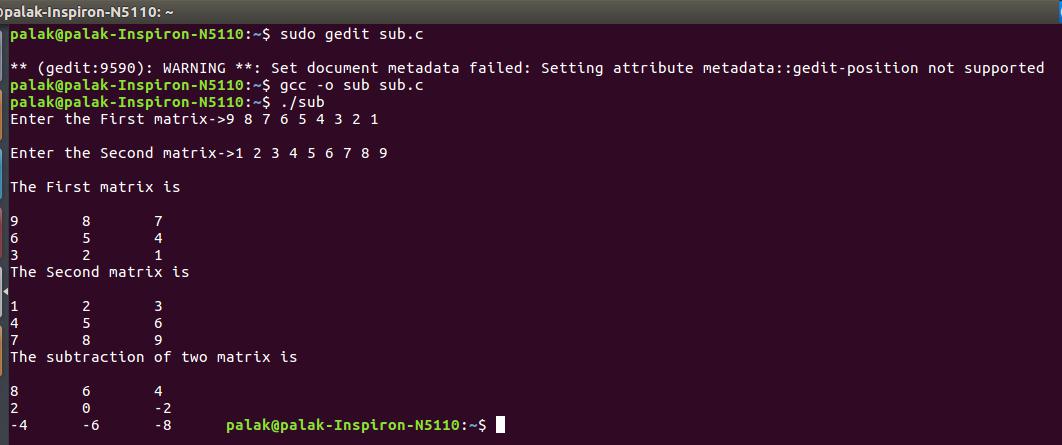
for(j=0;j<3;j++)

printf("%d\t",c[i][j]);

}

}

**Output:**



**Q14. Write a program to multiply two 2 dimensional arrays.**

#include <stdio.h>

void main()

{

int m,n,p,q,c,d,k,sum=0;

int first[10][10],second[10][10],multiply[10][10];

printf("Enter the number of rows and columns of first matrix\n"); scanf("%d%d",&m,&n);

printf("Enter the elements of first matrix\n");

for(c=0;c<m;c++)

for(d=0;d<n;d++)

scanf("%d",&first[c][d]);

printf("Enter the number of rows and columns of second matrix\n");

scanf("%d%d",&p,&q);

if(n!=p)

printf("Matrices with entered orders can't be multiplied with each other.\n");

else

{

printf("Enter the elements of second matrix\n"); for(c=0;c<p;c++)

for(d=0;d<q;d++)

scanf("%d",&second[c][d]);

for(c=0;c<m;c++)

{

for(d=0;d<q;d++)

{

for(k=0;k<p;k++)

{

sum=sum+first[c][k]\*second[k][d];

}

multiply[c][d]=sum;

sum=0;

}

}

printf("Product of entered matrices:-\n");

for(c=0;c<m;c++)

{

for(d=0;d<q;d++)

printf("%d\t",multiply[c][d]);

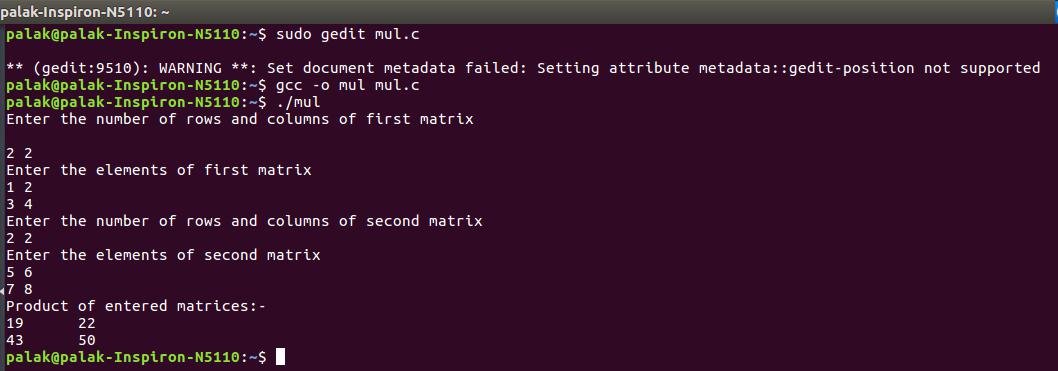
printf("\n");

}

}

}

**Output:**



**Q15. Write a program to find the transpose of a 2 dimensional array.**

#include <stdio.h>

int main()

{

int m, n, c, d, matrix[10][10], transpose[10][10];

printf("Enter the number of rows and columns of matrix\n"); scanf("%d%d", &m, &n);

printf("Enter the elements of matrix\n"); for (c = 0; c < m; c++) for(d = 0; d < n; d++)

scanf("%d\t",&matrix[c][d]);

printf("\n");

for (c = 0; c < m; c++) for( d = 0 ; d < n ; d++ ) transpose[d][c] = matrix[c][d]; printf("Transpose of entered matrix :-\n"); for (c = 0; c < n; c++)

{

for (d = 0; d < m; d++)

printf("%d\t",transpose[c][d]);

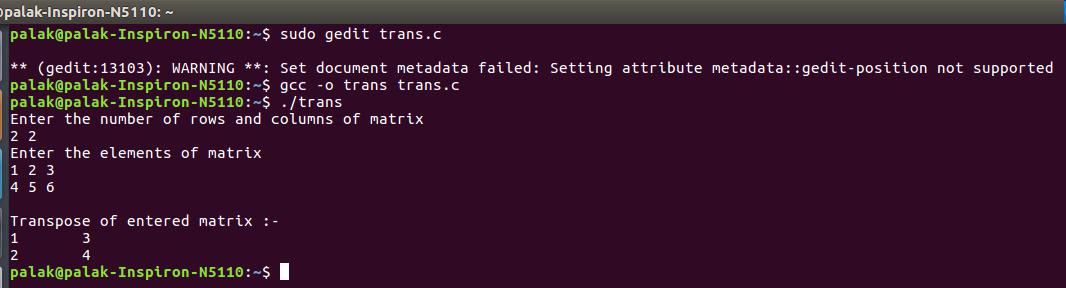
printf("\n");

}

return 0;

}

**Output:**



**Q16. WAP to perform the following operations on a singly linked list.**

**a.) create a node.**

**b.) insert a node.**

**c.)delete a node.**

**d.)search a node.**

**e.)count the number of nodes.**

**f.)traverse the list.**

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

struct node

{

int data;

struct node \*next;

};

**//FUNCTION PROTOTYPES**

struct node \*start = NULL;

struct node \*create\_ll(struct node \*); struct node \*display(struct node \*); struct node \*insert\_beg(struct node \*); struct node \*insert\_end(struct node \*); struct node \*insert\_before(struct node \*); struct node \*insert\_after(struct node \*); struct node \*delete\_beg(struct node \*);

struct node \*delete\_end(struct node \*); struct node \*delete\_node(struct node \*); struct node \*delete\_after(struct node \*); struct node \*delete\_list(struct node \*); struct node \*count\_node(struct node \*); struct node \*search\_list(struct node \*);

int main(int argc, char \*argv[])

{

int option;

do

{

printf("\n\n \*\*\*\*\*MAIN MENU \*\*\*\*\*");

printf("\n 1: Create a list");

printf("\n 2: Add a node at the beginning");

printf("\n 3: Add a node at the end");

printf("\n 4: Add a node before a given node");

printf("\n 5: Add a node after a given node");

printf("\n 6: Delete a node from the beginning");

printf("\n 7: Delete a node from the end");

printf("\n 8: Delete a given node");

printf("\n 9: Delete a node after a given node");

printf("\n 10: Delete the entire list");

printf("\n 11: Search a node with given value");

printf("\n 12: Count the total number of nodes in the list");

printf("\n 13: Traverse the list");

printf("\n 14: EXIT");

printf("\n\n Enter your option : ");

scanf("%d", &option);

switch(option)

{

case 1: start = create\_ll(start);

start = display(start);

break;

case 2: start = insert\_beg(start);

start = display(start);

break;

case 3: start = insert\_end(start);

start = display(start);

break;

case 4: start = insert\_before(start);

start = display(start);

break;

case 5: start = insert\_after(start);

start = display(start);

break;

case 6: start = delete\_beg(start);

start = display(start);

break;

case 7: start = delete\_end(start);

start = display(start);

break;

case 8: start = delete\_node(start);

start = display(start);

break;

case 9: start = delete\_after(start);

start = display(start);

break;

case 10: start = delete\_list(start);

start = display(start); printf("\n LINKED LIST DELETED"); break;

case 11: start = search\_list(start);

break;

case 12: start=count\_node(start);

start = display(start);

break;

case 13: start = display(start);

break;

}

}while(option !=14);

return 0;

}

**//CREATE A LINKED LIST**

struct node \*create\_ll(struct node \*start)

{

struct node \*new\_node, \*ptr;

int num;

printf("\n Enter -1 to end");

printf("\n Enter the data : ");

scanf("%d", &num);

while(num!=-1)

{

new\_node = (struct node\*)malloc(sizeof(struct node));

new\_node -> data=num;

if(start==NULL)

{

new\_node -> next = NULL;

start = new\_node;

}

else

{

ptr=start;

while(ptr->next!=NULL)

ptr=ptr->next;

ptr->next = new\_node;

new\_node->next=NULL;

}

printf("\n Enter the data : ");

scanf("%d", &num);

}

return start;

display(start);

}

**//DISPLAY THE LINKED LIST**

struct node \*display(struct node \*start)

{

struct node \*ptr;

ptr = start;

while(ptr != NULL)

{

printf("\t %d", ptr -> data);

ptr = ptr -> next;

}

return start;

}

**//INSERT IN THE BEGINNING OF LINKED LIST** struct node \*insert\_beg(struct node \*start)

{

struct node \*new\_node;

int num;

printf("\n Enter the data : ");

scanf("%d", &num);

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node -> data = num;

new\_node -> next = start;

start = new\_node;

return start;

}

**//INSERT AT THE END OF LINKED LIST** struct node \*insert\_end(struct node \*start)

{

struct node \*ptr, \*new\_node;

int num;

printf("\n Enter the data : ");

scanf("%d", &num);

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node -> data = num;

new\_node -> next = NULL;

ptr = start;

while(ptr -> next != NULL)

ptr = ptr -> next;

ptr -> next = new\_node;

return start;

}

**//INSERT BEFORE A GIVEN NODE**

struct node \*insert\_before(struct node \*start)

{

struct node \*new\_node, \*ptr, \*preptr; int num, val;

printf("\n Enter the data : ");

scanf("%d", &num);

printf("\n Enter the value before which the data has to be inserted : ");

scanf("%d", &val);

new\_node = (struct node \*)malloc(sizeof(struct node)); new\_node -> data = num;

ptr = start;

while(ptr -> data != val)

{

preptr = ptr;

ptr = ptr -> next;

}

preptr -> next = new\_node;

new\_node -> next = ptr;

return start;

}

**//INSERT AFTER A GIVEN NODE**

struct node \*insert\_after(struct node \*start)

{

struct node \*new\_node, \*ptr, \*preptr; int num, val;

printf("\n Enter the data : ");

scanf("%d", &num);

printf("\n Enter the value after which the data has to be inserted :

");

scanf("%d", &val);

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node -> data = num;

ptr = start;

preptr = ptr;

while(preptr -> data != val)

{

preptr = ptr;

ptr = ptr -> next;

}

preptr -> next=new\_node;

new\_node -> next = ptr;

return start;

}

**//DELETE FROM THE BEGINNING OF LINKED LIST** struct node \*delete\_beg(struct node \*start)

{

struct node \*ptr;

ptr = start;

start = start -> next;

free(ptr);

return start;

}

**//DELETE FROM THE END OF LINKED LIST** struct node \*delete\_end(struct node \*start)

{

struct node \*ptr, \*preptr;

ptr = start;

while(ptr -> next != NULL)

{

preptr = ptr;

ptr = ptr -> next;

}

preptr -> next = NULL;

free(ptr);

return start;

}

**//DELETE A GIVEN NODE**

struct node \*delete\_node(struct node \*start)

{

struct node \*ptr, \*preptr;

int val;

printf("\n Enter the value of the node which has to be deleted :

");

scanf("%d", &val);

ptr = start;

if(ptr -> data == val)

{

start = delete\_beg(start);

return start;

}

else

{

while(ptr -> data != val)

{

preptr = ptr;

ptr = ptr -> next;

}

preptr -> next = ptr -> next;

free(ptr);

return start;

}

}

**//DELETE AFTER A GIVEN NODE**

struct node \*delete\_after(struct node \*start)

{

struct node \*ptr, \*preptr;

int val;

printf("\n Enter the value after which the node has to deleted : "); scanf("%d", &val);

ptr = start;

preptr = ptr;

while(preptr -> data != val)

{

preptr = ptr;

ptr = ptr -> next;

}

preptr -> next=ptr -> next;

free(ptr);

return start;

}

**//DELETE THE ENTIRE LIST**

struct node \*delete\_list(struct node \*start)

{

struct node \*ptr;

if(start!=NULL)

{

ptr=start;

while(ptr != NULL)

{

printf("\n %d is to be deleted next", ptr -> data); start = delete\_beg(ptr);

ptr = start;

}

}

return start;

}

**//COUNT THE NUMBER OF NODES**

struct node \*count\_node(struct node \*start)

{

int count=0;

struct node \*ptr;

if(start!=NULL)

{

ptr=start;

while(ptr!=NULL)

{

count=count+1;

ptr=ptr->next;

}

printf("\n total number of nodes are: %d \n" , count);

}

return start;

}

**//SEARCH THE LINKED LIST**

struct node \*search\_list(struct node \*start)

{

int val;

printf("\n Enter the value of the node which is to be searched :

");

scanf("%d", &val);

struct node \*ptr , \*pos;

ptr=start;

while(ptr!=NULL)

{

if(ptr->data==val)

{

pos=ptr;

printf("value found\n");

}

ptr=ptr->next;

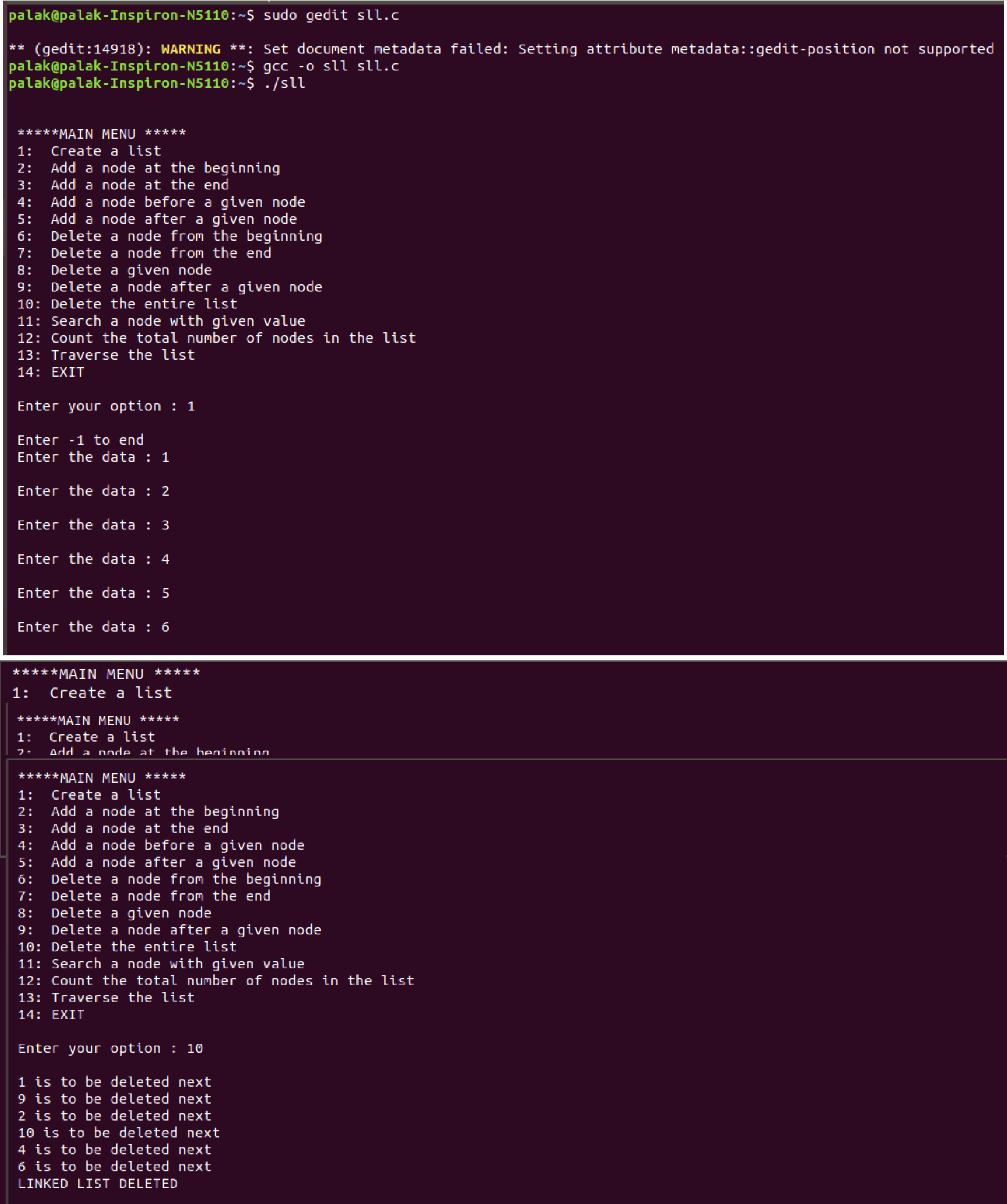
}

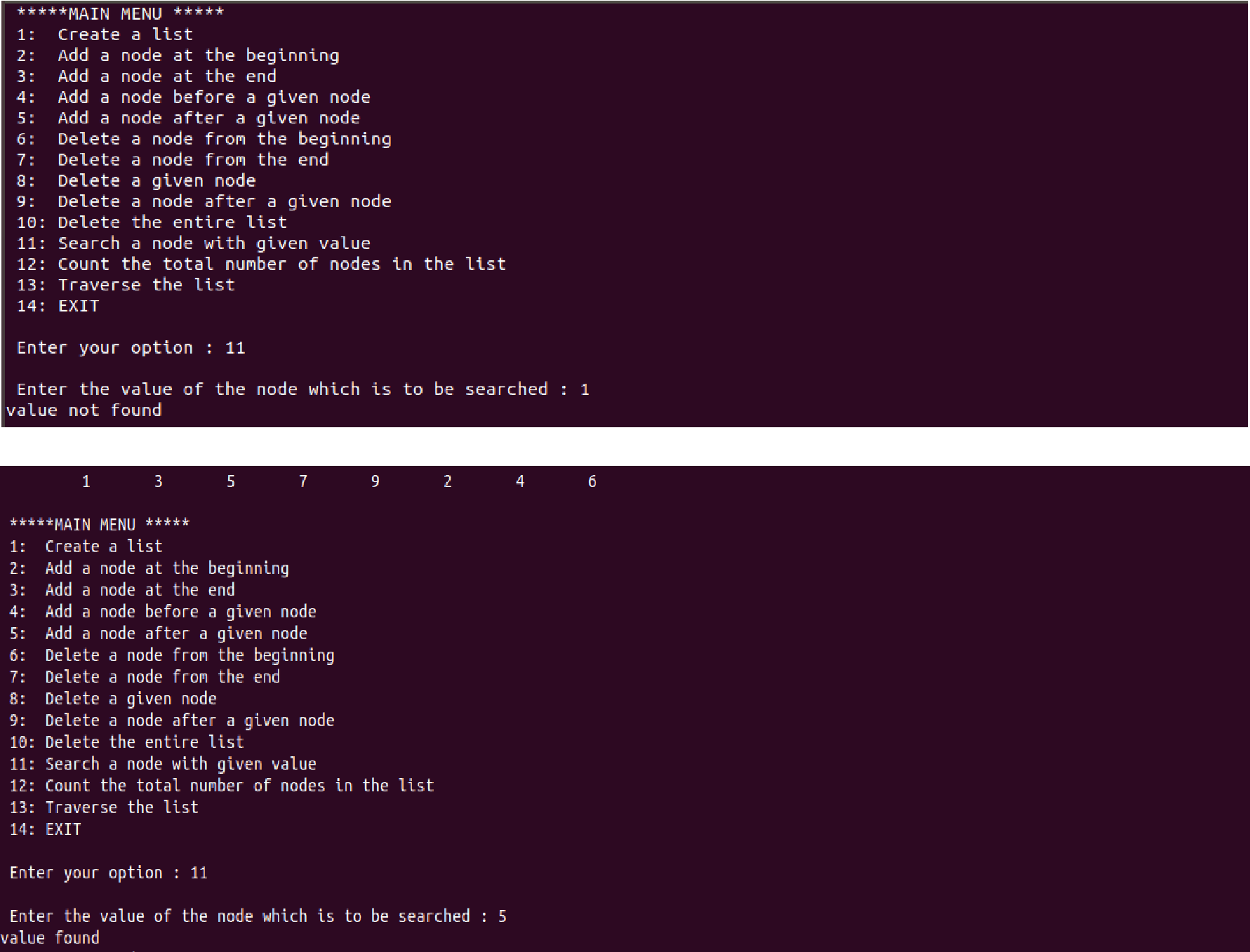
pos=NULL;

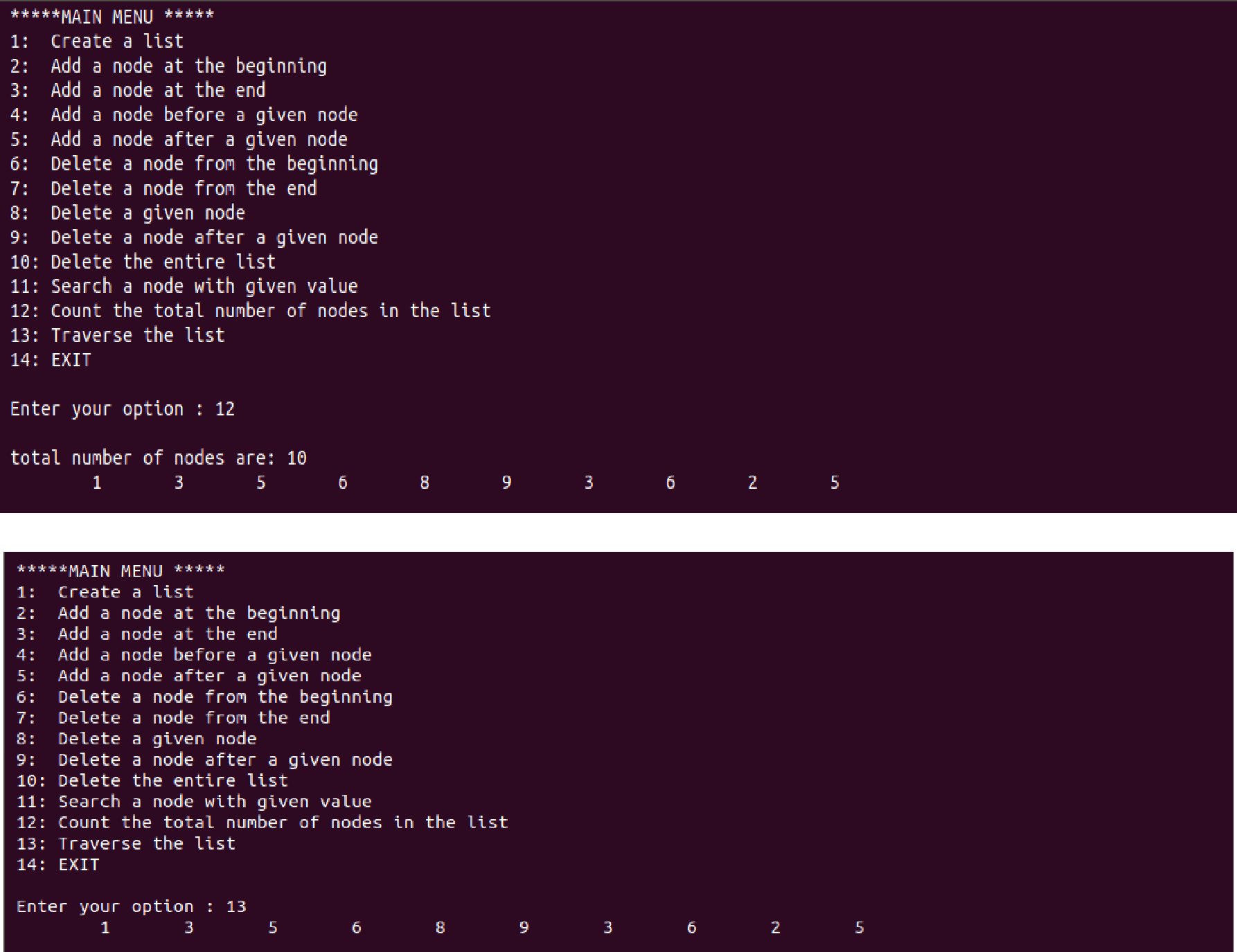
printf("value not found\n");

}

**Output:**







**Question. WAP to perform the following operations on a circular linked list.**

**a.) create a node.**

**b.) insert a node.**

**c.)delete a node.**

**d.)search a node.**

**e.)count the number of nodes.**

**f.)traverse the list.**

#include <stdio.h>

#include <malloc.h>

struct node

{

int data;

struct node \*next;

};

**//FUNCTION PROTOTYPING**

struct node \*start = NULL;

struct node \*create\_cll(struct node \*); struct node \*display(struct node \*); struct node \*insert\_beg(struct node \*); struct node \*insert\_end(struct node \*); struct node \*delete\_beg(struct node \*); struct node \*delete\_end(struct node \*); struct node \*delete\_after(struct node \*); struct node \*delete\_list(struct node \*);

void search();

int count(struct node \*);

struct node \*start , \*x;

int main()

{

int option;

do

{

printf("\n\n \*\*\*\*\*MAIN MENU \*\*\*\*\*");

printf("\n 1: Create a list");

printf("\n 2: Add a node at the beginning");

printf("\n 3: Add a node at the end");

printf("\n 4: Delete a node from the beginning");

printf("\n 5: Delete a node from the end");

printf("\n 6: Delete a node after a given node");

printf("\n 7: Delete the entire list");

printf("\n 8: search the list");

printf("\n 9: count the number of nodes in a linked list");

printf("\n 10: Display the list");

printf("\n 11: EXIT");

printf("\n\n Enter your option : ");

scanf("%d", &option);

switch(option)

{

case 1:

start = create\_cll(start);

start = display(start);

break;

case 2:

start = insert\_beg(start);

start = display(start);

break;

case 3:

start = insert\_end(start);

start = display(start);

break;

case 4:

start = delete\_beg(start);

start = display(start);

break;

case 5:

start = delete\_end(start);

start = display(start);

break;

case 6:

start = delete\_after(start);

start = display(start);

break;

case 7:

start = delete\_list(start);

printf("\n CIRCULAR LINKED LIST DELETED");

break;

case 8:

search(start);

break;

case 9:printf("%d",count(start));

break;

case 10:

start = display(start);

break;

}

}while(option !=11);

return 0;

}

**//CREATE THE CIRCULAR LINKED LIST** struct node \*create\_cll(struct node \*start)

{

struct node \*new\_node, \*ptr;

int num;

printf("\n Enter –1 to end");

printf("\n Enter the data : ");

scanf("%d", &num);

while(num!=-1)

{

new\_node = (struct node\*)malloc(sizeof(struct node));

new\_node->data = num;

if(start == NULL)

{

new\_node->next = new\_node;

start = new\_node;

}

else

{

ptr = start;

while(ptr-> next != start)

ptr=ptr->next;

ptr->next = new\_node;

new\_node->next = start;

}

printf("\n Enter the data : ");

scanf("%d", &num);

}

return start;

}

**//TRAVERSE THE CIRCULAR LINKED LIST** struct node \*display(struct node \*start)

{

struct node \*ptr;

ptr=start;

while(ptr-> next != start)

{

printf("\t %d", ptr->data);

ptr = ptr-> next;

}

printf("\t %d", ptr-> data);

return start;

}

**//INSERT AT THE BEGINNING OF LINKED LIST** struct node \*insert\_beg(struct node \*start)

{

struct node \*new\_node, \*ptr;

int num;

printf("\n Enter the data : ");

scanf("%d", &num);

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data = num;

ptr = start;

while(ptr-> next != start)

ptr = ptr-> next;

ptr-> next = new\_node;

new\_node-> next = start;

start = new\_node;

return start;

}

**//INSERT AT THE END OF LINKED LIST** struct node \*insert\_end(struct node \*start)

{

struct node \*ptr, \*new\_node;

int num;

printf("\n Enter the data : ");

scanf("%d", &num);

new\_node = (struct node \*)malloc(sizeof(struct node));

new\_node->data = num;

ptr = start;

while(ptr-> next != start)

ptr = ptr-> next;

ptr-> next = new\_node;

new\_node-> next = start;

return start;

}

**//DELETE FROM THE BEGINNING OF THE LINKED LIST** struct node \*delete\_beg(struct node \*start)

{

struct node \*ptr;

ptr = start;

while(ptr->next != start)

ptr = ptr-> next;

ptr-> next = start->next;

free(start);

start = ptr-> next;

return start;

}

**//DELETE FROM THE END OF THE LINKED LIST** struct node \*delete\_end(struct node \*start)

{

struct node \*ptr, \*preptr;

ptr = start;

while(ptr-> next != start)

{

preptr = ptr;

ptr = ptr-> next;

}

preptr->next = ptr-> next;

free(ptr);

return start;

}

**//DELETE AFTER A GIVEN NODE FROM THE LINKED LIST** struct node \*delete\_after(struct node \*start)

{

struct node \*ptr, \*preptr;

int val;

printf("\n Enter the value after which the node has to deleted : "); scanf("%d", &val);

ptr = start;

preptr = ptr;

while(preptr-> data != val)

{

preptr = ptr;

ptr = ptr-> next;

}

preptr-> next = ptr-> next;

if(ptr == start)

start = preptr-> next;

free(ptr);

return start;

}

**//DELETE THE ENTIRE LINKED LIST** struct node \*delete\_list(struct node \*start)

{

struct node \*ptr;

ptr = start;

while(ptr-> next != start)

start = delete\_end(start);

free(start);

return start;

}

**//SEARCH A GIVEN NODE IN THE LINKED LIST** void search()

{

int val, count=0 , flag=0;

printf("\nenter the element to search\n"); scanf("%d", &val);

if(start ==NULL)

printf("\n list is empty nothing to search");

else

{

x=start;

while(x->next!=start)

{

if(x->data==val)

{

printf("\nthe element is found at %d",count);

flag=1;

break;

}

count++;

x=x->next;

}

if (x->data==val)

{

printf("element found at position %d", count);

}

if(flag==0)

{

printf("\n element not found");

}

}

}

**//COUNT THE NUMBER OF NODES IN THE LINKED LIST** int count(struct node \*start)

{

struct node \*ptr;

ptr=start;

int result=0;

if(ptr!=NULL)

{

do

{

ptr=ptr->next;

result++;

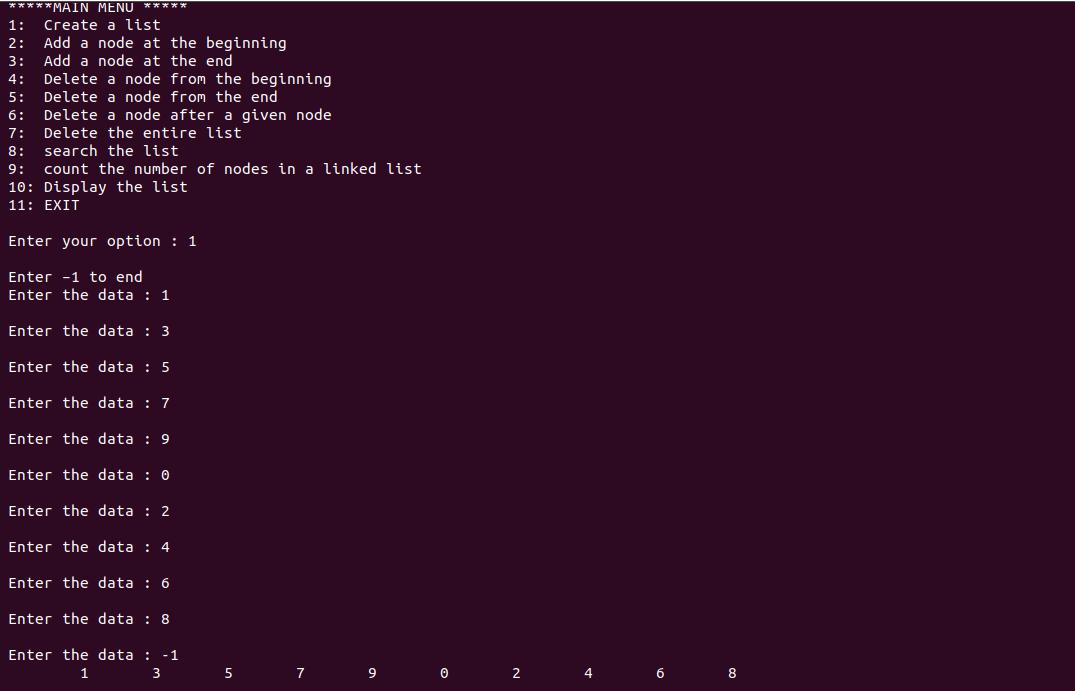
}while(ptr!=start);

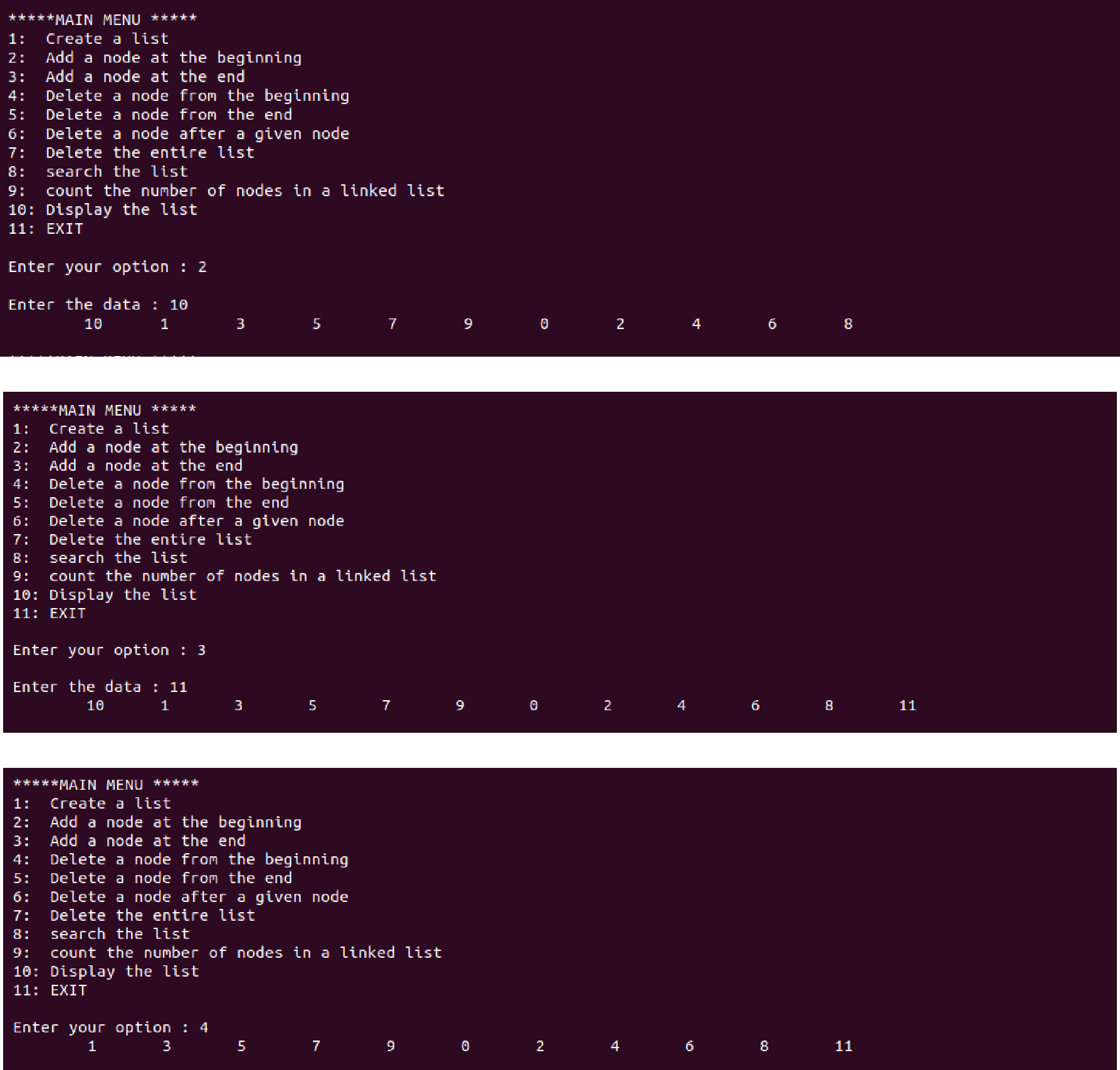
}

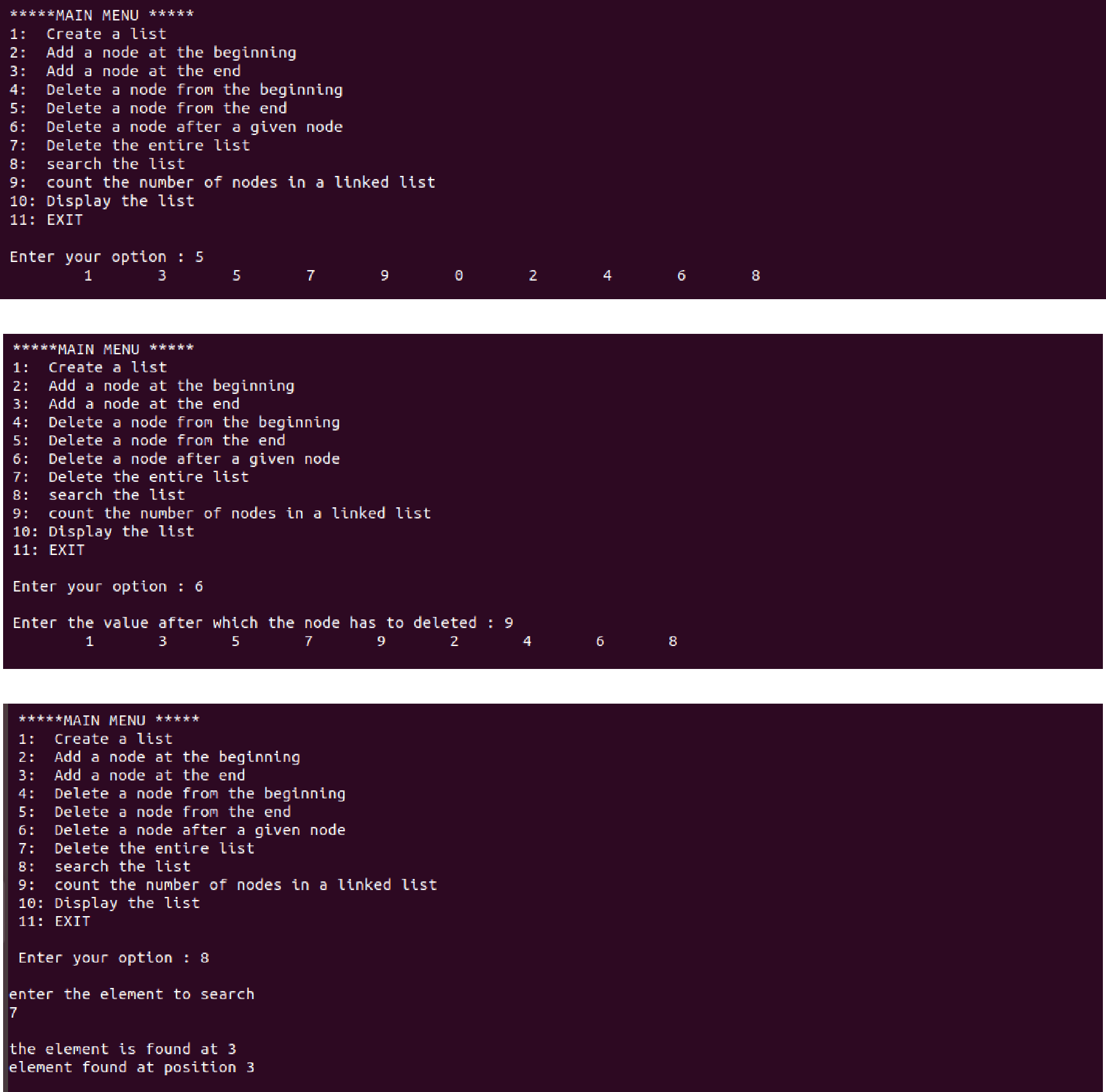
return result;

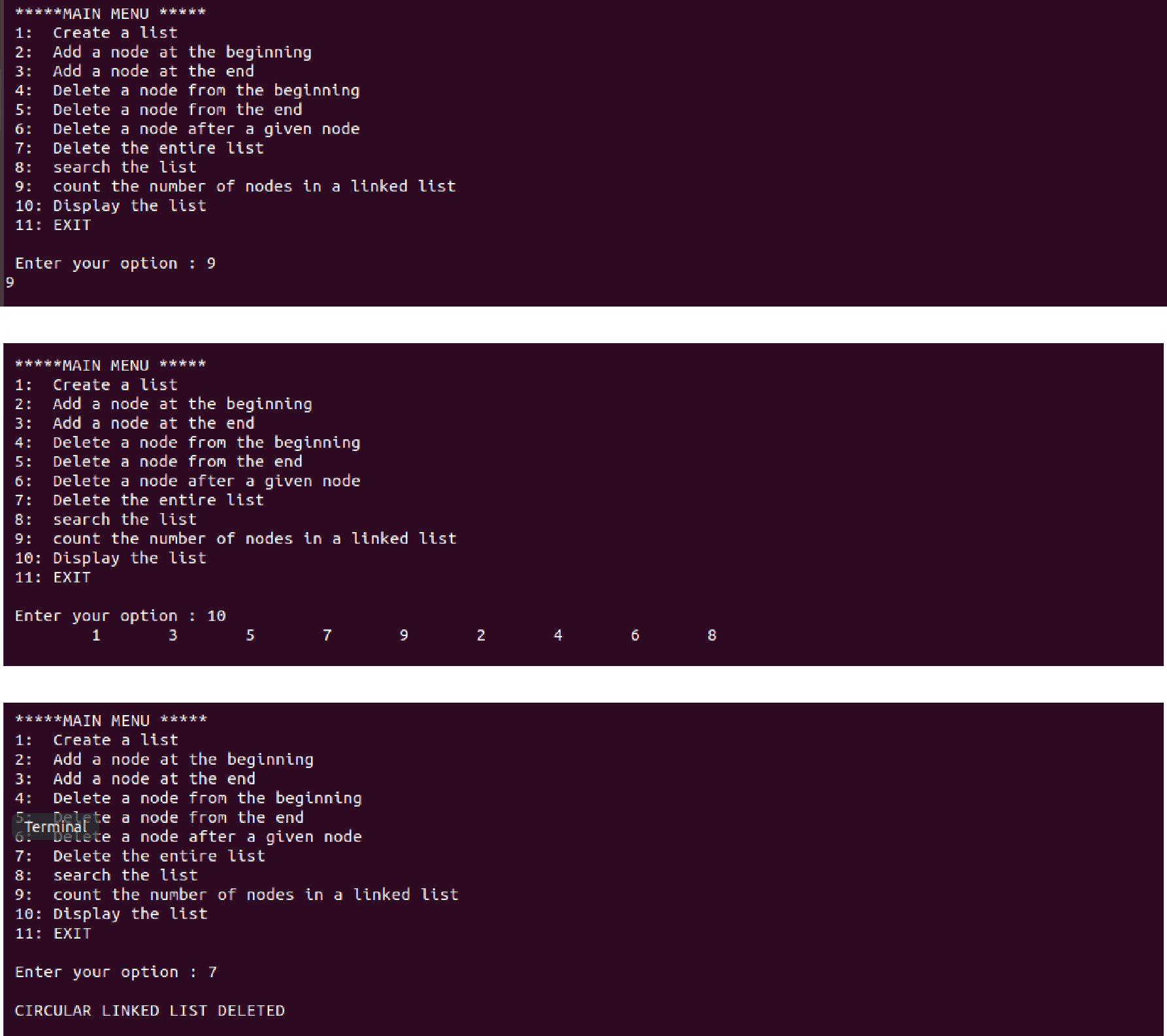
}

**Output:**









**Question :IMPLEMENT THE FOLLOWING ON DOUBLY LINKED LIST:**

1. **CREATE A NODE OF THE LINKED LIST**
2. **TRAVERSE THE LINKED LIST**
3. **INSERT A NODE IN THE FRONT,AT THE END, & AT ANY OTHER POSITION**
4. **DELETE THE VERY FIRST NODE,AT THE END, & AT ANY OTHER POSITION**
5. **COUNTS THE NUMBER OF NODES OF LINKED LIST**
6. **SEARCH THE NODE WITH THE GIVEN VALUE IN THE LIST**

#include<conio.h>

#include<stdio.h>

#include<malloc.h>

struct node

{

int data;

struct node \*next;

struct node \*prev;

};

struct node \*start=NULL;

struct node \*create(struct node \*);

struct node \*display(struct node \*);

struct node \*insert\_beg(struct node \*);

struct node \*insert\_after(struct node \*);

struct node \*insert\_end(struct node \*);

struct node \*delete\_beg(struct node \*);

struct node \*delete\_after(struct node \*);

struct node \*delete\_end(struct node \*);

int main()

{

int option;

clrscr();

do

{

printf("\n\*\*\*main menu\*\*\*");

printf("\n 1:create list");

printf("\n 2:display list");

printf("\n 3:add a node at beginning");

printf("\n 4:add a node at end");

printf("\n 5:add a node after a given node");

printf("\n 6:delete a node at beginning");

printf("\n 7:delete a node at end");

printf("\n 8:delete a node after a given node");

printf("\n 9:exit");

printf("\n enter your option :");

scanf("%d",&option);

switch(option)

{

case 1:start=create(start);

printf("\n circular linked list created");

break;

case 2:start=display(start);

break;

case 3:start=insert\_beg(start);

break;

case 4:start=insert\_end(start);

break;

case 5:start=insert\_after(start);

break;

case 6:start=delete\_beg(start);

break;

case 7:start=delete\_end(start);

break;

case 8:start=delete\_after(start);

break;

}

}

while(option!=9);

getch();

return 0;

}

struct node \*create(struct node \*start)

{

struct node \*new\_node, \*ptr;

int num;

printf("\n enter -1 to end");

printf("\n enter the data");

scanf("%d",&num);

while(num!=-1)

{

new\_node=(struct node \*)malloc(sizeof(struct node));

new\_node->data=num;

if(start==NULL)

{

new\_node->prev=NULL;

new\_node->next=NULL;

start=new\_node;

}

else

{

ptr=start;

while(ptr->next!=NULL)

ptr=ptr->next;

ptr->next=new\_node;

new\_node->prev=ptr;

new\_node->next=NULL;

}

printf("\n enter the data:");

scanf("%d",&num);

}

return start;

}

struct node \*display(struct node \*start)

{

struct node \*ptr;

ptr=start;

while(ptr->next!=NULL)

{

printf("\t %d",ptr->data);

ptr=ptr->next;

}

printf("\t %d",ptr->data);

return start;

}

struct node \*insert\_beg(struct node \*start)

{

struct node \*new\_node;

int num;

printf("\n enter the data");

scanf("%d",&num);

new\_node=(struct node \*)malloc(sizeof(struct node));

new\_node->data=num;

start->prev=new\_node;

new\_node->next=start;

new\_node->prev=NULL;

start=new\_node;

return start;

}

 struct node \*insert\_end(struct node \*start)

{

struct node \*new\_node, \*ptr;

int num;

printf("\n enter the data");

scanf("%d",&num);

new\_node=(struct node \*)malloc(sizeof(struct node));

new\_node->data=num;

ptr=start;

while(ptr->next!=NULL)

ptr=ptr->next;

ptr->next=new\_node;

new\_node->prev=ptr;

new\_node->next=NULL;

return start;

}

struct node \*delete\_beg(struct node \*start)

{

struct node \*ptr;

ptr=start;

start=start->next;

start->prev=NULL;

free(ptr);

return start;

}

 struct node \*delete\_end(struct node \*start)

{

struct node \*ptr;

ptr=start;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->prev->next=NULL;

free(ptr);

return start;

}

struct node \*delete\_after(struct node \*start)

{

struct node \*ptr, \*temp;

int val;

printf("\n enter the value after which a node has to be deleted");

scanf("%d",&val);

ptr=start;

while(ptr->data!=val)

{

ptr=ptr->next;

}

temp=ptr->next;

ptr->next=temp->next;

temp->next->prev=ptr;

free(temp);

return start;

}

 struct node \*insert\_after(struct node \*start)

{

struct node \*ptr, \*new\_node;

int val,num;

printf("\n enter the value after which a node has to be inserted");

scanf("%d",&val);

printf("\n enter the data ");

scanf("%d",&num);

new\_node=(struct node \*)malloc(sizeof(struct node));

new\_node->data=num;

ptr=start;

while(ptr->data!=val)

{

ptr=ptr->next;

}

new\_node->prev=ptr;

new\_node->next=ptr->next;

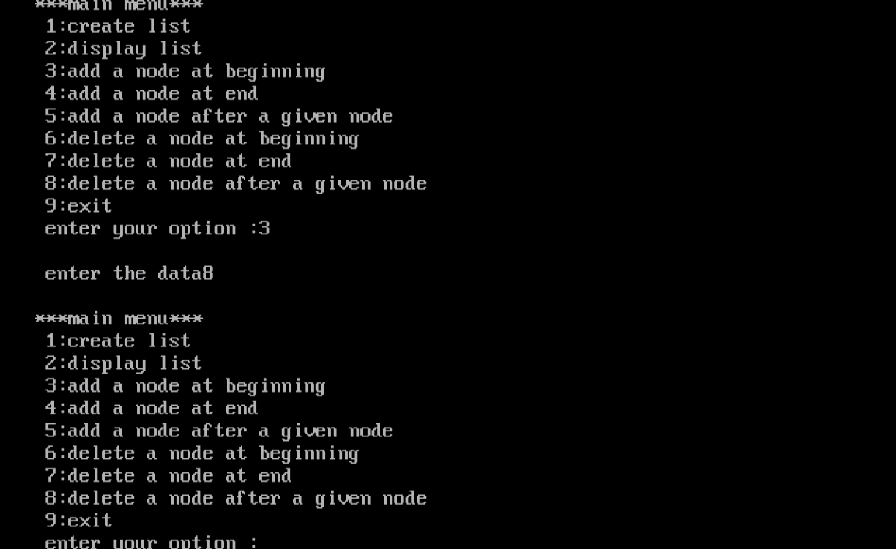
ptr->next->prev=new\_node;

ptr->next=new\_node;

return start;

}

Output:



**Question. WAP to implement all primitive operations of a stack using array.**

#include <stdio.h>

#include <stdlib.h>

#define MAX 5 // Altering this value changes size of stack created

int st[MAX], top=-1;

void push(int st[], int val);

int pop(int st[]);

int peek(int st[]);

void display(int st[]);

int main(int argc, char \*argv[])

{

int val, option;

do

{

printf("\n \*\*\*\*\*MAIN MENU\*\*\*\*\*");

printf("\n 1. PUSH");

printf("\n 2. POP");

printf("\n 3. PEEK");

printf("\n 4. DISPLAY");

printf("\n 5. EXIT");

printf("\n Enter your option: ");

scanf("%d", &option);

switch(option)

{

case 1:

printf("\n Enter the number to be pushed on

stack: ");

scanf("%d", &val);

push(st, val);

break;

case 2:

val = pop(st);

if(val != -1)

printf("\n The value deleted from stack is: %d",

val);

break;

case 3:

val = peek(st);

if(val != -1)

printf("\n The value stored at top of stack is:

%d", val);

break;

case 4:

display(st);

break;

}

}while(option != 5);

return 0;

}

**//PUSH VALUE INTO THE STACK** void push(int st[], int val)

{

if(top == MAX-1)

{

("\n STACK OVERFLOW");

}

else

{

top++;

st[top] = val;

}

}

//**POP VALUE OUT OF THE STACK** int pop(int st[])

{

int val;

if(top == -1)

{

printf("\n STACK UNDERFLOW");

return -1;

}

else

{

val = st[top];

top--;

return val;

}

}

**//DISPLAY STACK**

void display(int st[])

{

int i;

if(top == -1)

printf("\n STACK IS EMPTY");

else

{

for(i=top;i>=0;i--)

printf("\n %d",st[i]);

printf("\n"); // Added for formatting purposes

}

}

**//PEEK THE VALUE OF STACK** int peek(int st[])

{

if(top == -1)

{

printf("\n STACK IS EMPTY");

return -1;

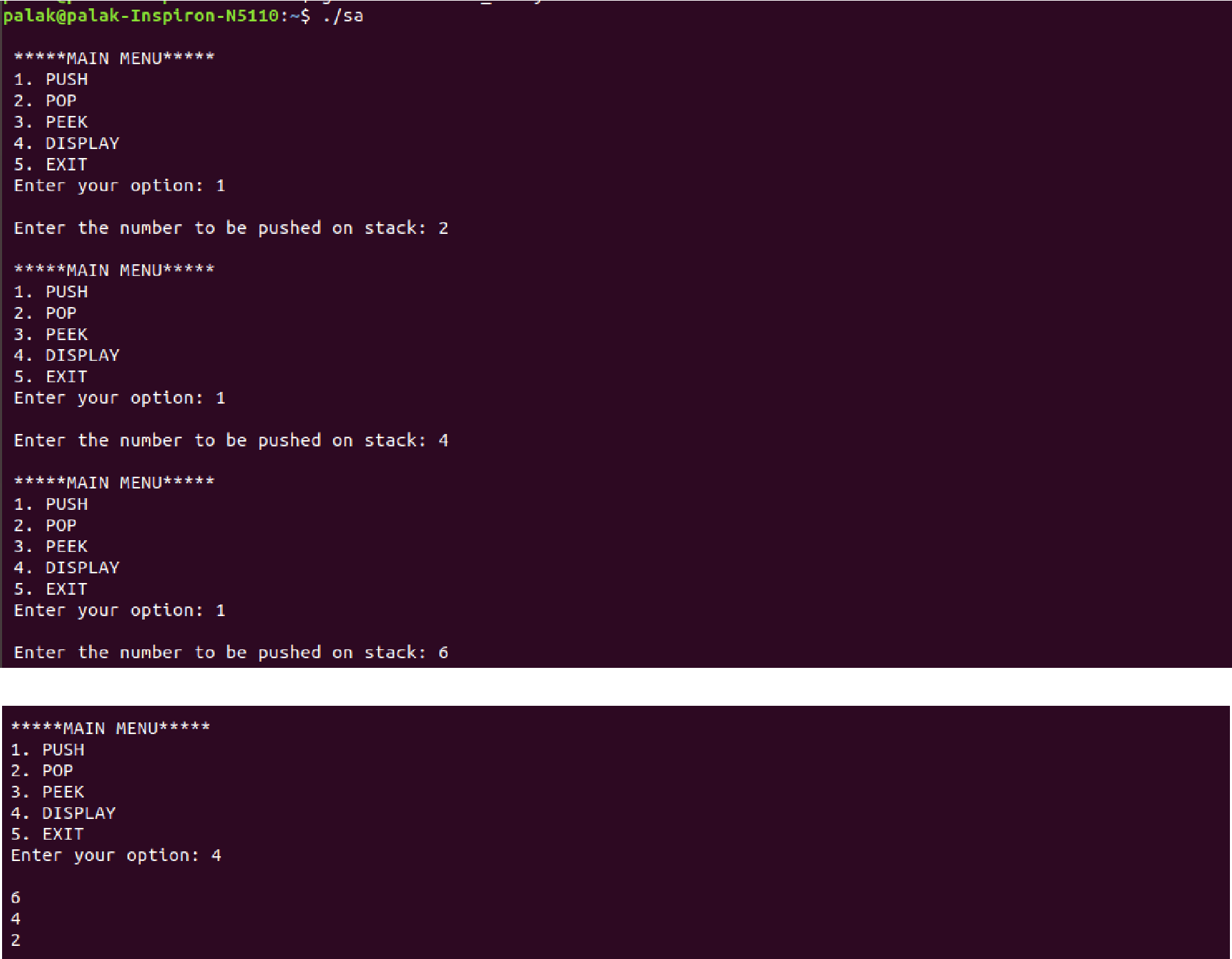
}

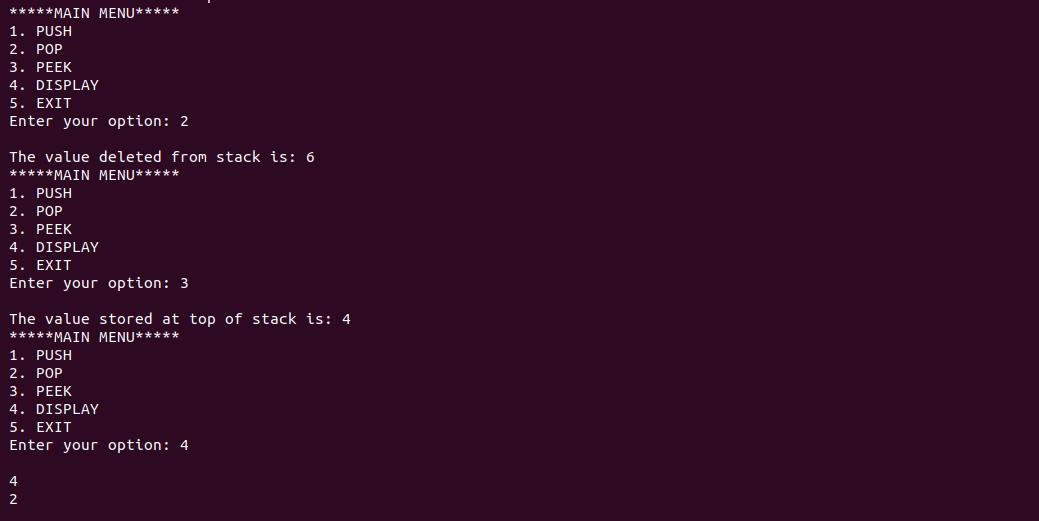
else

return (st[top]);

}

**Output:**





**Question. WAP to implement all primitive operations of a stack using linked list.**

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

struct stack

{

int data;

struct stack \*next;

};

struct stack \*top = NULL;

struct stack \*push(struct stack \*, int); struct stack \*display(struct stack \*); struct stack \*pop(struct stack \*); int peek(struct stack \*);

int main(int argc, char \*argv[])

{

int val, option;

do

{

printf("\n \*\*\*\*\*MAIN MENU\*\*\*\*\*");

printf("\n 1. PUSH");

printf("\n 2. POP");

printf("\n 3. PEEK");

printf("\n 4. DISPLAY");

printf("\n 5. EXIT");

printf("\n Enter your option: ");

scanf("%d",&option);

switch(option)

{

case 1:

printf("\n Enter the number to be pushed on stack: "); scanf("%d",&val);

top = push(top, val);

break;

case 2:

top = pop(top);

break;

case 3:

val=peek(top);

if (val != -1)

printf("\n The value at the top of stack is: %d",

val);

else

printf("\n STACK IS EMPTY");

break;

case 4:

top = display(top);

break;

}

}while(option != 5);

return 0;

}

**//PUSH INTO THE STACK**

struct stack \*push(struct stack \*top, int val)

{

struct stack \*ptr;

ptr = (struct stack\*)malloc(sizeof(struct stack));

ptr -> data = val;

if(top == NULL)

{

ptr -> next = NULL;

top = ptr;

}

else

{

ptr -> next = top;

top = ptr;

}

return top;

}

**//DISPLAY THE STACK**

struct stack \*display(struct stack \*top)

{

struct stack \*ptr;

ptr = top;

if(top == NULL)

printf("\n STACK IS EMPTY");

else

{

while(ptr != NULL)

{

printf("\n %d", ptr -> data);

ptr = ptr -> next;

}

}

return top;

}

**//POP OUT OF THE STACK**

struct stack \*pop(struct stack \*top)

{

struct stack \*ptr;

ptr = top;

if(top == NULL)

printf("\n STACK UNDERFLOW");

else

{

top = top -> next;

printf("\n The value being deleted is: %d", ptr -> data);

free(ptr);

}

return top;

}

**//PEEK THE VALUE OF THE STACK** int peek(struct stack \*top)

{

if(top==NULL)

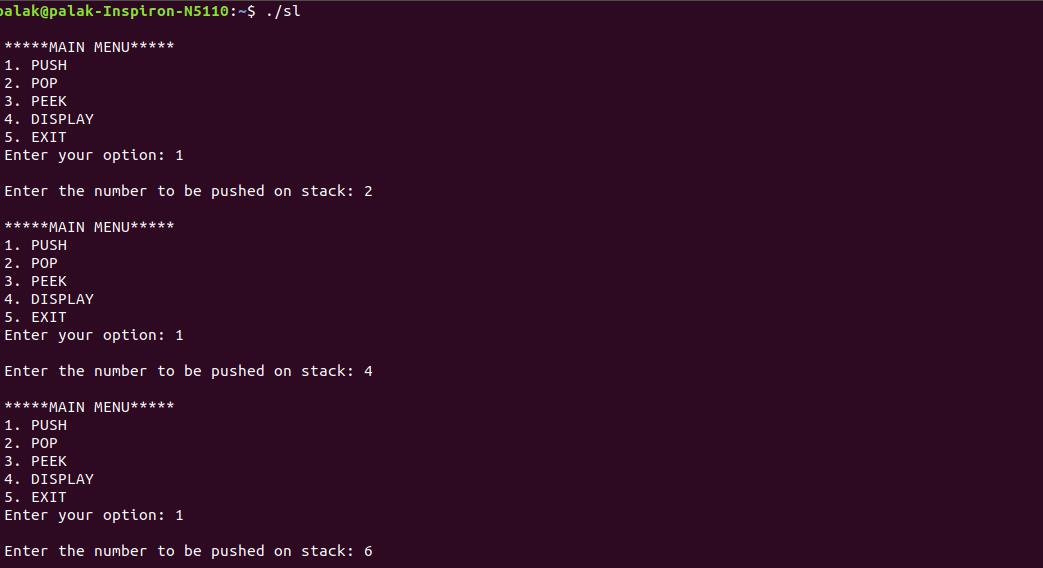
return -1;

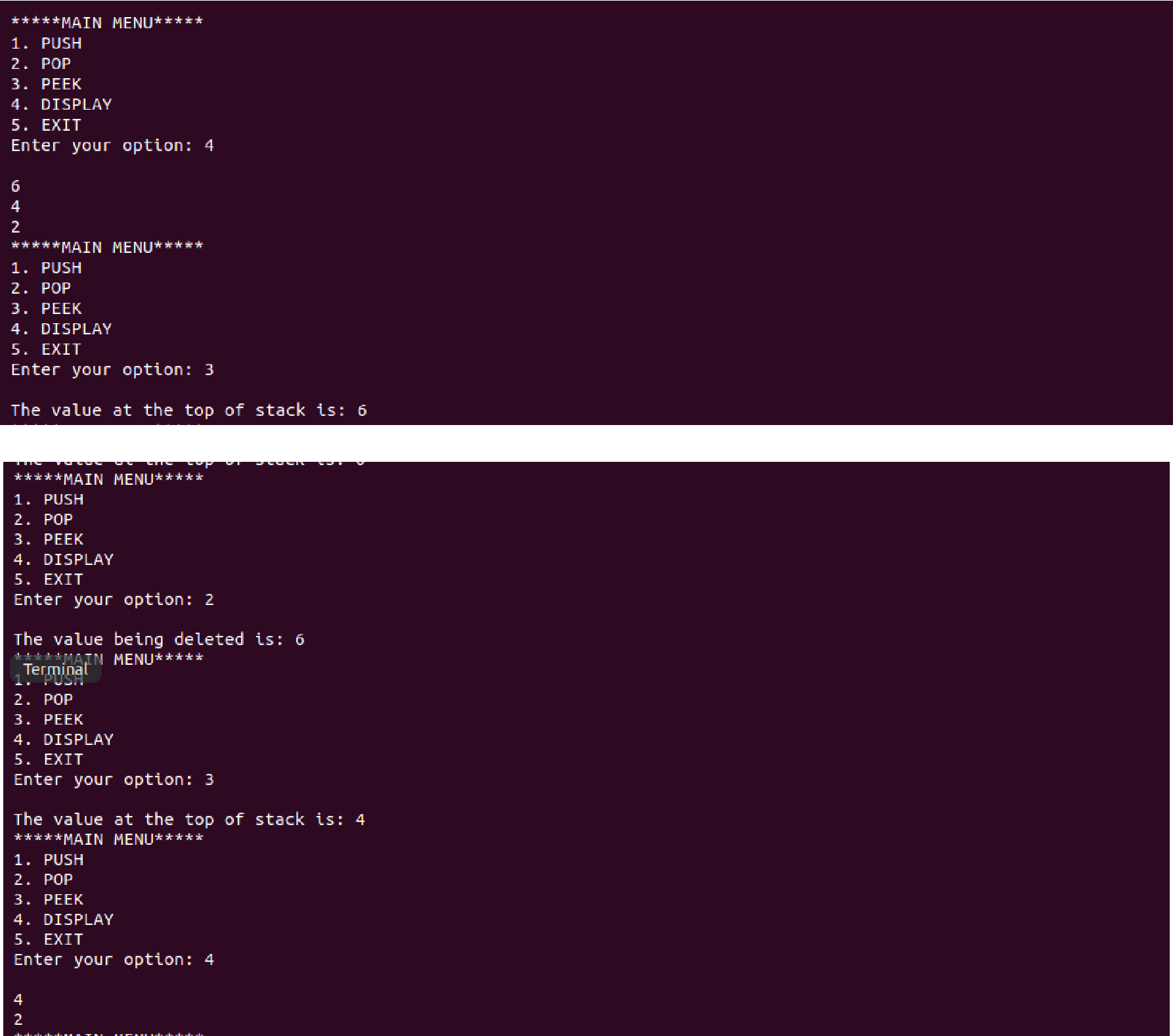
else

return top ->data;

}

**Output:**





**Question. WAP to implement all primitive operations of a queue using an array.**

#include <stdio.h>

#define MAX 10 // Changing this value will change length of array int queue[MAX];

int front = -1, rear = -1;

void insert(void);

int delete\_element(void);

int peek(void);

void display(void);

int main()

{

int option, val;

do

{

printf("\n\n \*\*\*\*\* MAIN MENU \*\*\*\*\*");

printf("\n 1. Insert an element");

printf("\n 2. Delete an element");

printf("\n 3. Peek");

printf("\n 4. Display the queue");

printf("\n 5. EXIT");

printf("\n Enter your option : ");

scanf("%d", &option);

switch(option)

{

case 1:

insert();

break;

case 2:

val = delete\_element();

if (val != -1)

printf("\n The number deleted is : %d", val);

break;

case 3:

val = peek();

if (val != -1)

printf("\n The first value in queue is : %d", val);

break;

case 4:

display();

break;

}

}while(option != 5);

return 0;

}

**//INSERT AN ELEMENT**

void insert()

{

int num;

printf("\n Enter the number to be inserted in the queue : ");

scanf("%d", &num);

if(rear == MAX-1)

printf("\n OVERFLOW");

else if(front == -1 && rear == -1)

front = rear = 0;

else

rear++;

queue[rear] = num;

}

**//DELETE AN ELEMENT**

int delete\_element()

{

int val;

if(front == -1 || front>rear)

{

printf("\n UNDERFLOW");

return -1;

}

else

{

val = queue[front];

front++;

if(front > rear)

front = rear = -1;

return val;

}

}

**//PEEK THE VALUE**

int peek()

{

if(front==-1 || front>rear)

{

printf("\n QUEUE IS EMPTY");

return -1;

}

else

{

return queue[front];

}

}

**//DISPLAY THE QUEUE**

void display()

{

int i;

printf("\n");

if(front == -1 || front > rear)

printf("\n QUEUE IS EMPTY");

else

{

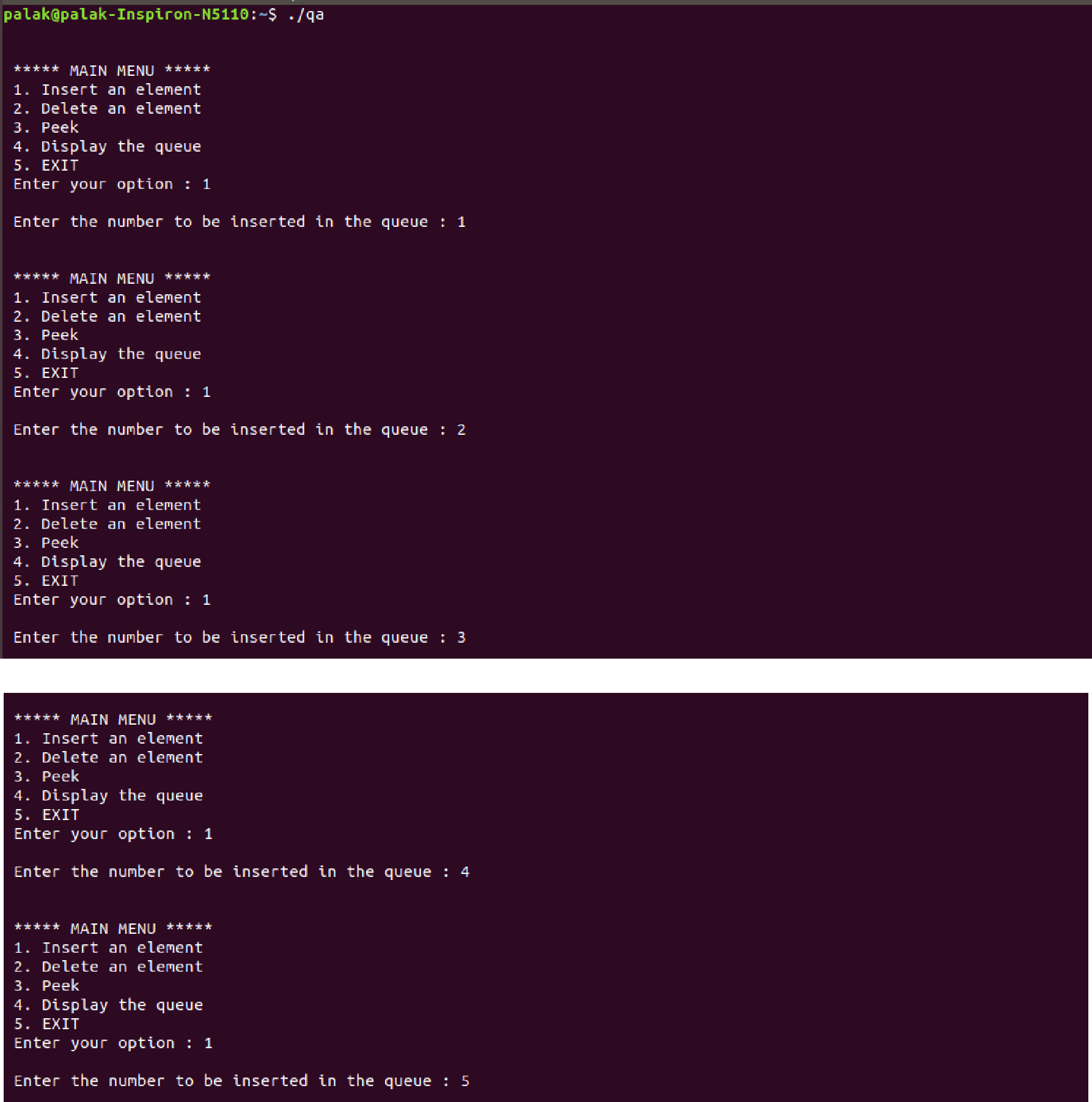
for(i = front;i <= rear;i++)

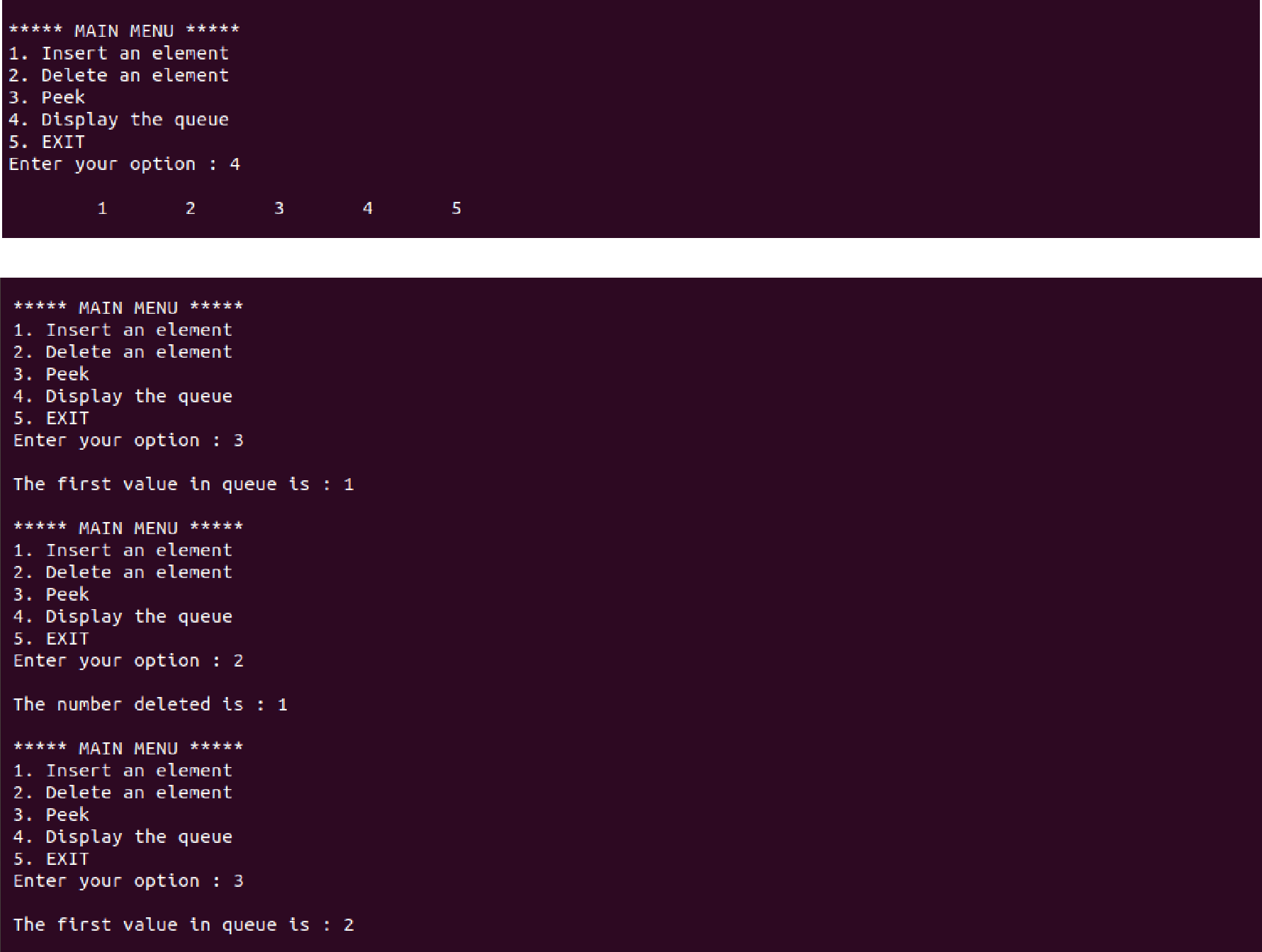
printf("\t %d", queue[i]);

}

}

**Output:**





**Question. WAP to implement all primitive operations of a queue using a linked list.**

#include<stdio.h>

#include<stdlib.h>

struct Node

{

int Data;

struct Node\* next;

}\*rear, \*front;

void delQueue()

{

struct Node \*temp, \*var=rear;

if(var==rear)

{

rear = rear->next;

free(var);

}

else

printf("\nQueue Empty");

}

void push(int value)

{

struct Node \*temp;

temp=(struct Node \*)malloc(sizeof(struct Node));

temp->Data=value;

if (front == NULL)

{

front=temp;

front->next=NULL;

rear=front;

}

else

{

front->next=temp;

front=temp;

front->next=NULL;

}

}

void display()

{

struct Node \*var=rear;

if(var!=NULL)

{

printf("\nElements are as: ");

while(var!=NULL)

{

printf("\t%d",var->Data);

var=var->next;

}

printf("\n");

}

else

printf("\nQueue is Empty");

}

int main()

{

int i=0;

front=NULL;

printf(" \n1. Push to Queue"); printf(" \n2. Pop from Queue"); printf(" \n3. Display Data of Queue"); printf(" \n4. Exit\n"); while(1)

{

printf(" \nChoose Option: ");

scanf("%d",&i);

switch(i)

{

case 1:

{

int value;

printf("\nEnter a value to push into Queue: "); scanf("%d",&value);

push(value);

display();

break;

}

case 2:

{

delQueue();

display();

break;

}

case 3:

{

display();

break;

}

case 4:

{

exit(0);

}

default:

{

printf("\nwrong choice for operation");

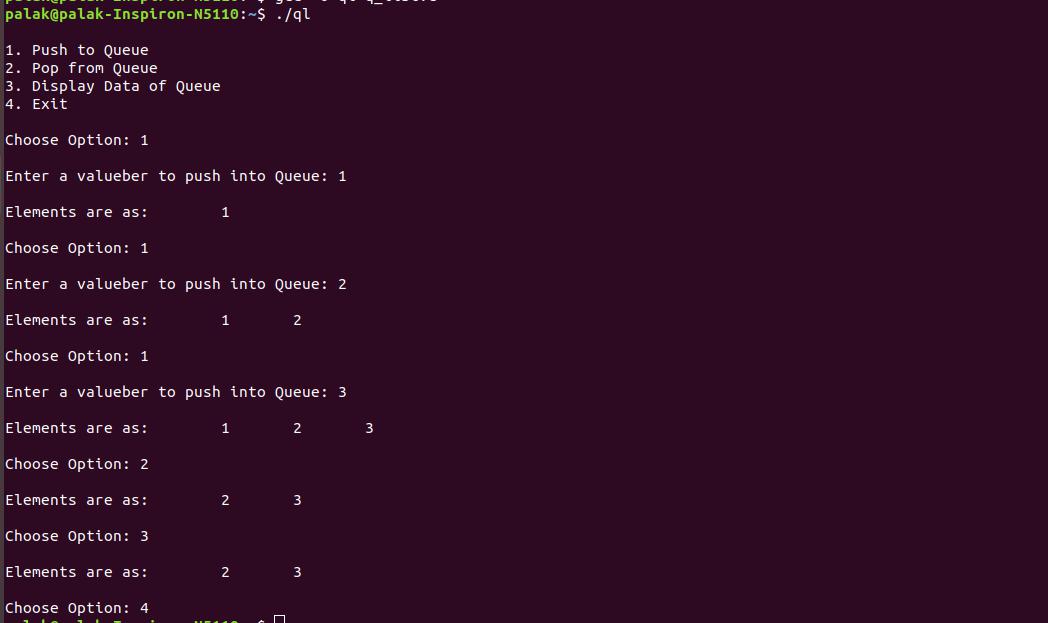
}

}

}

}

**Output:**



**Question. WAP to implement all primitive operations of a circular queue using an array.**

#include <stdio.h>

#define size 5

void insertq(int[], int);

void deleteq(int[]);

void display(int[]);

int front = - 1;

int rear = - 1;

int main()

{

int n, ch;

int queue[size];

do

{

printf("\n\n Circular Queue:\n1. Insert \n2. Delete\n3. Display\n0. Exit");

printf("\nEnter Choice 0-3? : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

printf("\nEnter number: ");

scanf("%d", &n);

insertq(queue, n);

break;

case 2:

deleteq(queue);

break;

case 3:

display(queue);

break;

}

}while (ch != 0);

}

**//INSERT ELEMENT IN QUEUE** void insertq(int queue[], int item)

{

if ((front == 0 && rear == size - 1) || (front == rear + 1))

{

printf("queue is full");

return;

}

else if (rear == - 1)

{

rear++;

front++;

}

else if (rear == size - 1 && front > 0)

{

rear = 0;

}

else

{

rear++;

}

queue[rear] = item;

}

**//DISPLAY THE QUEUE**

void display(int queue[])

{

int i;

printf("\n");

if (front > rear)

{

for (i = front; i < size; i++)

{

printf("%d ", queue[i]);

}

for (i = 0; i <= rear; i++)

printf("%d ", queue[i]);

}

else

{

for (i = front; i <= rear; i++)

printf("%d ", queue[i]);

}

}

**//DELETE ELEMENT FROM QUEUE** void deleteq(int queue[])

{

if (front == - 1)

{

printf("Queue is empty ");

}

else if (front == rear)

{

printf("\n %d deleted", queue[front]); front = - 1;

rear = - 1;

}

else

{

printf("\n %d deleted", queue[front]);

front++;

}

}

**Output:**





**Question. WAP to perform the following operations in a Binary Search Tree:**

**a.)Traverse the tree in Pre , Post , and Inorder.**

**b.)Insert a node in the given BST.**

**c.)Delete a node from the given BST.**

**d.)Search a given node in the BST.**

#include <stdio.h>

#include <malloc.h>

struct node

{

int data;

struct node \*left;

struct node \*right;

};

struct node \*tree;

void create\_tree(struct node \*);

struct node \*insertElement(struct node \*, int); void preorderTraversal(struct node \*); void inorderTraversal(struct node \*);

void postorderTraversal(struct node \*);

struct node \*deleteElement(struct node \*, int);

struct node \*deleteTree(struct node \*);

struct node \*search(struct node \*, int);

int main()

{

int option, val;

struct node \*ptr;

create\_tree(tree);

do

{

printf("\n \*\*\*\*\*\*MAIN MENU\*\*\*\*\*\*\* \n"); printf("\n 1. Insert Element"); printf("\n 2. Preorder Traversal");

printf("\n 3. Inorder Traversal");

printf("\n 4. Postorder Traversal");

printf("\n 5. Delete an element");

printf("\n 6. Delete the tree");

printf("\n 7. Search a node");

printf("\n 8. Exit");

printf("\n\n Enter your option : ");

scanf("%d", &option);

switch(option)

{

case 1:

printf("\n Enter the value of the new node : "); scanf("%d", &val);

tree = insertElement(tree, val);

break;

case 2:

printf("\n The elements of the tree are : \n"); preorderTraversal(tree);

break;

case 3:

printf("\n The elements of the tree are : \n"); inorderTraversal(tree);

break;

case 4:

printf("\n The elements of the tree are : \n"); postorderTraversal(tree);

break;

case 5:

printf("\n Enter the element to be deleted : "); scanf("%d", &val);

tree = deleteElement(tree, val);

break;

case 6:

tree = deleteTree(tree);

break;

case 7:

printf("\n Enter the element to be searched : "); scanf("%d",&val);

tree=search(tree,val);

if (search(tree,val))

printf("value found\n");

else

printf("value not found\n");

}

}while(option!=8);

return 0;

}

**//CREATE A TREE**

void create\_tree(struct node \*tree)

{

tree = NULL;

}

**//INSERT AN ELEMENT IN THE TREE**

struct node \*insertElement(struct node \*tree, int val)

{

struct node \*ptr, \*nodeptr, \*parentptr;

ptr = (struct node\*)malloc(sizeof(struct node));

ptr->data = val;

ptr->left = NULL;

ptr->right = NULL;

if(tree==NULL)

{

tree=ptr;

tree->left=NULL;

tree->right=NULL;

}

else

{

parentptr=NULL;

nodeptr=tree;

while(nodeptr!=NULL)

{

parentptr=nodeptr;

if(val<nodeptr->data)

nodeptr=nodeptr->left;

else

nodeptr = nodeptr->right;

}

if(val<parentptr->data)

parentptr->left = ptr;

else

parentptr->right = ptr;

}

return tree;

}

**//PREORDER TRAVERSAL**

void preorderTraversal(struct node \*tree)

{

if(tree != NULL)

{

printf("%d\t", tree->data);

preorderTraversal(tree->left);

preorderTraversal(tree->right);

}

}

**//INORDER TRAVERSAL**

void inorderTraversal(struct node \*tree)

{

if(tree != NULL)

{

inorderTraversal(tree->left);

printf("%d\t", tree->data);

inorderTraversal(tree->right);

}

}

**//POSTORDER TRAVERSAL**

void postorderTraversal(struct node \*tree)

{

if(tree != NULL)

{

postorderTraversal(tree->left);

postorderTraversal(tree->right);

printf("%d\t", tree->data);

}

}

**//DELETE ELEMENT FROM THE TREE**

struct node \*deleteElement(struct node \*tree, int val)

{

struct node \*cur, \*parent, \*suc, \*psuc, \*ptr;

if(tree->left==NULL)

{

printf("\n The tree is empty ");

return(tree);

}

parent = tree;

cur = tree->left;

while(cur!=NULL && val!= cur->data)

{

parent = cur;

cur = (val<cur->data)? cur->left:cur->right;

}

if(cur == NULL)

{

printf("\n The value to be deleted is not present in the tree"); return(tree);

}

if(cur->left == NULL)

ptr = cur->right;

else if(cur->right == NULL)

ptr = cur->left;

else

{

* Find the in–order successor and its parent psuc = cur;

cur = cur->left;

while(suc->left!=NULL)

{

psuc = suc;

suc = suc->left;

}

if(cur==psuc)

{

// Situation 1

suc->left = cur->right;

}

else

{

// Situation 2

suc->left = cur->left;

psuc->left = suc->right;

suc->right = cur->right;

}

ptr = suc;

}

* Attach ptr to the parent node if(parent->left == cur) parent->left=ptr;

else

parent->right=ptr;

free(cur);

return tree;

}

**//DELETE THE ENTIRE TREE**

struct node \*deleteTree(struct node \*tree)

{

if(tree!=NULL)

{

deleteTree(tree->left);

deleteTree(tree->right);

free(tree);

}

}

**//SEARCH AN ELEMENT IN THE TREE** struct node \*search(struct node \*tree,int val)

{

if(tree->data==val||tree==NULL)

return tree;

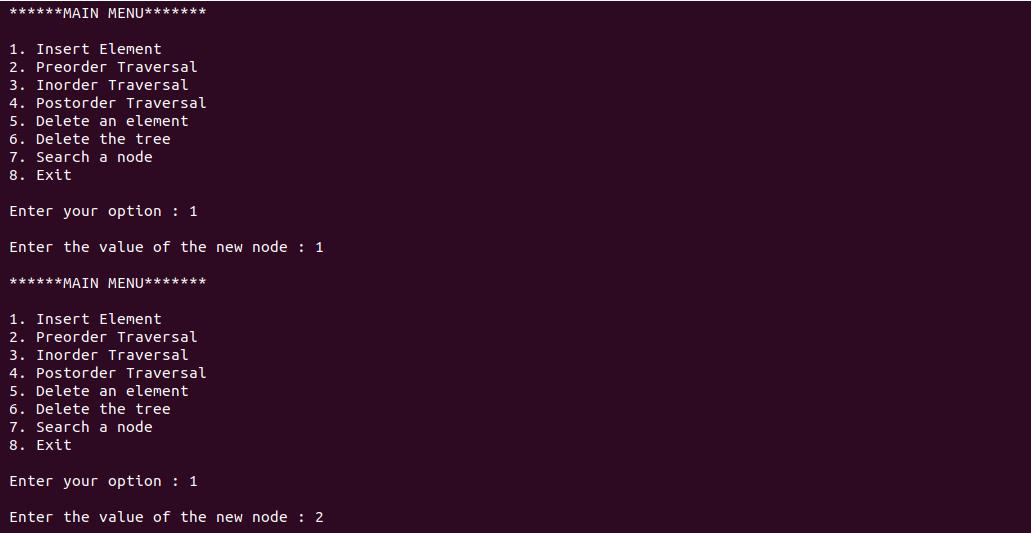
else if(val<tree->data)

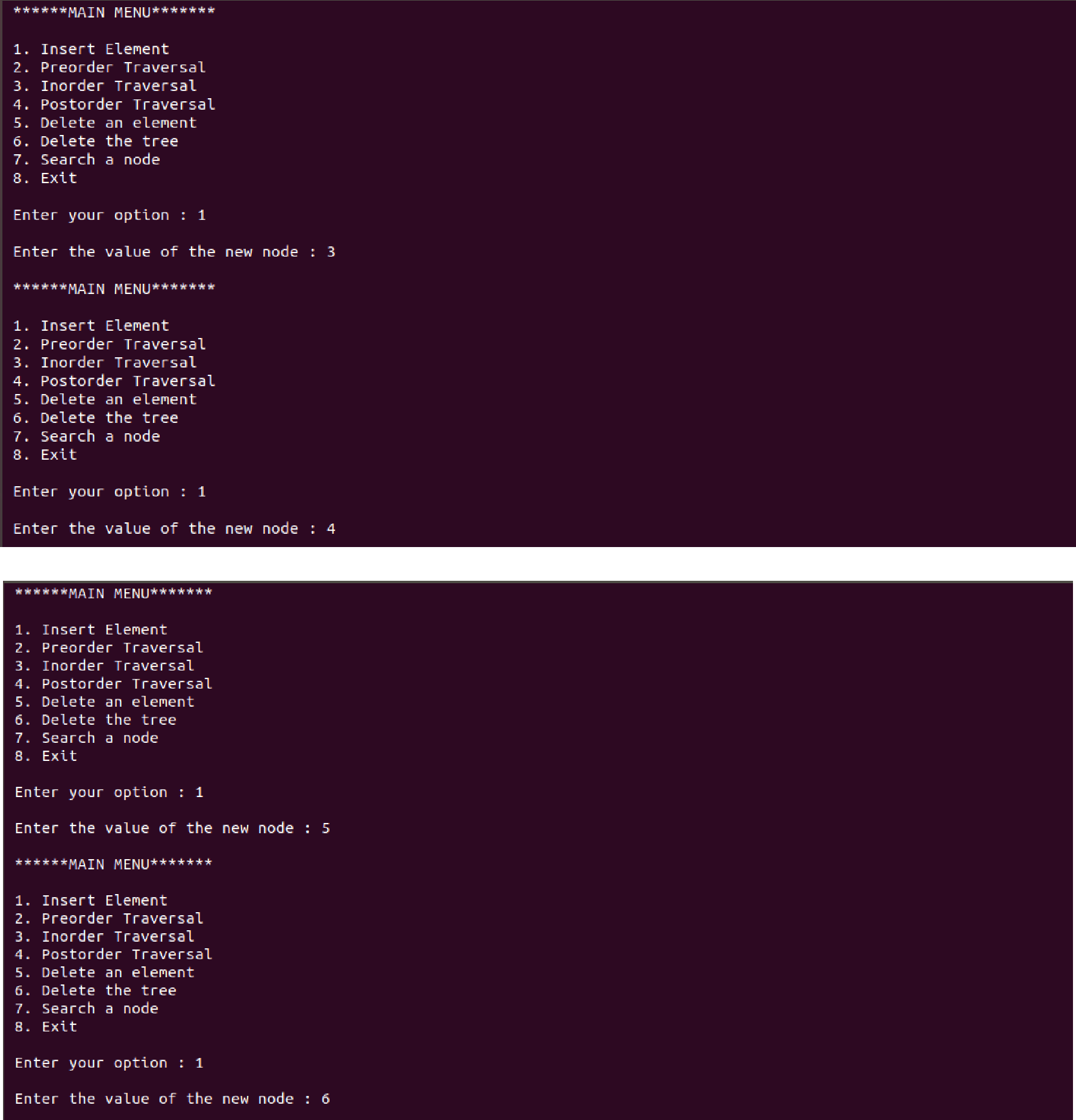
return search(tree->left,val);

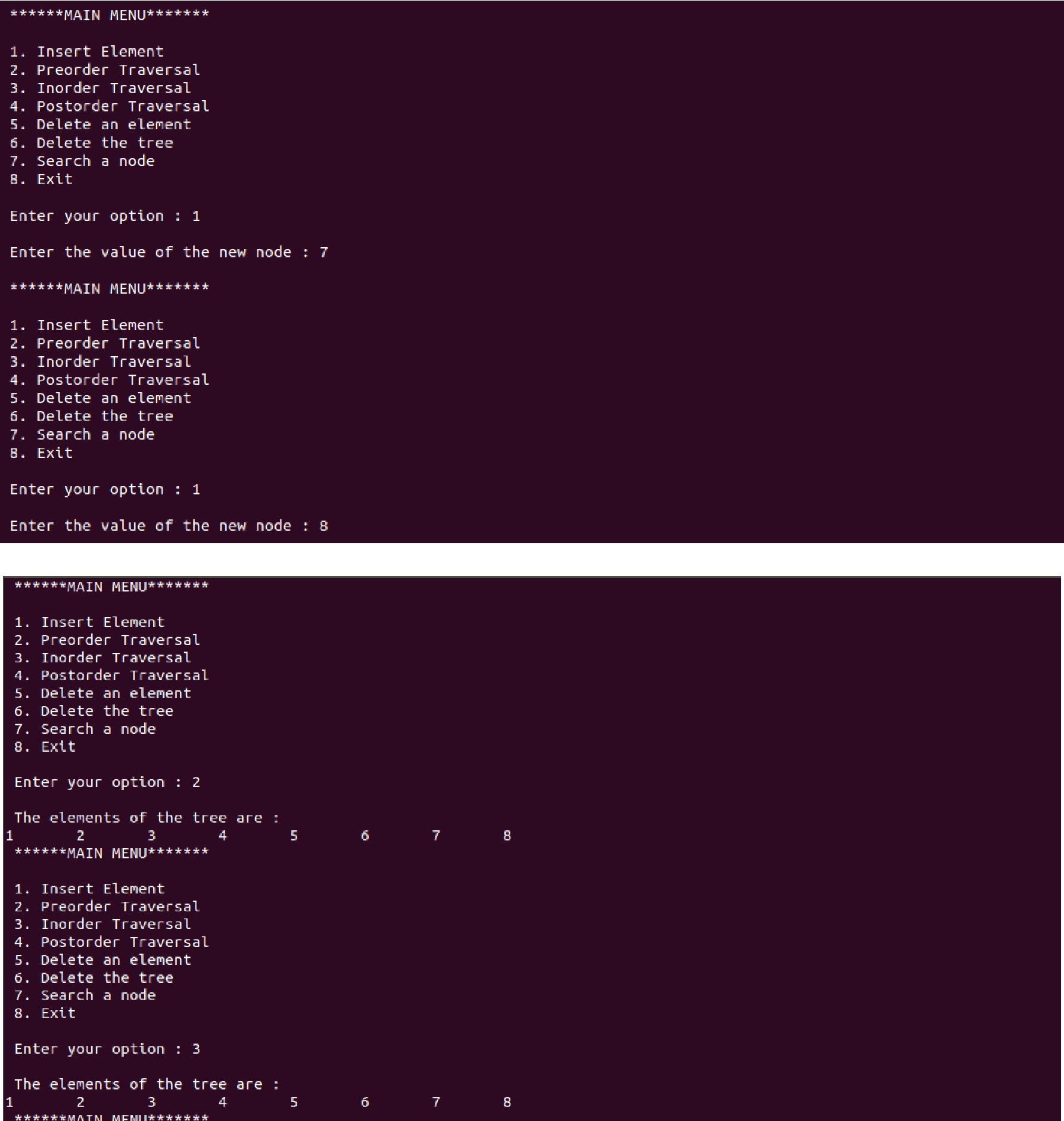
else

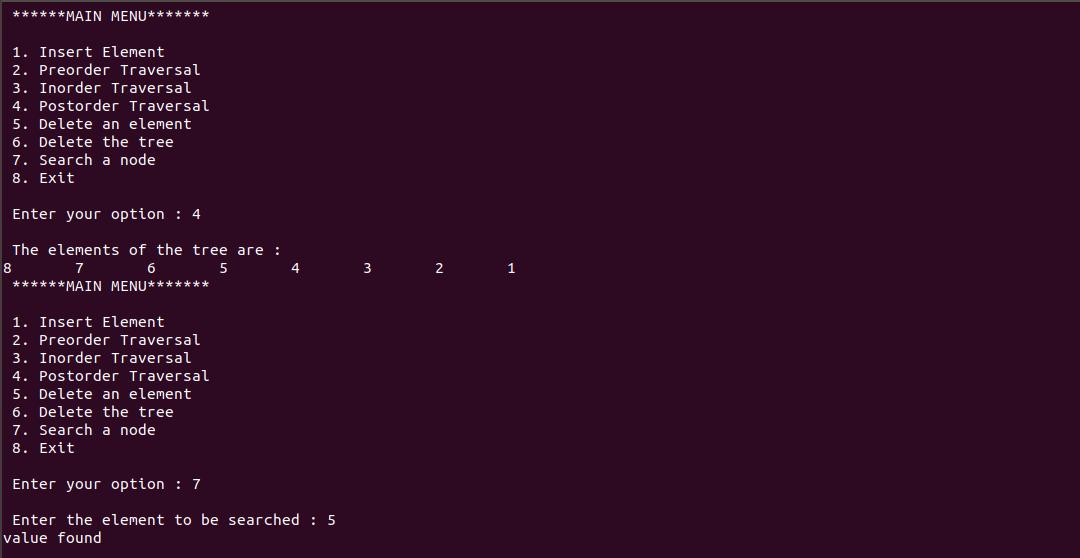
return search(tree->right,val);

}

**Output:**







**Q25. WAP to represent a graph using adjacency matrix.**

#include<stdio.h>

int a[50][50];

int n;

void main()

{

int choice;

do

{

printf("Adjacencey matrix representations:\n");

printf("ENter number of vertices:\n"); scanf("%d",&n);

printf("1.Unidirected graph\n2.Directed Graph\n3.Exit"); scanf("%d",&choice);

switch(choice)

{

case 1:undirg();

break;

case 2:dirg();

break;

case 3: exit(0);

}

}while(1);

}

void dirg()

{

int i,in\_deg,out\_deg,j;

read\_graph();

for(i=1;i<=n;i++)

{

in\_deg=0,out\_deg=0;

for(j=1;j<=n;j++)

{

if(a[i][j]==1)

in\_deg++;

else

out\_deg++;

}

printf("in\_Deg=%d%\nout\_Deg=%d",in\_deg,out\_deg);

}

}

void undirg()

{

int i,j,deg;

read\_graph();

for(i=1;i<=n;i++)

{

deg=0;

for(j=1;j<=n;j++)

{

if (a[i][j]==1)

deg++;

}

printf("Degree of %i is %d\n",i,deg);

}

}

void read\_graph()

{

int c,i,j;

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

printf("Are vertices %d and %d adjacent?(1/0)\n",i,j); scanf("%d",&c);

if(c==1)

a[i][j]=1;

else

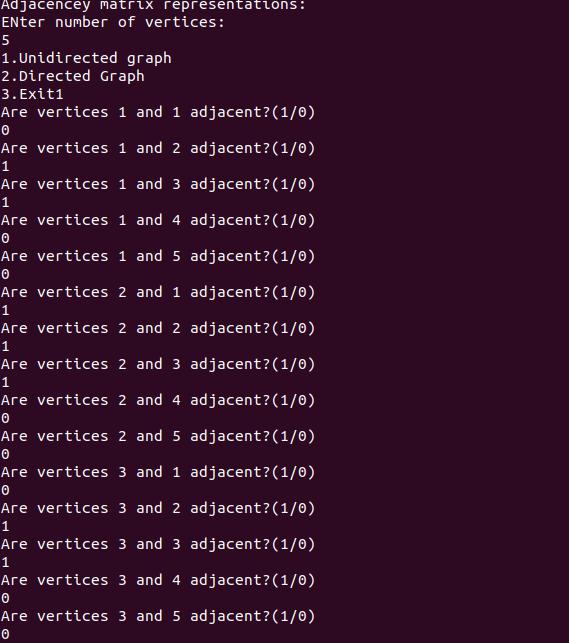
a[i][j]=0;

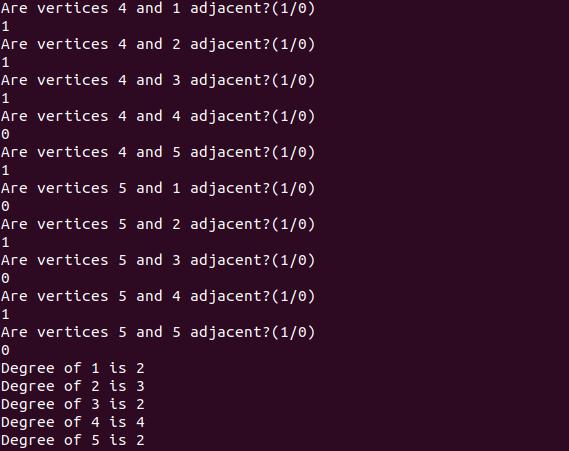
}

}

}

**Output:**





**Q26. WAP to represent a graph using weight matrix.**

#include<stdio.h>

#include<stdlib.h>

void view\_graph();

void read\_graph();

int w[50][50];

int n;

void main()

{

printf("Number of nodes?:\n");

scanf("%d",&n);

read\_graph();

view\_graph();

}

void view\_graph()

{

int i,j;

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

printf("%d ",w[i][j]);

}

printf("\n");

}

}

void read\_graph()

{

int i,j;

printf("PLease enter weights of vertices:\n");

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

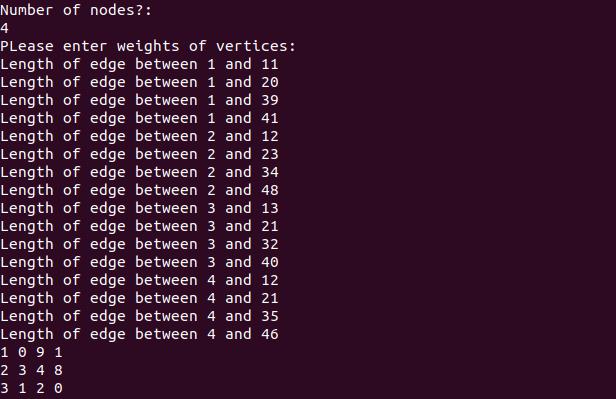
printf("Length of edge between %d and %d",i,j); scanf("%d",&w[i][j]);

}

}

}

**Output:**



**Q27. WAP to represent a graph using path matrix.**

#include<stdio.h>

#include<limits.h>

#define R 3

#define C 3

int min(int x, int y, int z);

int minCost(int cost[R][C], int m, int n)

{

if (n < 0 || m < 0)

return INT\_MAX;

else if (m == 0 && n == 0)

return cost[m][n];

else

return cost[m][n] + min( minCost(cost, m-1, n-1),

minCost(cost, m-1, n),

minCost(cost, m, n-1) );

}

int min(int x, int y, int z)

{

if (x < y)

return (x < z)? x : z;

else

return (y < z)? y : z;

}

int main()

{

int cost[R][C] = { {1, 2, 3},

{4, 8, 2},

{1, 5, 3} };

printf("Shortest path length between 2 and 2 : %d \n", minCost(cost, 2, 2));

return 0;

}

**Output:**



**Q28. Write a program to traverse graph using depth first search.**

#include<stdio.h>

int visited[10],adj[10][10];

int n;

void DFS(int i);

void main()

{

int i,j;

printf("Number of nodes?");

scanf("%d",&n);

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

if(i<j)

{

visited[i]=0;

printf("Does there a node exist between %d and

%d",i,j);

scanf("%d",&adj[i][j]);

}

}

}

DFS(1);

}

void DFS(int i)

{

int j;

printf("%d\n",i);

visited[i]=1;

for(j=1;j<=n;j++)

{

if(!visited[j] && adj[i][j]==1)

{

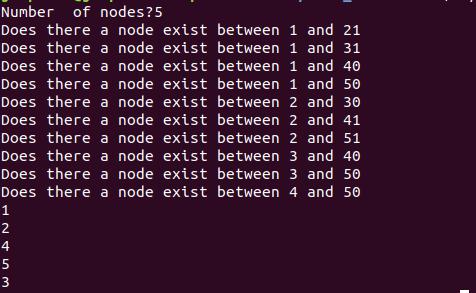
DFS(j);

}

}

}

**Output:**



**Q29. Write a program to traverse graph using breadth first search.**

#include<stdio.h>

#include<stdlib.h>

#define MAX 100

#define initial 1

#define waiting 2

#define visited 3

int n;

int adj[MAX][MAX];

int state[MAX];

void create\_graph();

void BF\_Traversal();

void BFS(int v);

int queue[MAX], front = -1,rear = -1;

void insert\_queue(int vertex);

int delete\_queue();

int isEmpty\_queue();

int main()

{

create\_graph();

BF\_Traversal();

return 0;

}

void BF\_Traversal()

{

int v;

for(v=0; v<n; v++)

state[v] = initial;

printf("Enter Start Vertex for BFS: \n"); scanf("%d", &v);

BFS(v);

}

void BFS(int v)

{

int i;

insert\_queue(v);

state[v] = waiting;

while(!isEmpty\_queue())

{

v = delete\_queue( );

printf("%d ",v);

state[v] = visited;

for(i=0; i<n; i++)

{

if(adj[v][i] == 1 && state[i] == initial)

{

insert\_queue(i);

state[i] = waiting;

}

}

}

printf("\n");

}

void insert\_queue(int vertex)

{

if(rear == MAX-1)

printf("Queue Overflow\n");

else

{

if(front == -1)

front = 0;

rear = rear+1;

queue[rear] = vertex ;

}

}

int isEmpty\_queue()

{

if(front == -1 || front > rear)

return 1;

else

return 0;

}

int delete\_queue()

{

int delete\_item;

if(front == -1 || front > rear)

{

printf("Queue Underflow\n");

exit(1);

}

delete\_item = queue[front];

front = front+1;

return delete\_item;

}

void create\_graph()

{

int count,max\_edge,origin,destin;

printf("Enter number of vertices : "); scanf("%d",&n);

max\_edge = n\*(n-1);

for(count=1; count<=max\_edge; count++)

{

printf("Enter edge %d( -1 -1 to quit ) : ",count); scanf("%d %d",&origin,&destin);

if((origin == -1) && (destin == -1))

break;

if(origin>=n || destin>=n || origin<0 || destin<0)

{

printf("Invalid edge!\n");

count--;

}

else

{

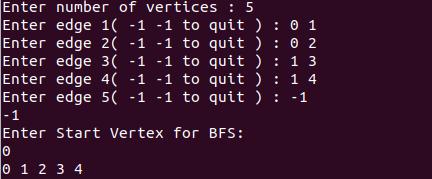
adj[origin][destin] = 1;

}

}

}

**Output:**



**Q30. Write a program to implement Dijkstra’s algorithm.**

#include<stdio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main()

{

int G[MAX][MAX],i,j,n,u;

printf("Enter no. of vertices:");

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

for(i=0;i<n;i++)

for(j=0;j<n;j++)

scanf("%d",&G[i][j]);

printf("\nEnter the starting node:");

scanf("%d",&u);

dijkstra(G,n,u);

return 0;

}

void dijkstra(int G[MAX][MAX],int n,int startnode)

{

int cost[MAX][MAX],distance[MAX],pred[MAX]; int visited[MAX],count,mindistance,nextnode,i,j;

**//pred[] stores the predecessor of each node** /**/count gives the number of nodes seen so far** **//create the cost matrix**

for(i=0;i<n;i++)

for(j=0;j<n;j++)

if(G[i][j]==0)

cost[i][j]=INFINITY;

else

cost[i][j]=G[i][j];

**//initialize pred[],distance[] and visited[]** for(i=0;i<n;i++)

{

distance[i]=cost[startnode][i];

pred[i]=startnode;

visited[i]=0;

}

distance[startnode]=0;

visited[startnode]=1;

count=1;

while(count<n-1)

{

mindistance=INFINITY;

**//nextnode gives the node at minimum distance** for(i=0;i<n;i++)

if(distance[i]<mindistance&&!visited[i])

{

mindistance=distance[i];

nextnode=i;

}

**//check if a better path exists through nextnode** visited[nextnode]=1;

for(i=0;i<n;i++)

if(!visited[i])

if(mindistance+cost[nextnode][i]<distance[i])

{

distance[i]=mindistance+cost[nextnode][i];

pred[i]=nextnode;

}

count++;

}

**//print the path and distance of each node** for(i=0;i<n;i++)

if(i!=startnode)

{

printf("\nDistance of node%d=%d",i,distance[i]);

printf("\nPath=%d",i);

j=i;

do

{

j=pred[j];

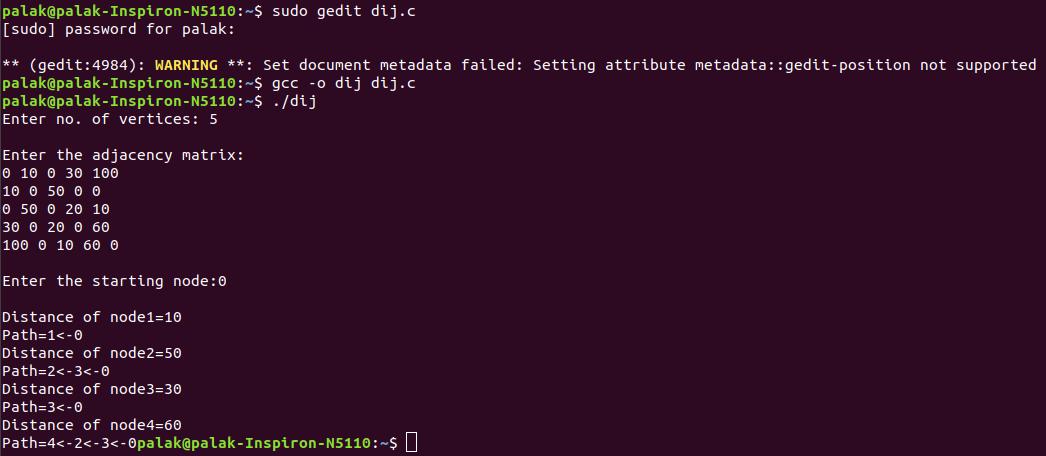
printf("<-%d",j);

}while(j!=startnode);

}

}

**Output:**



**Q31. Write a program to implement Prim’s algorithm.**

#include<stdio.h>

#include<stdlib.h>

#define infinity 9999

#define MAX 20

int G[MAX][MAX],spanning[MAX][MAX],n;

int prims();

int main()

{

int i,j,total\_cost;

printf("Enter no. of vertices:");

scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

for(i=0;i<n;i++)

for(j=0;j<n;j++)

scanf("%d",&G[i][j]);

total\_cost=prims();

printf("\nspanning tree matrix:\n");

for(i=0;i<n;i++)

{

printf("\n");

for(j=0;j<n;j++)

printf("%d\t",spanning[i][j]);

}

printf("\n\nTotal cost of spanning tree=%d",total\_cost);

return 0;

}

int prims()

{

int cost[MAX][MAX];

int u,v,min\_distance,distance[MAX],from[MAX];

int visited[MAX],no\_of\_edges,i,min\_cost,j;

**//create cost[][] matrix,spanning[][]** for(i=0;i<n;i++)

for(j=0;j<n;j++)

{

if(G[i][j]==0)

cost[i][j]=infinity;

else

cost[i][j]=G[i][j];

spanning[i][j]=0;

}

**//initialise visited[],distance[] and from[]** distance[0]=0;

visited[0]=1;

for(i=1;i<n;i++)

{

distance[i]=cost[0][i];

from[i]=0;

visited[i]=0;

}

min\_cost=0; /**/cost of spanning tree**

no\_of\_edges=n-1; **//no. of edges to be added**

while(no\_of\_edges>0)

{

**//find the vertex at minimum distance from the tree** min\_distance=infinity;

for(i=1;i<n;i++)

if(visited[i]==0&&distance[i]<min\_distance)

{

v=i;

min\_distance=distance[i];

}

u=from[v];

**//insert the edge in spanning tree**

spanning[u][v]=distance[v];

spanning[v][u]=distance[v];

no\_of\_edges--;

visited[v]=1;

**//updated the distance[] array**

for(i=1;i<n;i++)

if(visited[i]==0&&cost[i][v]<distance[i])

{

distance[i]=cost[i][v];

from[i]=v;

}

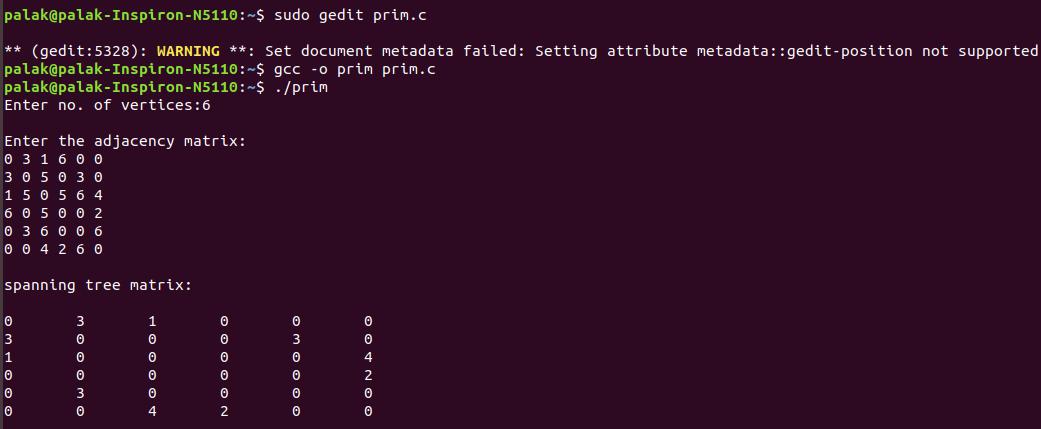
min\_cost=min\_cost+cost[u][v];

}

return(min\_cost);

}

**Output:**



**Q32. Write a program to implement Kruskal’s algorithm.**

#include<stdio.h>

#define MAX 30

typedef struct edge

{

int u,v,w;

}edge;

typedef struct edgelist

{

edge data[MAX];

int n;

}edgelist;

edgelist elist;

int G[MAX][MAX],n;

edgelist spanlist;

void kruskal();

int find(int belongs[],int vertexno);

void union1(int belongs[],int c1,int c2);

void sort();

void print();

void main()

{

int i,j,total\_cost;

printf("\nEnter number of vertices:"); scanf("%d",&n);

printf("\nEnter the adjacency matrix:\n");

for(i=0;i<n;i++)

for(j=0;j<n;j++)

scanf("%d",&G[i][j]);

kruskal();

print();

}

void kruskal()

{

int belongs[MAX],i,j,cno1,cno2;

elist.n=0;

for(i=1;i<n;i++)

for(j=0;j<i;j++)

{

if(G[i][j]!=0)

{

elist.data[elist.n].u=i;

elist.data[elist.n].v=j;

elist.data[elist.n].w=G[i][j];

elist.n++;

}

}

sort();

for(i=0;i<n;i++)

belongs[i]=i;

spanlist.n=0;

for(i=0;i<elist.n;i++)

{

cno1=find(belongs,elist.data[i].u);

cno2=find(belongs,elist.data[i].v);

if(cno1!=cno2)

{

spanlist.data[spanlist.n]=elist.data[i];

spanlist.n=spanlist.n+1;

union1(belongs,cno1,cno2);

}

}

}

int find(int belongs[],int vertexno)

{

return(belongs[vertexno]);

}

void union1(int belongs[],int c1,int c2)

{

int i;

for(i=0;i<n;i++)

if(belongs[i]==c2)

belongs[i]=c1;

}

void sort()

{

int i,j;

edge temp;

for(i=1;i<elist.n;i++)

for(j=0;j<elist.n-1;j++)

if(elist.data[j].w>elist.data[j+1].w)

{

temp=elist.data[j];

elist.data[j]=elist.data[j+1];

elist.data[j+1]=temp;

}

}

void print()

{

int i,cost=0;

for(i=0;i<spanlist.n;i++)

{

printf("\n%d\t%d\t%d",

spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);

cost=cost+spanlist.data[i].w;

}

printf("\n\nCost of the spanning tree=%d",cost);

}

**Output:**



**Q33. Write a program to implement Merge Sort algorithm.**

#include<stdio.h>

void mergesort(int a[],int i, int j);

void merge(int a[],int i1, int j1, int i2, int j2);

int main()

{

int a[30];

int n,i;

printf("Enter number of elements: "); scanf("%d",&n);

printf("Enter array elements: ");

for(i=0;i<n;i++)

{

scanf("%d",&a[i]);

}

mergesort(a,0,n-1);

printf("\nSorted array is: \n");

for(i=0;i<n;i++)

{

printf("%d ",a[i]);

}

printf("\n");

return 0;

}

void mergesort(int a[], int i, int j)

{

int mid;

if(i<j)

{

mid=(i+j)/2;

mergesort(a,i,mid);

mergesort(a,mid+1,j);

merge(a,i,mid,mid+1,j);

}

}

void merge(int a[], int i1, int j1, int i2, int j2)

{

int temp[50];

int i,j,k;

i=i1;

j=i2;

k=0;

while(i<=j1 && j<=j2)

{

if(a[i]<a[j])

temp[k++]=a[i++];

else

temp[k++]=a[j++];

}

while(i<=j1)

temp[k++]=a[i++];

while(j<=j2)

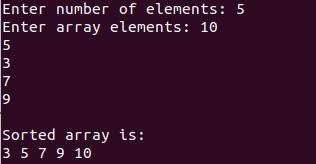
temp[k++]=a[j++];

for(i=i1,j=0;i<=j2;i++,j++)

a[i]=temp[j];

}

**Output:**



**Q34. Write a program to implement Quick Sort algorithm.**

#include<stdio.h>

void quicksort(int number[25],int first,int last){

int i, j, pivot, temp;

if(first<last)

{

pivot=first;

i=first;

j=last;

while(i<j)

{

while(number[i]<=number[pivot]&&i<last)

i++;

while(number[j]>number[pivot])

j--;

if(i<j)

{

temp=number[i];

number[i]=number[j];

number[j]=temp;

}

}

temp=number[pivot];

number[pivot]=number[j];

number[j]=temp;

quicksort(number,first,j-1);

quicksort(number,j+1,last);

}

}

int main()

{

int i, count, number[25];

printf("How many elements are u going to enter?: ");

scanf("%d",&count);

printf("Enter %d elements: ", count);

for(i=0;i<count;i++)

scanf("%d",&number[i]); quicksort(number,0,count-1); printf("Order of Sorted elements: ");

for(i=0;i<count;i++)

printf(" %d",number[i]);

return 0;

}

**Output:**



Q18. Doubly linked list.

#include&lt;stdio.h&gt;

struct list

{

int info;

struct list \* next;

struct list \* prev;

}\*n,\*z,\*h;

void new\_node()

{

int no;

n=(struct list\*)malloc(sizeof(struct list));

printf(&quot;\n enter the no&quot;);

scanf(&quot;%d&quot;,&amp;no);

n-&gt;info=no;

n-&gt;next=NULL;

n-&gt;prev=NULL;

}

void create()

{

char ans;

do

{

new\_node();

if(h==NULL)

h=z=n;

else

{

z-&gt;next=n;

n-&gt;prev=z;

z=n;

}

printf(&quot;\n do you want to continue inserting nodes: y or n&quot;); fflush(stdin);

scanf(&quot;%c&quot;,&amp;ans);

}while(ans==&#39;y&#39;);

}

void insatbeg()

{

new\_node();

h-&gt;prev=n;

n-&gt;next=h;

h=n;

}

void insatend()

{

new\_node();

for(z=h;z-&gt;next!=NULL;z=z- &gt;next);

z-&gt;next=n;

n-&gt;prev=z;

}

void insatpos()

{

int p,i;

new\_node();

printf(&quot;\n enter the position where you want to insert&quot;); scanf(&quot;%d&quot;,&amp;p); for(i=1,z=h;i&lt;=p-2;i++,z=z- &gt;next); z-&gt;next- &gt;prev=n;

n-&gt;next=z- &gt;next;

z-&gt;next=n;

n-&gt;prev=z;

}

void delatbeg()

{

printf(&quot;\n %d is deleted&quot;,h-&gt;info);

n=h;

h=n-&gt;next;

h-&gt;prev=NULL;

free(n);

}

void delatend()

{

for(z=h;z-&gt;next- &gt;next!=NULL;z=z-&gt;next);

printf(&quot;\n %d is deleted&quot;,z-&gt;next- &gt;info);

n=z-&gt;next;

z-&gt;next=NULL;

n-&gt;prev=NULL;

free(n);

}

void delatpos()

{

int p,i;

printf(&quot;\n enter the position from where you want to delete&quot;);

scanf(&quot;%d&quot;,&amp;p);

for(i=1,z=h;i&lt;=p-2;i++,z=z- &gt;next);

printf(&quot;\n%d is deleted&quot;,z-&gt;next- &gt;info);

n=z-&gt;next;

z-&gt;next=n- &gt;next;

n-&gt;prev=NULL;

n-&gt;next- &gt;prev=z;

free(n);

}

void display()

{

printf(&quot;\n left to right \n&quot;);

for(z=h;z!=NULL;z=z-&gt;next)

{

printf(&quot;%d\t&quot;,z-&gt;info);

}

printf(&quot;\n right to left \n&quot;);

for(z=h;z-&gt;next!=NULL;z=z- &gt;next);

while(z-&gt;prev!=NULL)

{

printf(&quot;%d\t&quot;,z-&gt;info);

z=z-&gt;prev;

}

printf(&quot;%d\t&quot;,z-&gt;info);

}

void search()

{

int no,flag=0,p=1;

printf(&quot;\n enter the no you want to search&quot;); scanf(&quot;%d&quot;,&amp;no); for(z=h;z!=NULL;z=z-&gt;next) {

if(z-&gt;info==no)

{

flag=1;

printf(&quot;\n %d is found at %d position&quot;,no,p);

break;

}

p++;

}

if(flag==0)

printf(&quot;\n %d is not found&quot;,no);

}

void count()

{

int count=0;

for(z=h;z!=NULL;z=z-&gt;next)

count++;

printf(&quot;\n No of nodes = %d&quot;,count);

}

void main()

{

char ans;

int ch;

do

{

printf(&quot;\n 1. create doubly list&quot;); printf(&quot;\n 2. insert at start&quot;); printf(&quot;\n 3. insert at end&quot;); printf(&quot;\n 4. insert at given position&quot;); printf(&quot;\n 5. delete at start&quot;); printf(&quot;\n 6. delete at end&quot;); printf(&quot;\n 7. delete at given position&quot;); printf(&quot;\n 8. traverse the list&quot;); printf(&quot;\n 9. search&quot;); printf(&quot;\n 10. count nodes&quot;); printf(&quot;\n enter your choice&quot;); scanf(&quot;%d&quot;,&amp;ch);

switch(ch)

{

case 1 : create();

break;

case 2 : insatbeg();

break;

case 3 : insatend();

break;

case 4 : insatpos();

break;

case 5 : delatbeg();

break;

case 6 : delatend();

break;

case 7 : delatpos();

break;

case 8 : display();

break;

case 9 : search();

break;

case 10: count();

break;

default : printf(&quot;\n wrong input&quot;);

break;

}

printf(&quot;\n do you want to continue: y or n&quot;);

fflush(stdin);

scanf(&quot;%c&quot;,&amp;ans);

}while(ans==&#39;y&#39;);

}