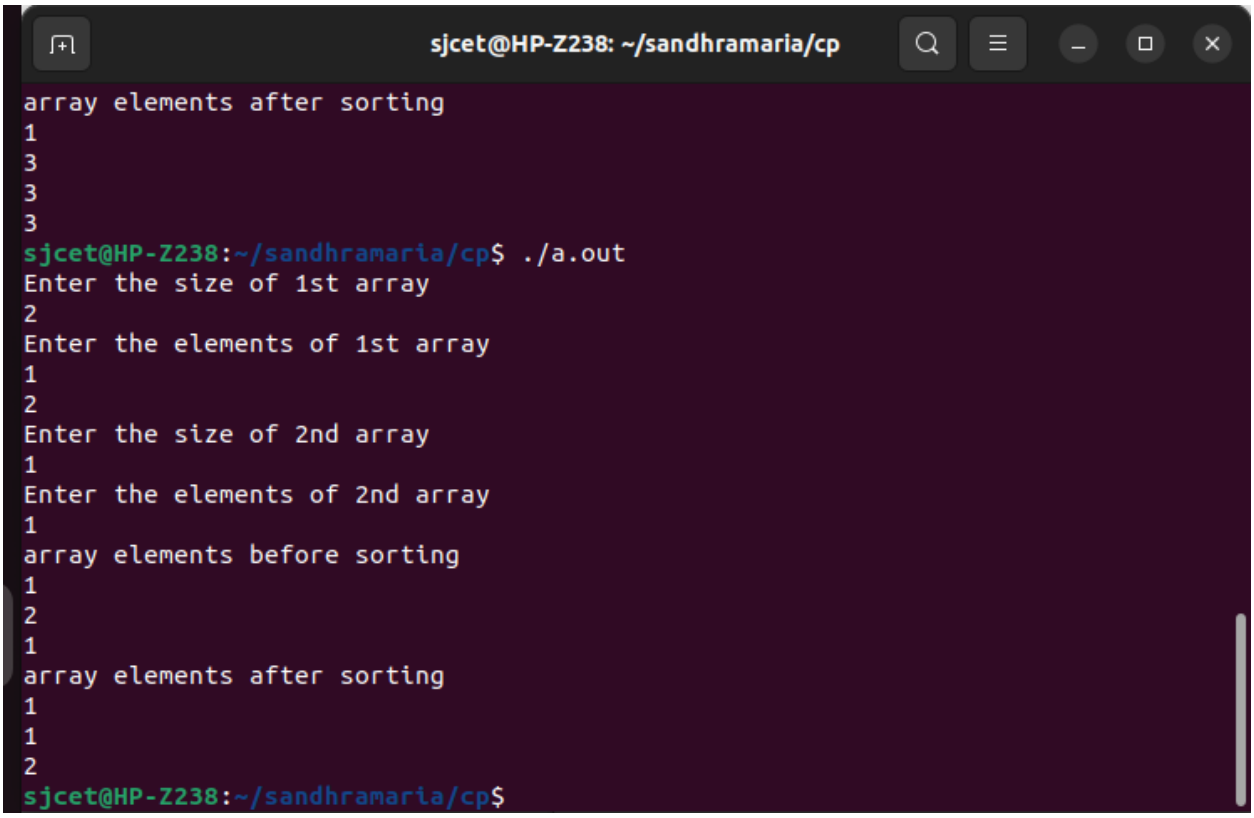
ADVANCED DATA STRUCTURES - USING C

INDEX

1 . Merge 2 sorted array

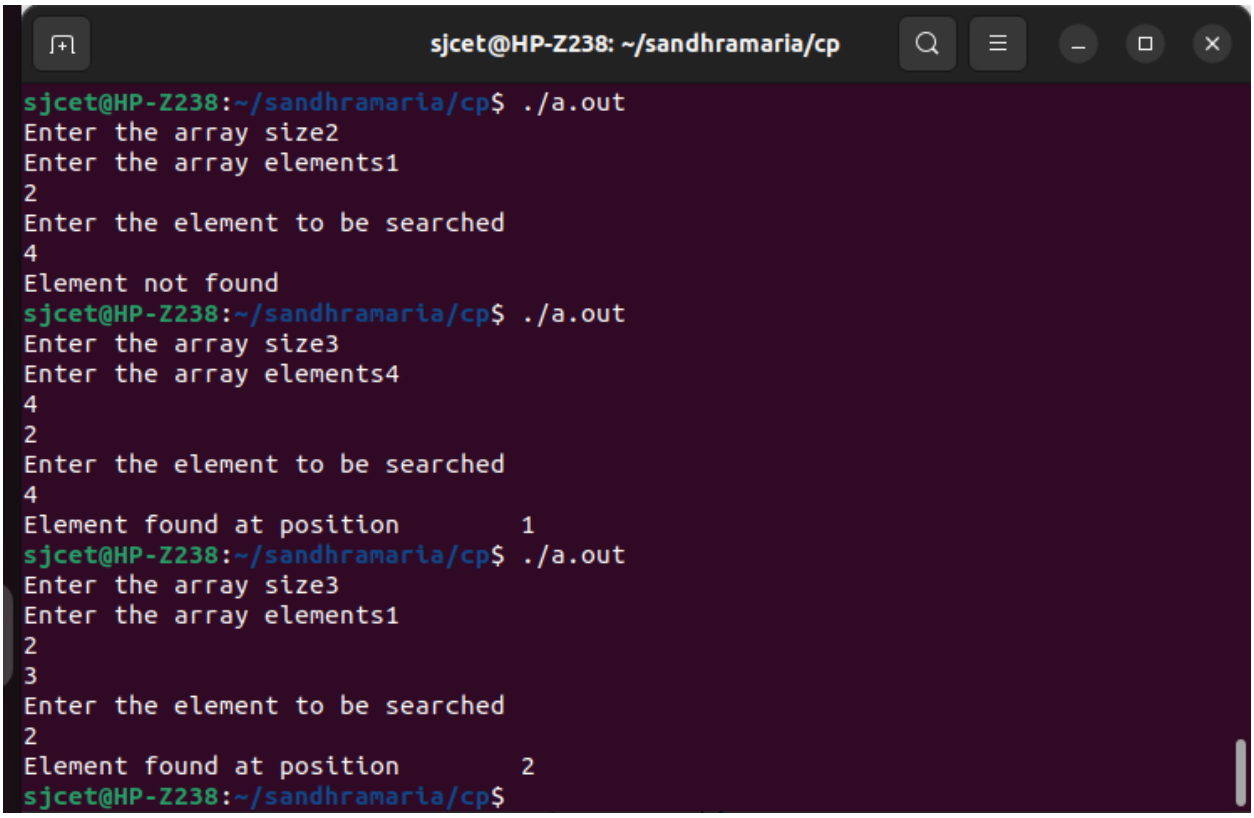
```
#include <stdio.h>
void main(){
int array1[100],array2[100],array3[100],size1,size2,size3;
printf("Enter the size of 1st array\n");
scanf("%d",&size1);
printf("Enter the elements of 1st array\n");
for(int i=0;i<size1;i++)
scanf("%d",&array1[i]);
printf("Enter the size of 2nd array\n");
scanf("%d",&size2);
printf("Enter the elements of 2nd array\n");
for(int i=0;i<size2;i++)
scanf("%d",&array2[i]);
size3=size1+size2;
for(int i=0;i<size1;i++)
array3[i]=array1[i];
for(int i=0;i<size2;i++)
array3[i+size1]=array2[i];
printf("array elements before sorting\n");
for(int i=0;i<size3;i++)
```

```
printf("%d \n",array3[i]);
for(int i = 0; i < size3; i++)
{
int temp;
for(int j = i + 1; j < size3; j++)
{
if(array3[i] > array3[j])
{
temp = array3[i];
array3[i] = array3[j];
array3[j] = temp;
}
}
}
printf("array elements after sorting\n");
for(int i=0;i<size3;i++)
printf("%d \n",array3[i]);
}
```


sjcet@HP-Z238: ~/sandhramaria/cp
array elements after sorting
1
3
3
3
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
Enter the size of 1st array
2
Enter the elements of 1st array
1
2
Enter the size of 2nd array
1
Enter the elements of 2nd array
1
array elements before sorting
1
2
1
array elements after sorting
1
1
2
sjcet@HP-Z238:~/sandhramaria/cp$

2.Linear search

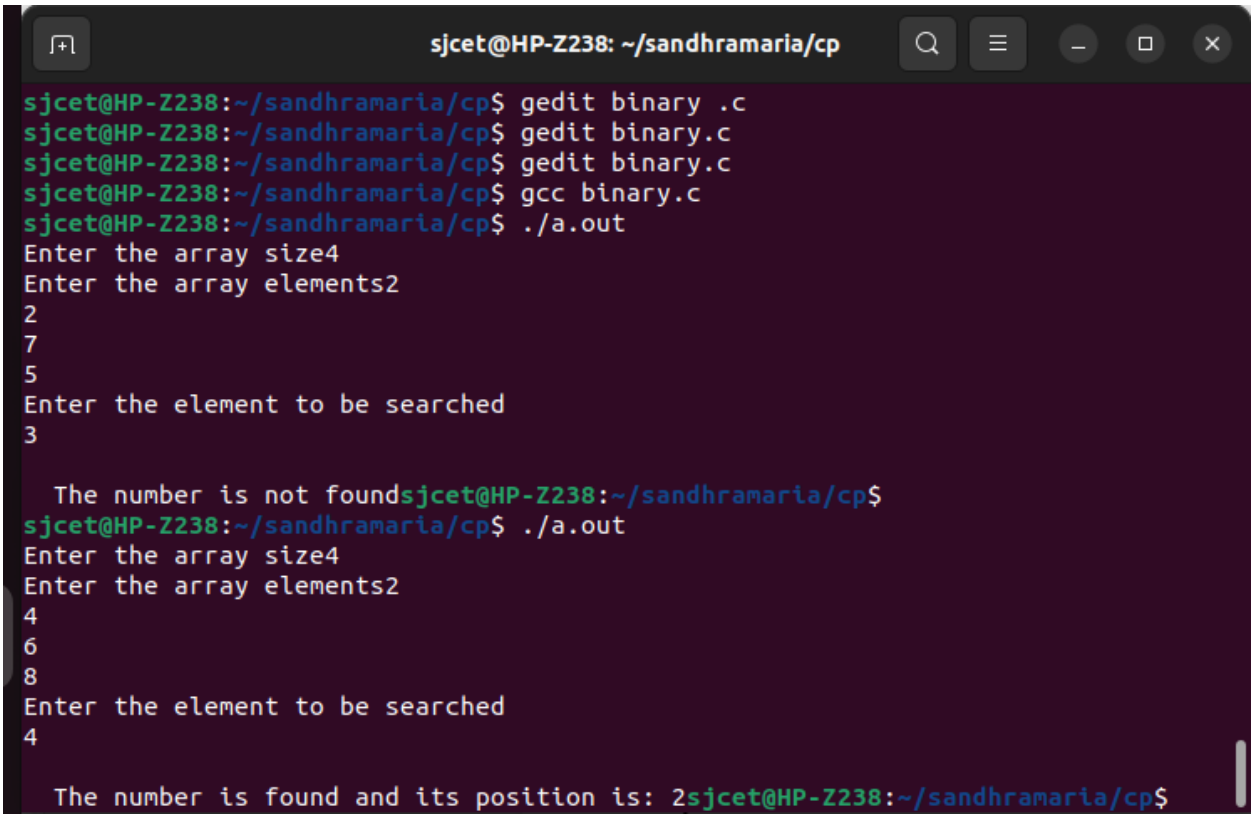
```
#include<stdio.h>
void main()
{
int a[100],n,s,flag=0;
printf("Enter the array size");
scanf("%d",&n);
printf("Enter the array elements");
for(int i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
printf("Enter the element to be searched\n");
scanf("%d",&s);
for(int i=0;i<n;i++)
{
if(a[i]==s)
{
printf("Element found at position\t %d \n",i+1);
flag=1;
break;
}
}
if(flag==0)
{
printf("Element not found\n");
}

}
```


sjcet@HP-Z238: ~/sandhramaria/cp
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
Enter the array size2
Enter the array elements1
2
Enter the element to be searched
4
Element not found
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
Enter the array size3
Enter the array elements4
4
2
Enter the element to be searched
4
Element found at position 1
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
Enter the array size3
Enter the array elements1
2
3
Enter the element to be searched
2
Element found at position 2
sjcet@HP-Z238:~/sandhramaria/cp$

3. Binary Search

```
#include<stdio.h>
void main()
{
int a[100],n,s,first,last,mid,flag;
printf("Enter the array size");
scanf("%d",&n);
printf("Enter the array elements");
for(int i=0;i<n;i++)
{
scanf("%d",&a[i]);
}
printf("Enter the element to be searched\n");
scanf("%d",&s);
first=0;
last=n-1;
flag=0;
while(first<=last)
{
mid=(first+last)/2;
if(s==a[mid])
{
flag=1;
break;
}
else if(s>a[mid])
{
  first=mid+1;
}
else
{
   last=mid-1;
}
}
```

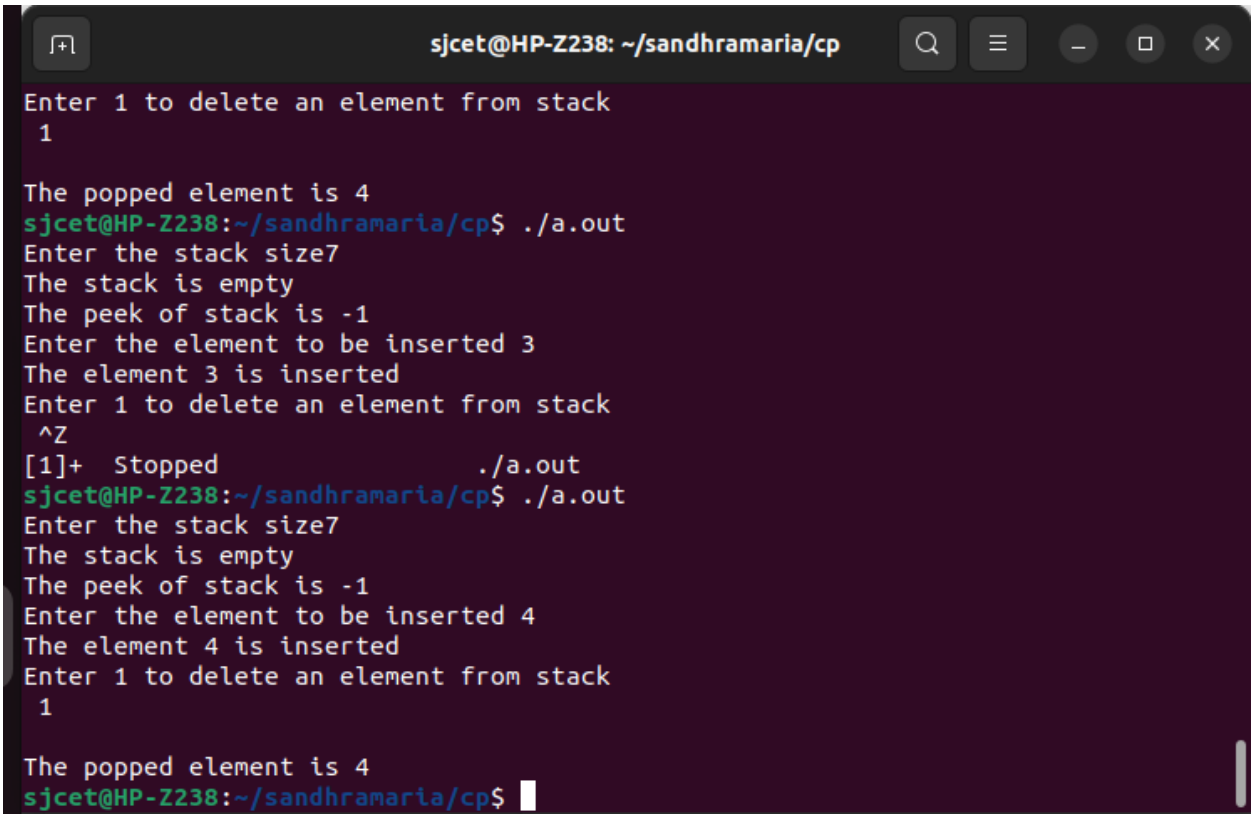
```
    if(flag==0)
    printf("\n  The number is not found");
    else
    printf("\n  The number is found and its position is: %d",mid+1);
}
```



4.Stack Implementation

```
#include<stdio.h>
void push();
void pop();
void peek();
int isfull();
int isempty();
int stack[100],maxsize,top=-1;
void main()
{

   printf("Enter the stack size");
   scanf("%d",&maxsize);
   isempty();
   isfull();
   peek();
   int item;
   printf("Enter the element to be inserted ");
   scanf("%d",&item);
   push(item);
   int term;
   printf("Enter 1 to delete an element from stack\n ");
   scanf("%d",&term);
   if(term==1)
       pop();
}
int isempty()
{
   if(top==-1)
      {
         printf("The stack is empty\n");
         return 0;
      }
   else
```

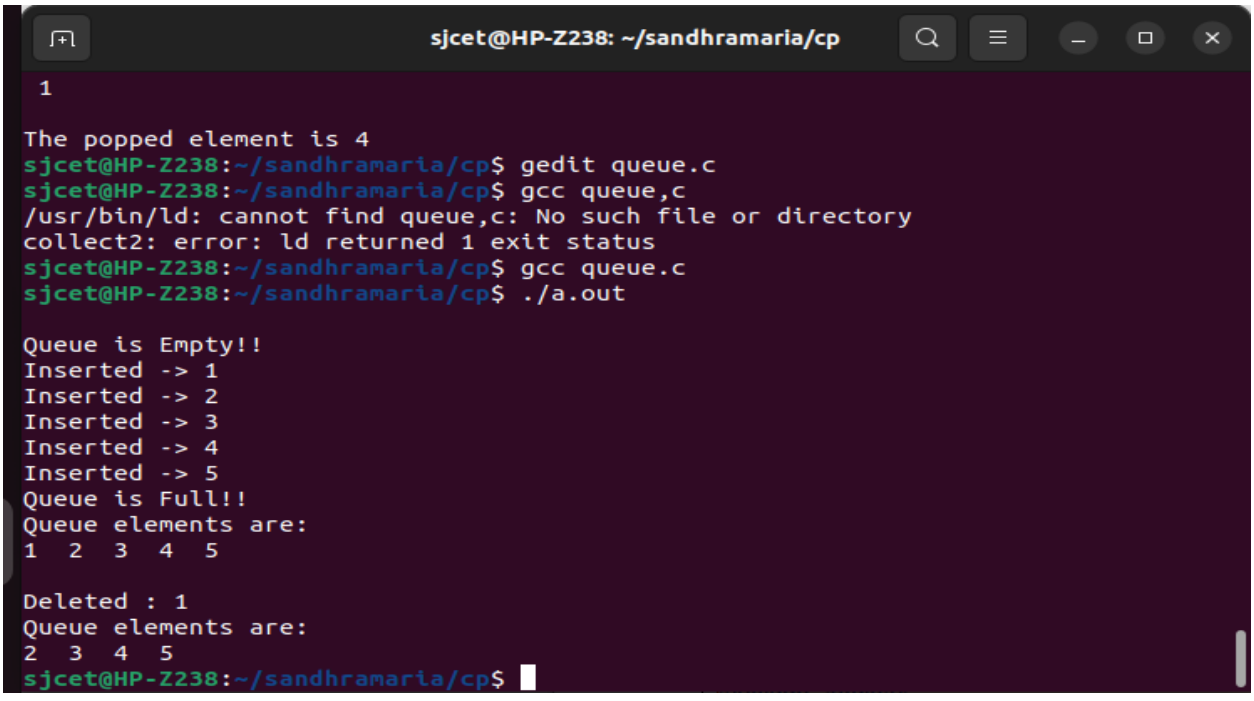
```
     return 1;
}
int isfull()
{
    if(top==maxsize)
      {
         printf("The stack is full\n");
         return 0;
      }
     else
         return 1;
}
void peek()
{
     printf("The peek of stack is %d\t\n",top);
}
void push(int data)
{
    if(isfull()==1)
       {
          top=top+1;
          stack[top]=data;
          printf("The element %d is inserted \n",data);
       }

}
void pop()
{
    if(isempty()==1)
       {
          printf("\nThe popped element is %d \n",stack[top]);
          top=top-1;

       }
}
```


sjcet@HP-Z238: ~/sandhramaria/cp
Enter 1 to delete an element from stack
1
The popped element is 4
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
Enter the stack size7
The stack is empty
The peek of stack is -1
Enter the element to be inserted 3
The element 3 is inserted
Enter 1 to delete an element from stack
^Z
[1]+ Stopped ./a.out
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
Enter the stack size7
The stack is empty
The peek of stack is -1
Enter the element to be inserted 4
The element 4 is inserted
Enter 1 to delete an element from stack
1
The popped element is 4
sjcet@HP-Z238:~/sandhramaria/cp$

5. Queue Implementation (linear)

```
#include <stdio.h>
#define SIZE 5
void enQueue(int);
void deQueue();
void display();
int items[SIZE], front = -1, rear = -1;
int main() {
  deQueue()
  enQueue(1);
  enQueue(2);
  enQueue(3);
  enQueue(4);
  enQueue(5);
  enQueue(6);
  display();
  deQueue();
  display();
  return 0;
}
void enQueue(int value) {
  if (rear == SIZE - 1)
    printf("\nQueue is Full!!");
  else {
    if (front == -1)
      front = 0;
    rear++;
    items[rear] = value;
    printf("\nInserted -> %d", value);
  }
}
void deQueue() {
  if (front == -1)
    printf("\nQueue is Empty!!");
```

```
  else {
    printf("\nDeleted : %d", items[front]);
    front++;
    if (front > rear)
      front = rear = -1;
  }
}
void display() {
  if (rear == -1)
    printf("\nQueue is Empty!!!");
  else {
    int i;
    printf("\nQueue elements are:\n");
    for (i = front; i <= rear; i++)
      printf("%d  ", items[i]);
  }
  printf("\n");
}
```



6. Circular Queue Implementation

```
#include <stdio.h>
#define SIZE 5
int items[SIZE];
int front = -1, rear = -1;
int isFull() {
  if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return 1;
  return 0;
}
int isEmpty() {
  if (front == -1) return 1;
  return 0;
}
void enQueue(int element) {
  if (isFull())
    printf("\n Queue is full!! \n");
  else {
    if (front == -1) front = 0;
    rear = (rear + 1) % SIZE;
    items[rear] = element;
    printf("\n Inserted -> %d", element);
  }
}
int deQueue() {
  int element;
  if (isEmpty()) {
    printf("\n Queue is empty !! \n");
    return (-1);
  } else {
    element = items[front];
    if (front == rear) {
      front = -1;
      rear = -1;
    }
```

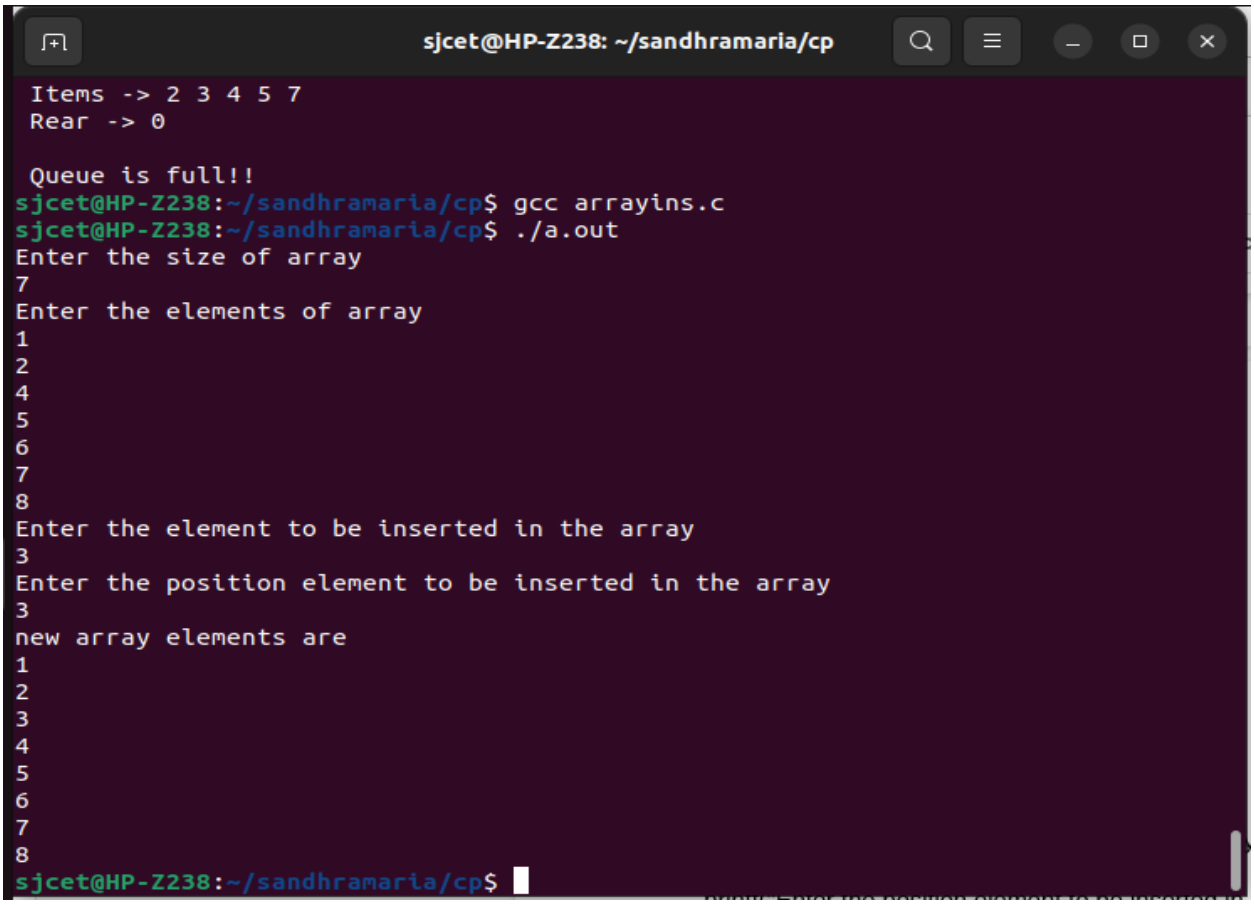
```
    else {
      front = (front + 1) % SIZE;
    }
    printf("\n Deleted element -> %d \n", element);
    return (element);
  }
}
void display() {
  int i;
  if (isEmpty())
    printf(" \n Empty Queue\n");
  else {
    printf("\n Front -> %d ", front);
    printf("\n Items -> ");
    for (i = front; i != rear; i = (i + 1) % SIZE) {
      printf("%d ", items[i]);
    }
    printf("%d ", items[i]);
    printf("\n Rear -> %d \n", rear);
  }
}
int main() {
   deQueue();
  enQueue(1);
  enQueue(2);
  enQueue(3);
  enQueue(4);
  enQueue(5);
 enQueue(6);
  display();
  deQueue();
  display();
  enQueue(7);
  display();
  enQueue(8);
```

```
  return 0;
}
```


sjcet@HP-Z238: ~/sandhramaria/cp
sjcet@HP-Z238:~/sandhramaria/cp$ gedit circqueue.c
sjcet@HP-Z238:~/sandhramaria/cp$ gcc circqueue.c
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
Queue is empty !!
Inserted -> 1
Inserted -> 2
Inserted -> 3
Inserted -> 4
Inserted -> 5
Queue is full!!
Front -> 0
Items -> 1 2 3 4 5
Rear -> 4
Deleted element -> 1
Front -> 1
Items -> 2 3 4 5
Rear -> 4
Inserted -> 7
Front -> 1
Items -> 2 3 4 5 7
Rear -> 0
Queue is full!!
sjcet@HP-Z238:~/sandhramaria/cp$

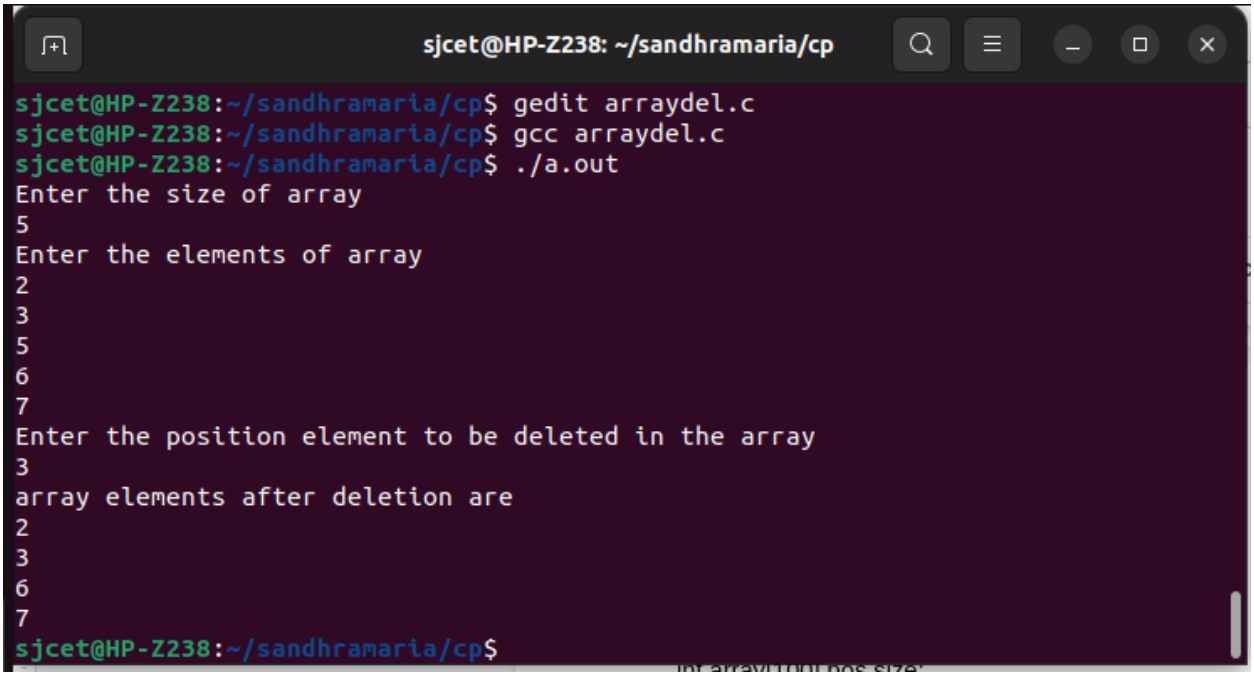
7. Array Insertion

```
#include <stdio.h>
void main(){
   int array[100], item, pos,size;
   printf("Enter the size of array\n");
   scanf("%d",&size);
   printf("Enter the elements of array\n");
   for(int i=0;i<size;i++)
      scanf("%d",&array[i]);
   printf("Enter the element to be inserted in the array\n");
   scanf("%d",&item);
   printf("Enter the position element to be inserted in the array\n");
   scanf("%d",&pos);
   size=size+1;
   for(int i=size-1;i>=pos;i--)
      array[i]=array[i-1];
   array[pos-1]=item;
   printf("new array elements are\n");
   for(int i=0;i<size;i++)
      printf("%d \n",array[i]);

}
```


sjcet@HP-Z238: ~/sandhramaria/cp
Items -> 2 3 4 5 7
Rear -> 0
Queue is full!!
sjcet@HP-Z238:~/sandhramaria/cp$ gcc arrayins.c
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
Enter the size of array
7
Enter the elements of array
1
2
4
5
6
7
8
Enter the element to be inserted in the array
3
Enter the position element to be inserted in the array
3
new array elements are
1
2
3
4
5
6
7
8
sjcet@HP-Z238:~/sandhramaria/cp$

8. Array deletion

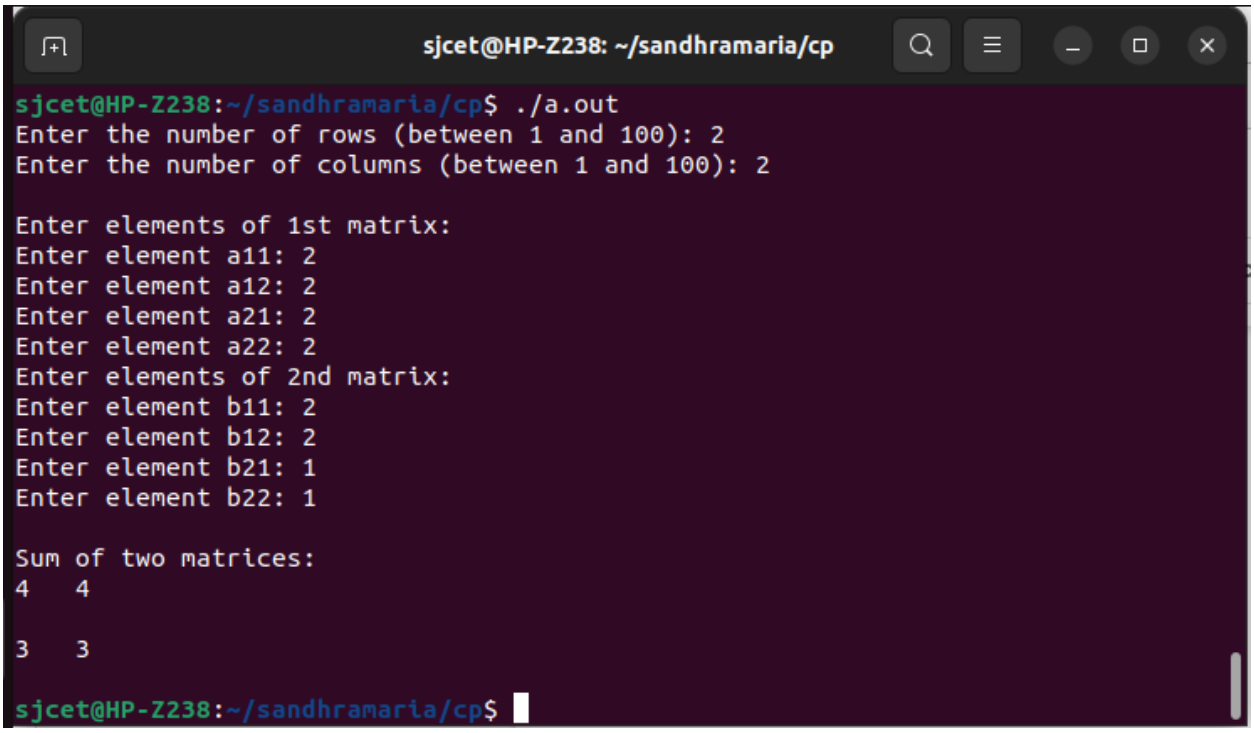
```
#include <stdio.h>
void main(){
int array[100],pos,size;
printf("Enter the size of array\n");
scanf("%d",&size);
printf("Enter the elements of array\n");
for(int i=0;i<size;i++)
scanf("%d",&array[i]);
printf("Enter the position element to be deleted in the array\n");
scanf("%d",&pos);
for(int i=pos-1;i<size-1;i++)
array[i]=array[i+1];
printf("array elements after deletion are\n");
for(int i=0;i<size-1;i++)
printf("%d \n",array[i]);
}
```


sjcet@HP-Z238: ~/sandhramaria/cp
sjcet@HP-Z238:~/sandhramaria/cp$ gedit arraydel.c
sjcet@HP-Z238:~/sandhramaria/cp$ gcc arraydel.c
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
Enter the size of array
5
Enter the elements of array
2
3
5
6
7
Enter the position element to be deleted in the array
3
array elements after deletion are
2
3
6
7
sjcet@HP-Z238:~/sandhramaria/cp$

9. Matrix Addition

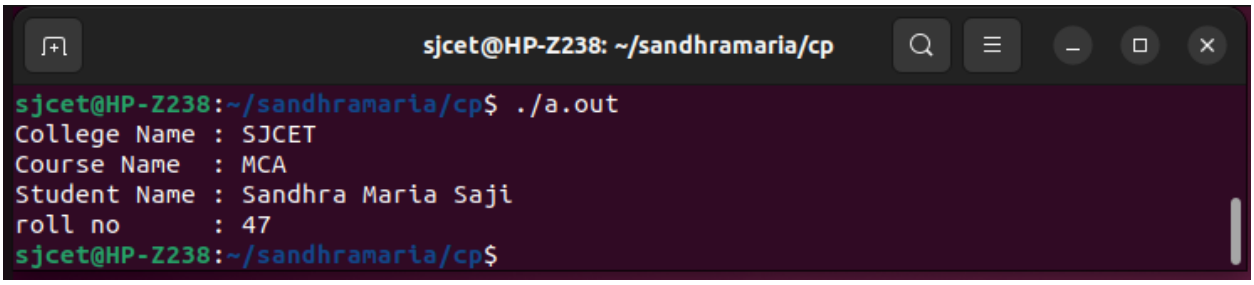
```
#include <stdio.h>
int main() {
  int r, c, a[100][100], b[100][100], sum[100][100], i, j;
  printf("Enter the number of rows (between 1 and 100): ");
  scanf("%d", &r);
  printf("Enter the number of columns (between 1 and 100): ");
  scanf("%d", &c);

  printf("\nEnter elements of 1st matrix:\n");
  for (i = 0; i < r; ++i)
    for (j = 0; j < c; ++j) {
      printf("Enter element a%d%d: ", i + 1, j + 1);
      scanf("%d", &a[i][j]);
    }

  printf("Enter elements of 2nd matrix:\n");
  for (i = 0; i < r; ++i)
    for (j = 0; j < c; ++j) {
      printf("Enter element b%d%d: ", i + 1, j + 1);
      scanf("%d", &b[i][j]);
    }


  for (i = 0; i < r; ++i)
    for (j = 0; j < c; ++j) {
      sum[i][j] = a[i][j] + b[i][j];
    }


  printf("\nSum of two matrices: \n");
  for (i = 0; i < r; ++i)
    for (j = 0; j < c; ++j) {
      printf("%d   ", sum[i][j]);
```

```
      if (j == c - 1) {
        printf("\n\n");
      }
    }

  return 0;
}
```



10. Structure Implimentation

```
#include<stdio.h>
#include<string.h>
struct student {
   int rollno;
   char name[20];
   char course[5];
};
struct college {
   char name1[7];
   struct student s1;
};
void main(){
   struct college c1;
   c1.s1.rollno=47;
   strcpy(c1.s1.name,"Sandhra Maria Saji”);
   strcpy(c1.s1.course,"MCA");
   strcpy(c1.name1,"SJCET");
   printf("College Name : %s\n",c1.name1);
   printf("Course Name  : %s\n",c1.s1.course);
   printf("Student Name : %s\n",c1.s1.name);
   printf("roll no      : %d\n",c1.s1.rollno);
}
```



11. Structure Implementation linear linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node *next;
};
struct node *head;
void insert_begin();
void insert_end();
void insert_middle();
void delete_begin();
void delete_end();
void delete_middle();
void display();
void search();

void main ()
{
int choice=0;
while(choice!=9)
{
printf("\n\nSelect your choice\n");
printf("\n1.Insert in Begining\n2.Insert at End\n3.Insert in between some
location\n4.Delete from Beginning\n5.Delete from End\n6.Delete node after
specified location\n7.Search for an element\n8.Display\n9.Exit\n");
printf("\nEnter your choice\n");
scanf("\n%d",&choice);
switch(choice)
{
case 1:
insert_begin();
break;
```

```
case 2:
insert_end();
break;
case 3:
insert_middle();
break;
case 4:
delete_begin();
break;
case 5:
delete_end();
break;
case 6:
delete_middle();
break;
case 7:
search();
break;
case 8:
display();
break;
case 9:
exit(0);
break;
default:
printf("Invalid Choice \n\n");
printf("NB:Please enter valid choice..");
}
}
}


void insert_begin()
{
 struct node*ptr;
```

```
 int item;
 ptr=(struct node*)malloc(sizeof(struct node*));
 if(ptr==NULL)
 {
  printf("\nOVERFLOW");
 }
 else
 {
 printf("\nEnter value\n");
 scanf("%d",&item);
 ptr->data = item;
 ptr->next = head;
 head = ptr;
 printf("\nNode inserted");
 }
}


void insert_end()
{
 struct node *ptr,*temp;
 int item;
 ptr = (struct node*)malloc(sizeof(struct node));
 if(ptr == NULL)
 {
  printf("\nOVERFLOW");
 }
 else
 {
  printf("\nEnter value?\n");
  scanf("%d",&item);
  ptr->data = item;
  if(head == NULL)
  {
   ptr -> next = NULL;
```

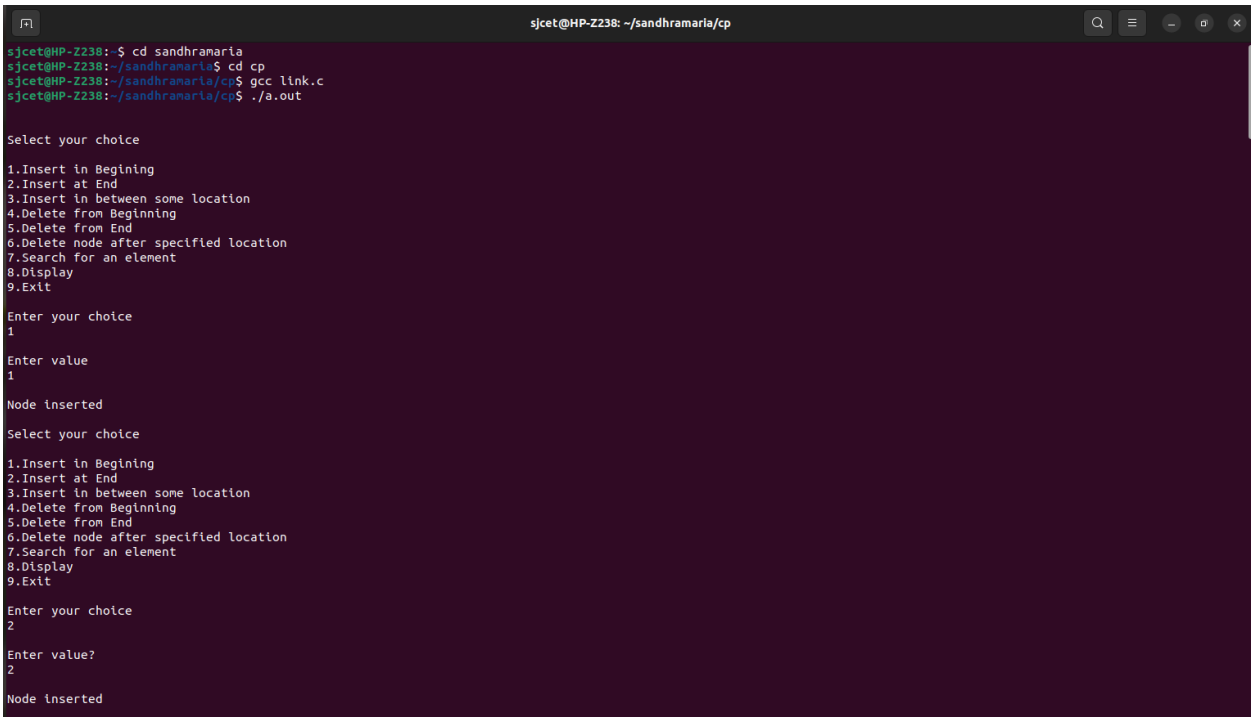
```
   head = ptr;
   printf("\nNode inserted");
  }
  else
  {
   temp = head;
   while (temp -> next != NULL)
   {
    temp = temp -> next;
   }
   temp->next = ptr;
   ptr->next = NULL;
   printf("\nNode inserted");
  }
 }
}

void  insert_middle()
{
int i,loc,item;
struct node *ptr, *temp;
ptr = (struct node *) malloc (sizeof(struct node));
if(ptr == NULL)
{
printf("\nOVERFLOW");
}
else
{
printf("\nEnter element value");
scanf("%d",&item);
ptr->data = item;
printf("\nEnter the location after which you want to insert ");
scanf("%d",&loc);
temp=head;
for(i=0;i<loc-2;i++)
```

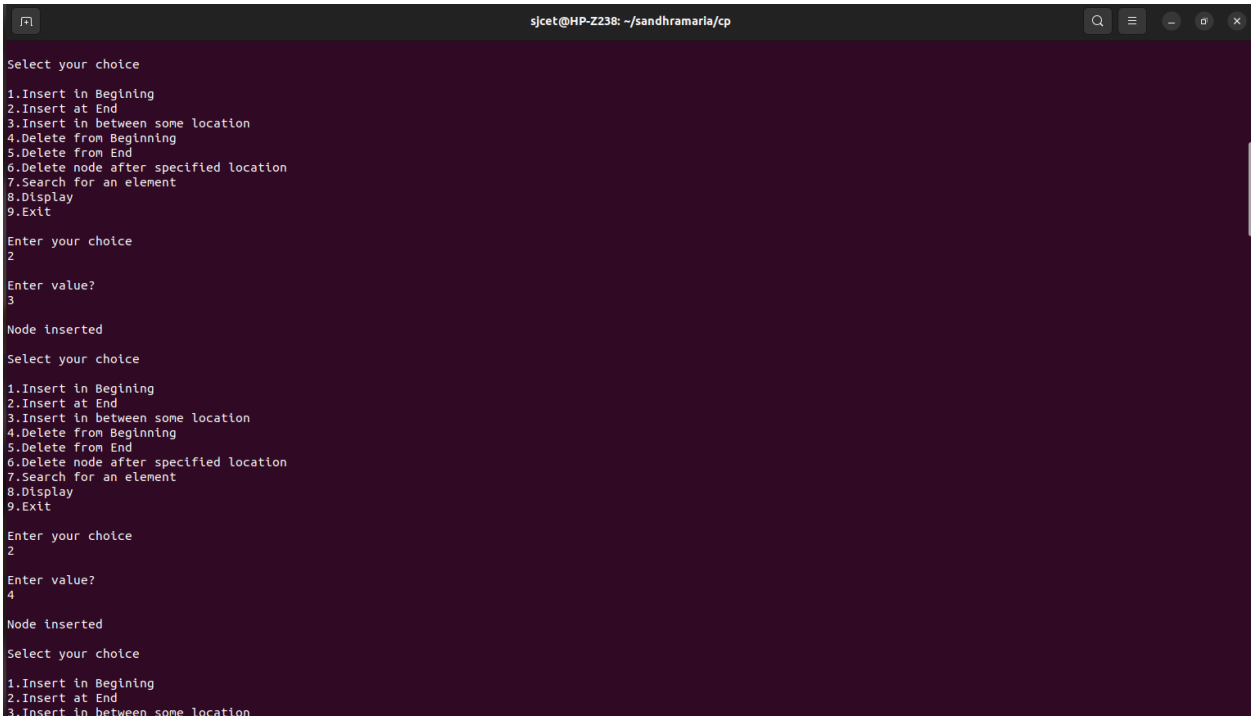
```
{
temp = temp->next;
if(temp == NULL)
 {
printf("\ncan't insert\n");
return;
}
}
ptr ->next = temp ->next;
temp ->next = ptr;
printf("\nNode inserted");
}
}


void delete_begin()
{
struct node *ptr;
if(head == NULL)
{
printf("\nList is empty\n");
}
else
{
ptr = head;
head = ptr->next;
free(ptr);
printf("\nNode deleted from the begining\n");
}
}


void delete_end()
{
struct node *ptr,*ptr1;
```

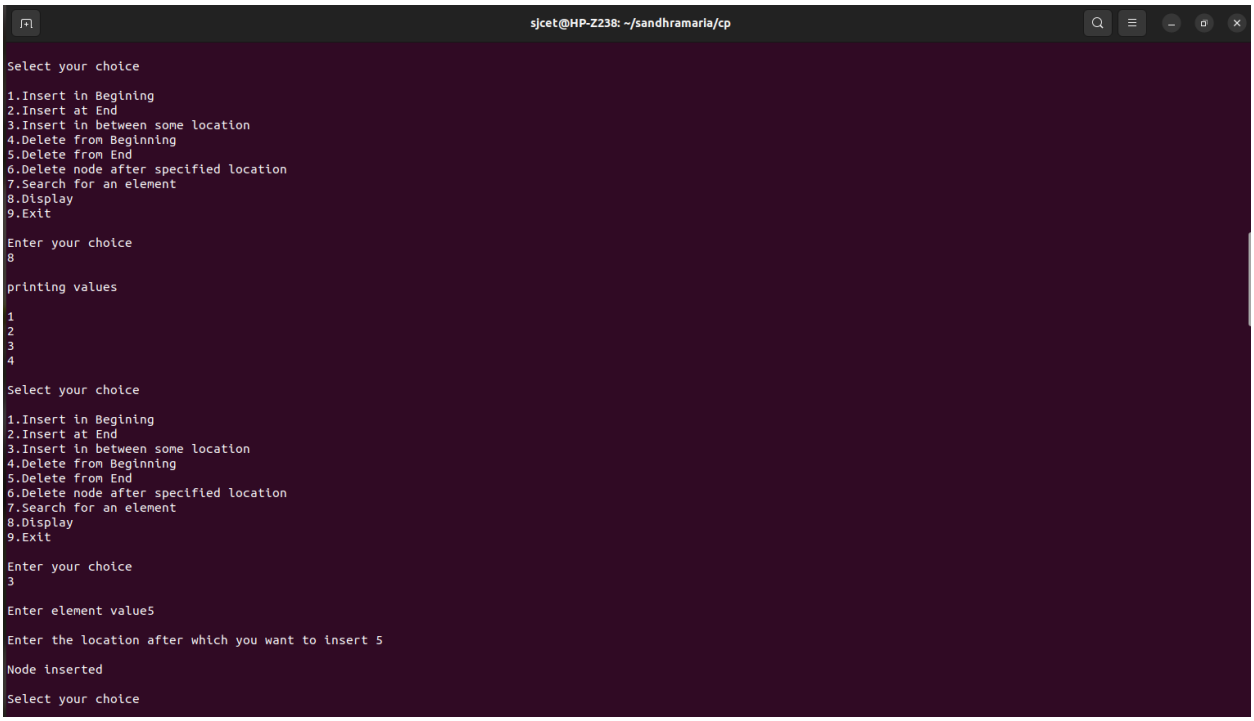
```
if(head == NULL)
{
printf("\nlist is empty");
}
else if(head -> next == NULL)
{
head = NULL;
free(head);
printf("\nThe single node of the list deleted\n");
}
else
{
ptr = head;
while(ptr->next != NULL)
{
ptr1 = ptr;
ptr = ptr ->next;
}
ptr1->next = NULL;
free(ptr);
printf("\nDeleted Node from the last\n");
}
}


void delete_middle()
{
struct node *ptr,*ptr1;
int loc,i;
printf("\n Enter the location of the node after which you want to perform deletion
\n");
scanf("%d",&loc);
ptr=head;
for(i=0;i<loc;i++)
{
```

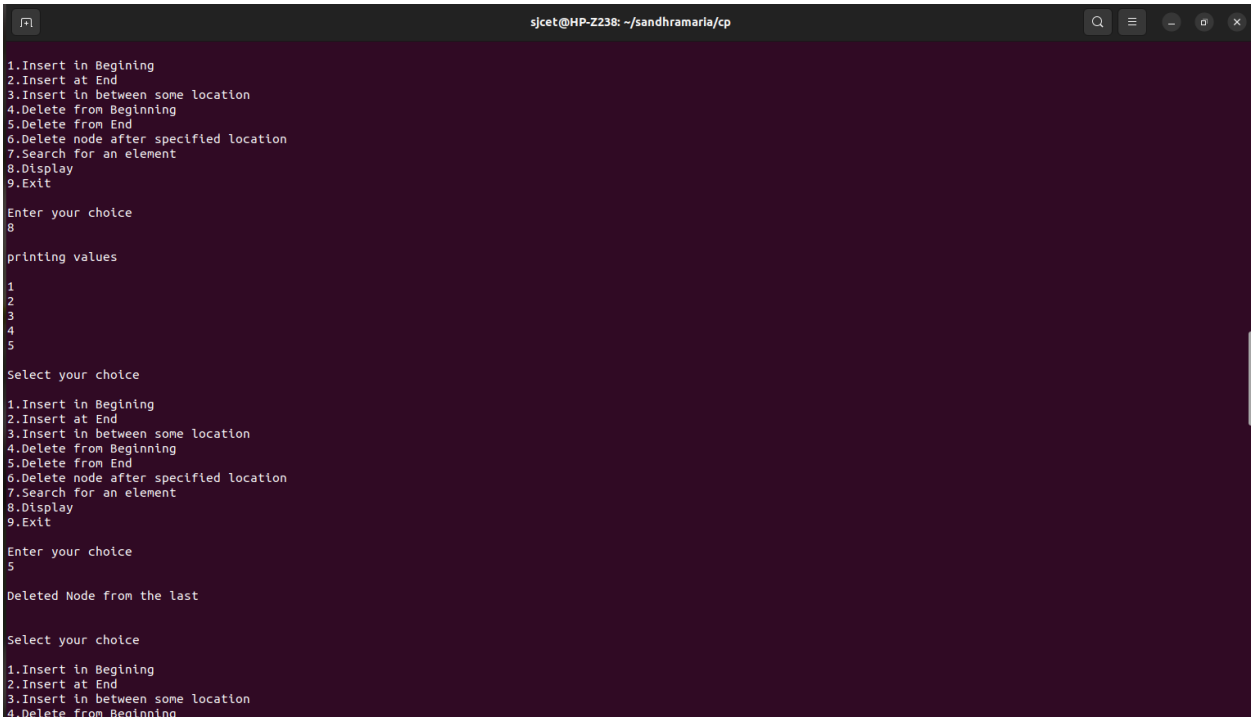
```
ptr1 = ptr;
ptr = ptr->next;
if(ptr == NULL)
{
printf("\nNo element, can't delete");
return;
}
}
ptr1 ->next = ptr ->next;
free(ptr);
printf("\nDeleted node %d ",loc+1);
}


void search()
{
struct node *ptr;
int item,i=0,flag;
ptr = head;
if(ptr == NULL)
{
printf("\nEmpty List\n");
}
else
{
printf("\nEnter item which you want to search?\n");
scanf("%d",&item);
while (ptr!=NULL)
{
if(ptr->data == item)
{
printf("item found at location %d ",i+1);
flag=0;
}
else
```

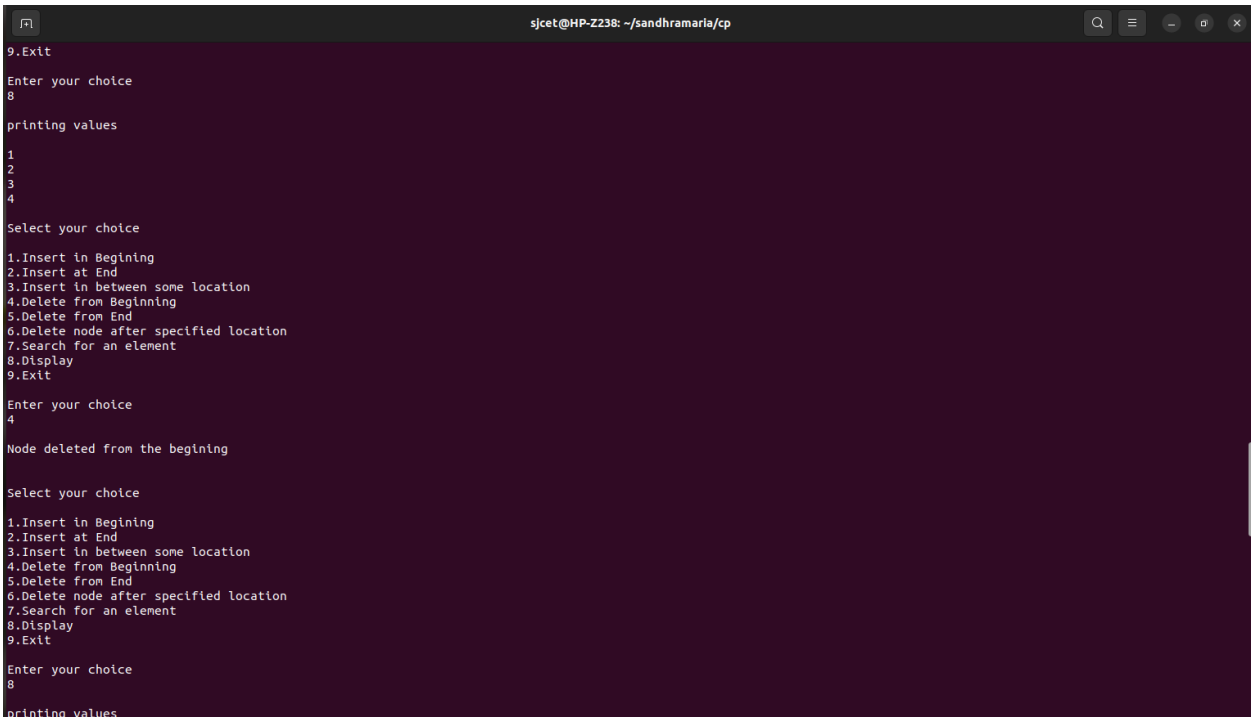
```
{
flag=1;
}
i++;
ptr = ptr -> next;
}
if(flag==1)
{
printf("Item not found\n");
}
}
}


void display()
{
struct node *ptr;
ptr = head;
if(ptr == NULL)
{
printf("Nothing to print, Empty list");
}
else
{
printf("\nprinting values\n");
while (ptr!=NULL)
{
printf("\n%d",ptr->data);
ptr = ptr -> next;
}
}
}
```

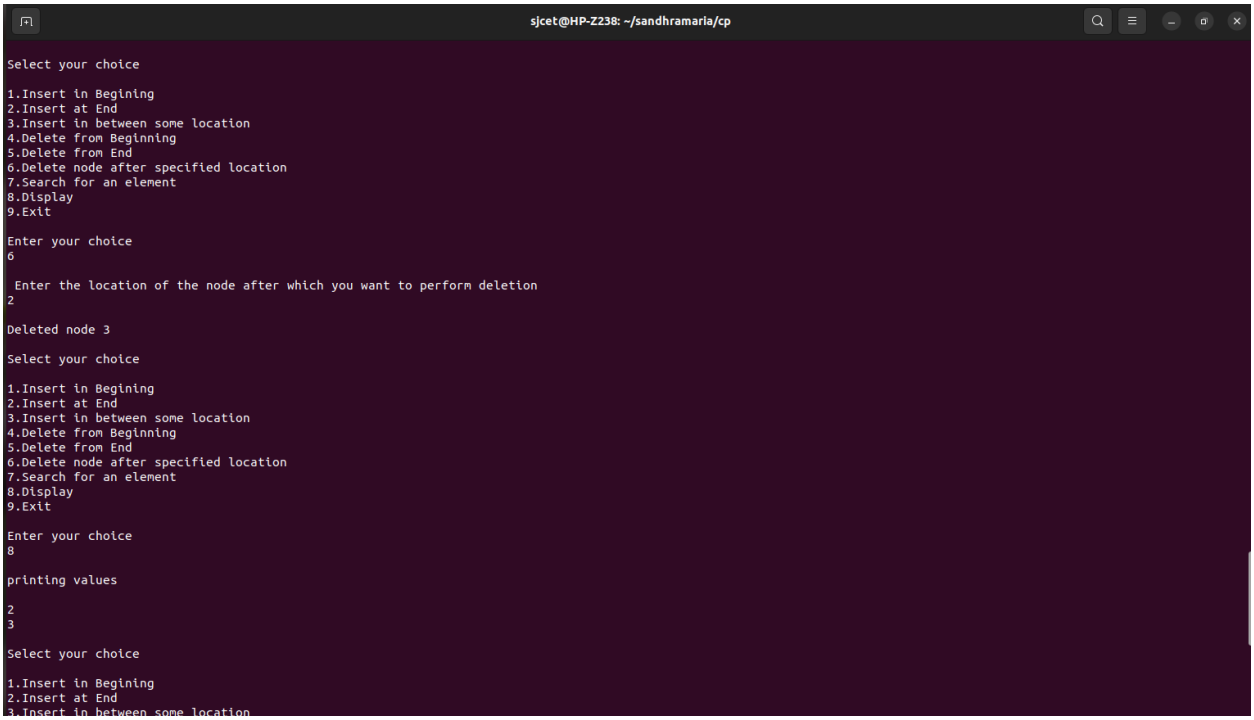

sjcet@HP-Z238: ~/sandhramaria/cp
sjcet@HP-Z238:~$ cd sandhramaria
sjcet@HP-Z238:~/sandhramaria$ cd cp
sjcet@HP-Z238:~/sandhramaria/cp$ gcc link.c
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
1
Enter value
1
Node inserted
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
2
Enter value?
2
Node inserted

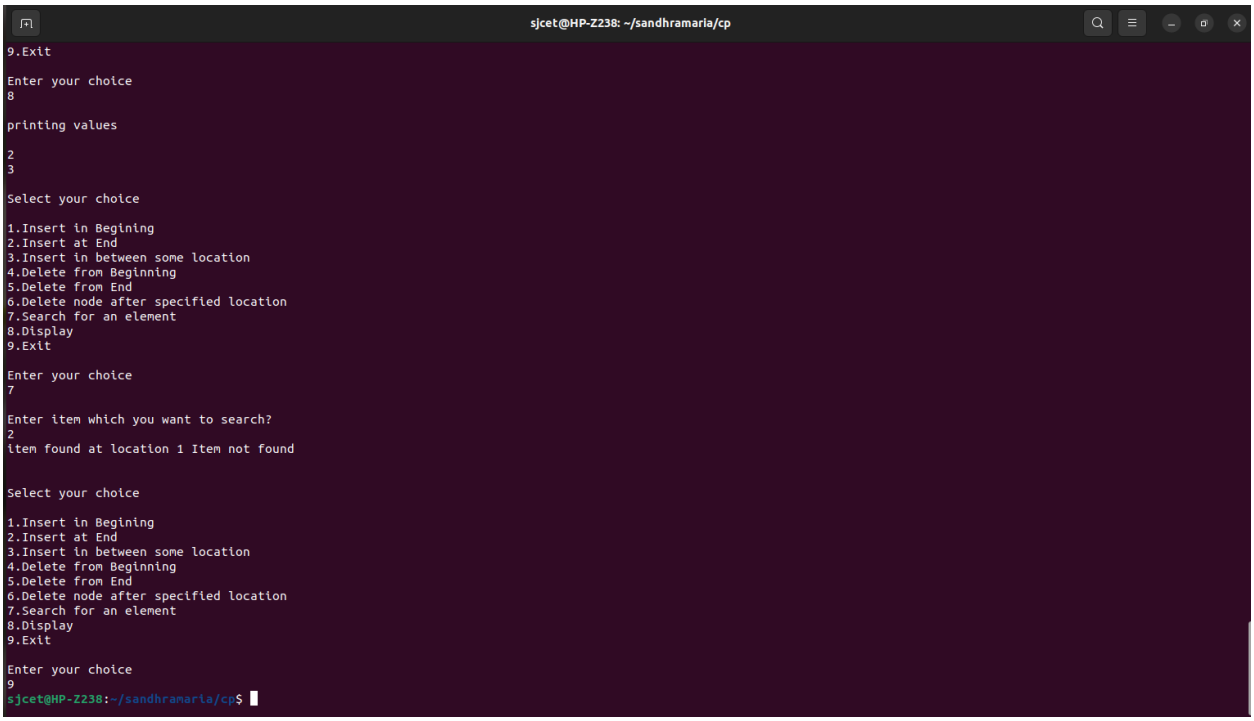

sjcet@HP-Z238: ~/sandhramaria/cp
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
2
Enter value?
3
Node inserted
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
2
Enter value?
4
Node inserted
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location


sjcet@HP-Z238: ~/sandhramaria/cp
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
8
printing values
1
2
3
4
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
3
Enter element value5
Enter the location after which you want to insert 5
Node inserted
Select your choice


sjcet@HP-Z238: ~/sandhramaria/cp
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
8
printing values
1
2
3
4
5
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
5
Deleted Node from the last
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning


sjcet@HP-Z238: ~/sandhramaria/cp
9.Exit
Enter your choice
8
printing values
1
2
3
4
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
4
Node deleted from the begining
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
8
printing values


sjcet@HP-Z238: ~/sandhramaria/cp
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
6
Enter the location of the node after which you want to perform deletion
2
Deleted node 3
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
8
printing values
2
3
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location


sjcet@HP-Z238: ~/sandhramaria/cp
9.Exit
Enter your choice
8
printing values
2
3
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
7
Enter item which you want to search?
2
item found at location 1 Item not found
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
9
sjcet@HP-Z238:~/sandhramaria/cp$

12. Implementation doubly linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
struct node *prev;
struct node *next;
int data;
};
struct node *head;
void insert_begin();
void insert_end();
void insert_middle();
void delete_begin();
void delete_end();
void delete_middle();
void display();
void search();

void main ()
{
int choice =0;
while(choice != 9)
{
printf("\n\nSelect your choice\n");
printf("\n1.Insert in Begining\n2.Insert at End\n3.Insert in between some
location\n4.Delete from Beginning\n5.Delete from End\n6.Delete node after
specified location\n7.Search for an element\n8.Display\n9.Exit\n");
printf("\nEnter your choice\n");
scanf("\n%d",&choice);
switch(choice)
{
case 1:
insert_begin();
```

```
break;
case 2:
insert_end();
break;
case 3:
insert_middle();
break;
case 4:
delete_begin();
break;
case 5:
delete_end();
break;
case 6:
delete_middle();
break;
case 7:
search();
break;
case 8:
display();
break;
case 9:
exit(0);
break;
default:
printf("\nInvalid Choice \n");
printf("NB:Please enter valid choice..");
}
}
}


void insert_begin()
{
```

```
struct node *ptr;
int item;
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
{
printf("\nOVERFLOW");
}
else
{
printf("\nEnter Item value");
scanf("%d",&item);
if(head==NULL)
{
ptr->next = NULL;
ptr->prev=NULL;
ptr->data=item;
head=ptr;
}
else
{
ptr->data=item;
ptr->prev=NULL;
ptr->next = head;
head->prev=ptr;
head=ptr;
}
printf("\nNode  inserted\n");
}
}


void insert_end()
{
struct node *ptr,*temp;
int item;
```

```
ptr = (struct node *) malloc(sizeof(struct node));
if(ptr == NULL)
{
printf("\nOVERFLOW");
}
else
{
printf("\nEnter value");
scanf("%d",&item);
ptr->data=item;
if(head == NULL)
{
ptr->next = NULL;
ptr->prev = NULL;
head = ptr;
}
else
{
temp = head;
while(temp->next!=NULL)
{
temp = temp->next;
}
temp->next = ptr;
ptr ->prev=temp;
ptr->next = NULL;
}
}
printf("\nnode inserted\n");
}


void insert_middle()
{
struct node *ptr,*temp;
```

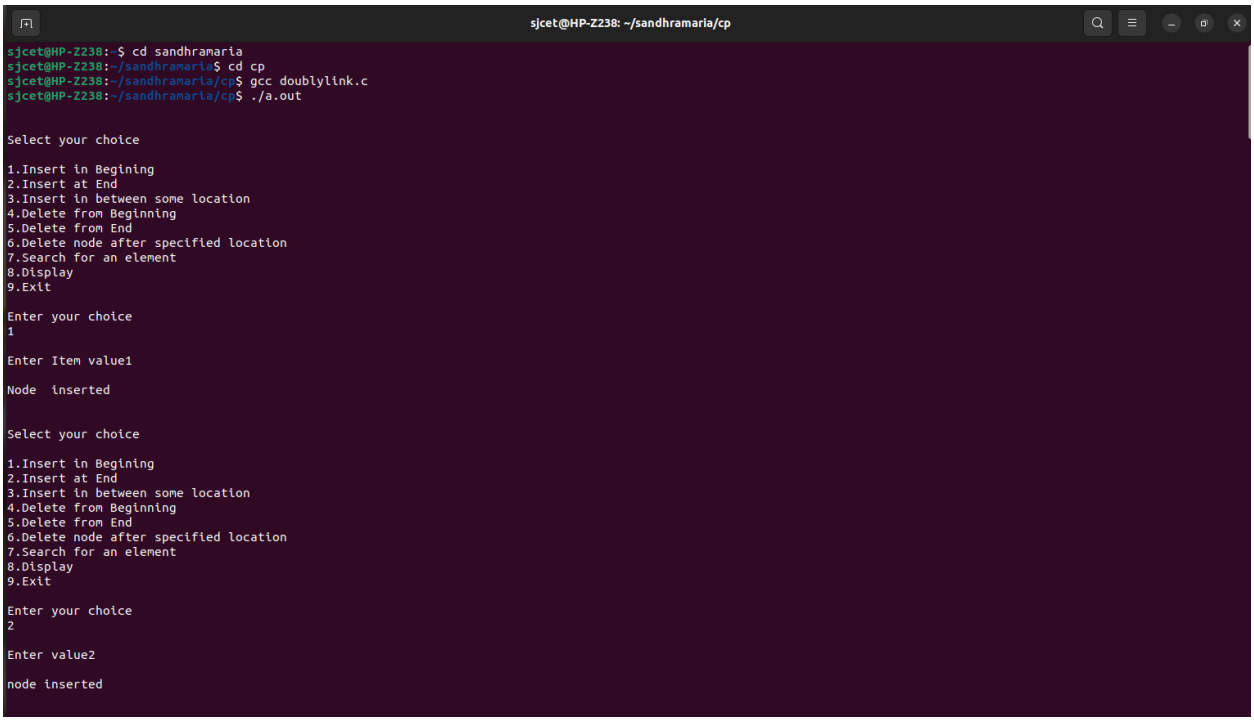
```
int item,loc,i;
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
{
printf("\n OVERFLOW");
}
else
{
temp=head;
printf("Enter the location");
scanf("%d",&loc);
for(i=0;i<loc-2;i++)
{
temp = temp->next;
if(temp == NULL)
{
printf("\n There are less than %d elements", loc);
return;
}
}
printf("Enter value");
scanf("%d",&item);
ptr->data = item;
ptr->next = temp->next;
ptr -> prev = temp;
temp->next = ptr;
temp->next->prev=ptr;
printf("\nnode inserted\n");
}
}


void delete_begin()
{
struct node *ptr;
```

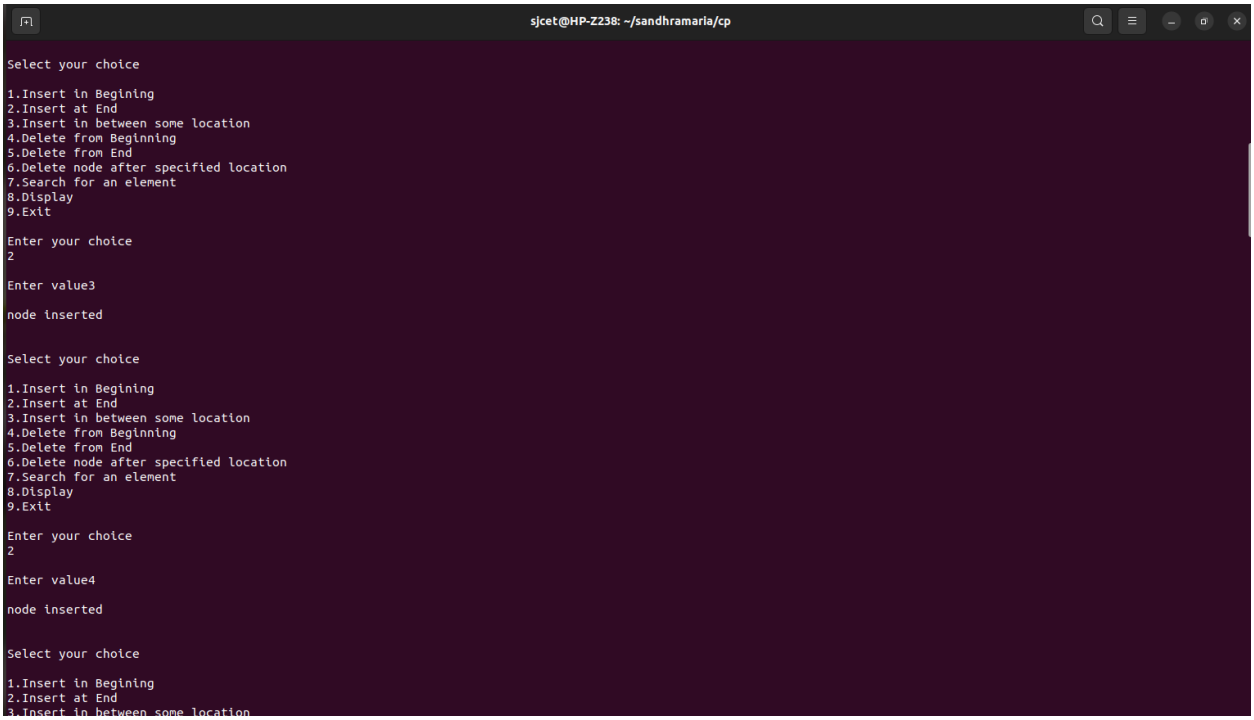
```
if(head == NULL)
{
printf("\n UNDERFLOW");
}
else if(head->next == NULL)
{
head = NULL;
free(head);
printf("\nnode deleted\n");
}
else
{
ptr = head;
head = head -> next;
head -> prev = NULL;
free(ptr);
printf("\nnode deleted\n");
}
}


void delete_end()
{
struct node *ptr;
if(head == NULL)
{
printf("\n UNDERFLOW");
}
else if(head->next == NULL)
{
head = NULL;
free(head);
printf("\nNode deleted\n");
}
else
```

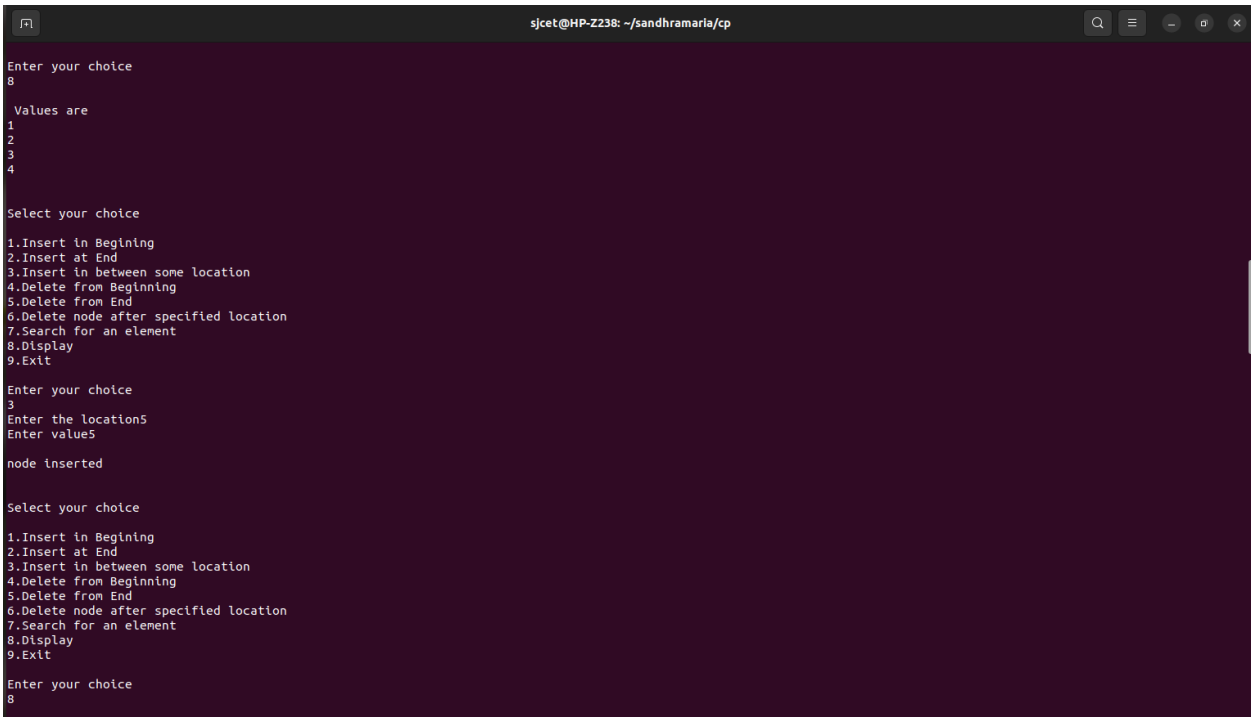
```
{
ptr = head;
if(ptr->next != NULL)
{
ptr=ptr->next;
}
ptr->prev->next=NULL;
free(ptr);
printf("\nNode deleted\n");
}
}


void delete_middle()
{
struct node *ptr, *temp;
int val;
printf("\n Enter the data after which the node is to be deleted : ");
scanf("%d", &val);
ptr = head;
while(ptr->data!=val)
ptr=ptr->next;
if(ptr->next==NULL)
{
printf("\nCan't delete\n");
}
else if(ptr->next->next==NULL)
{
ptr->next=NULL;
}
else
{
temp = ptr->next;
ptr->next=temp->next;
temp -> next -> prev = ptr;
```

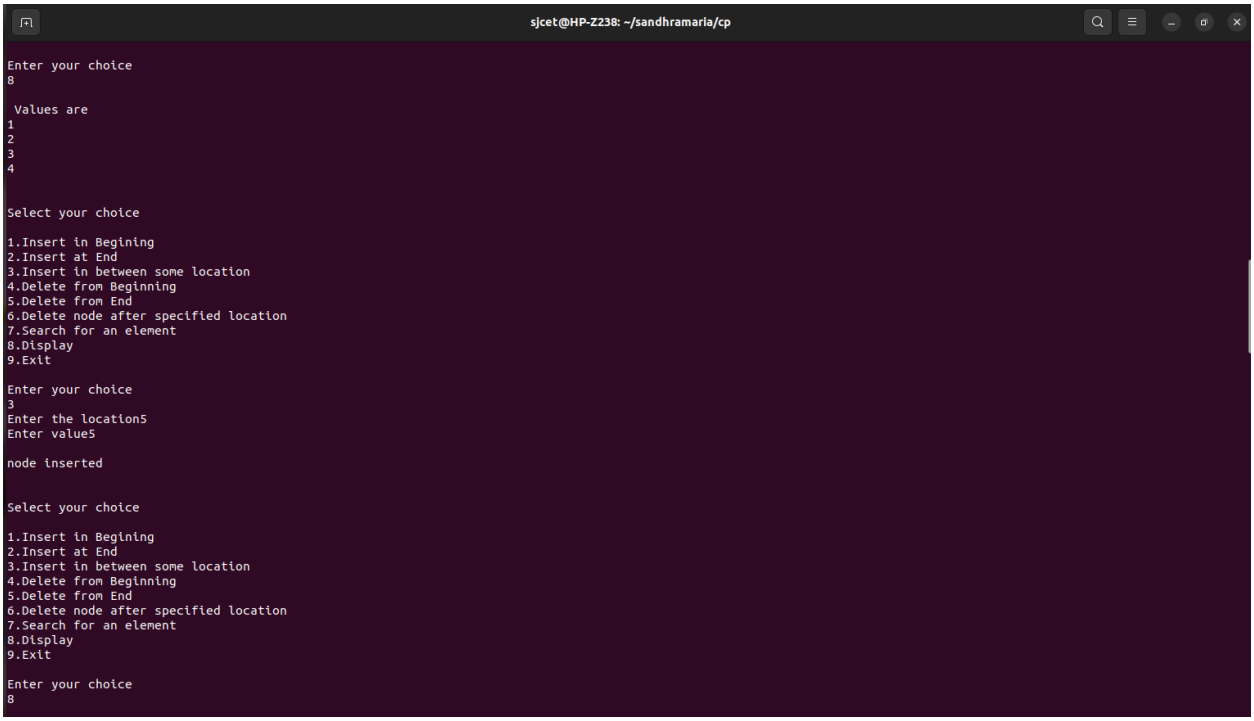
```
free(temp);
printf("\nNode deleted\n");
}
}


void display()
{
struct node *ptr;
if(head==NULL)
{
printf("\n UNDERFLOW, Empty list");
}
else
{
printf("\n Values are\n");
ptr = head;
while(ptr!= NULL)
{
printf("%d\n",ptr->data);
ptr=ptr->next;
}
}
}


void search()
{
struct node *ptr;
int item,i=0,flag;
ptr = head;
if(ptr==NULL)
{
printf("\nEmpty List\n");
}
```

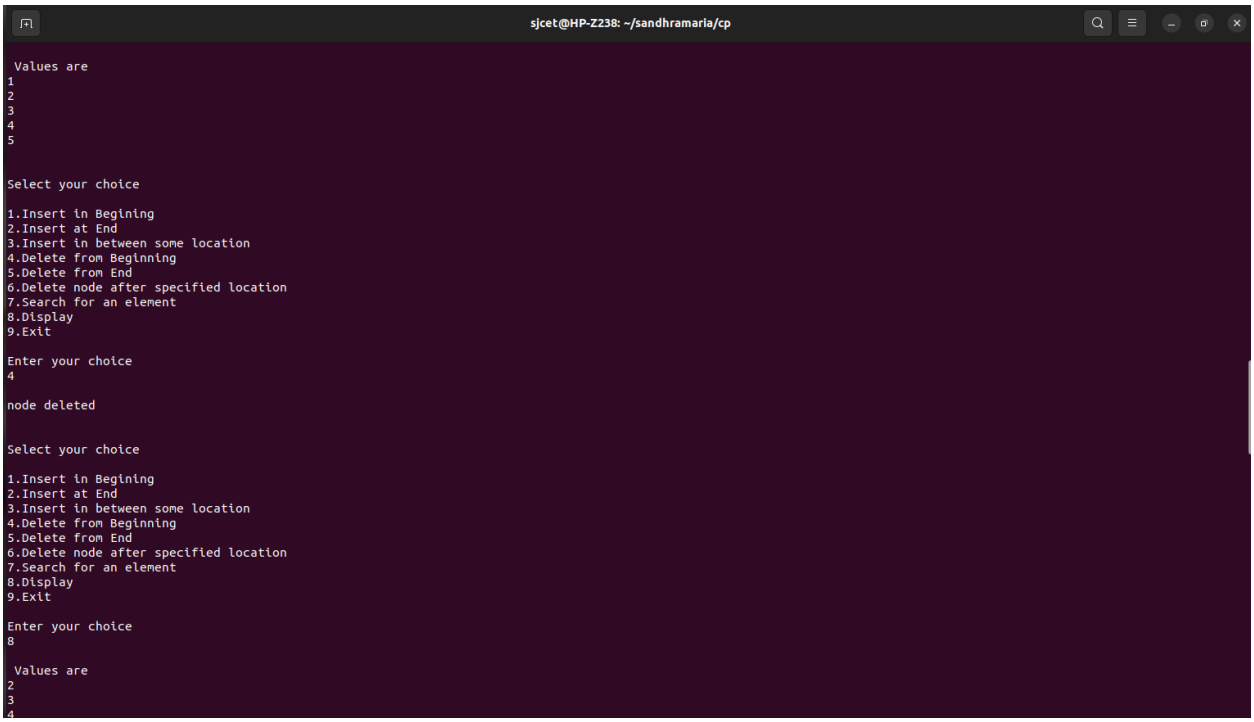
```
else
{
printf("\nEnter item which you want to search?\n");
scanf("%d",&item);
while (ptr!=NULL)
{
if(ptr->data==item)
{
printf("\nitem found at location %d ",i+1);
flag=0;
break;
}
else
{
flag=1;
}
i++;
ptr=ptr->next;
}
if(flag==1)
{
printf("\nItem not found\n");
}
}
}
```

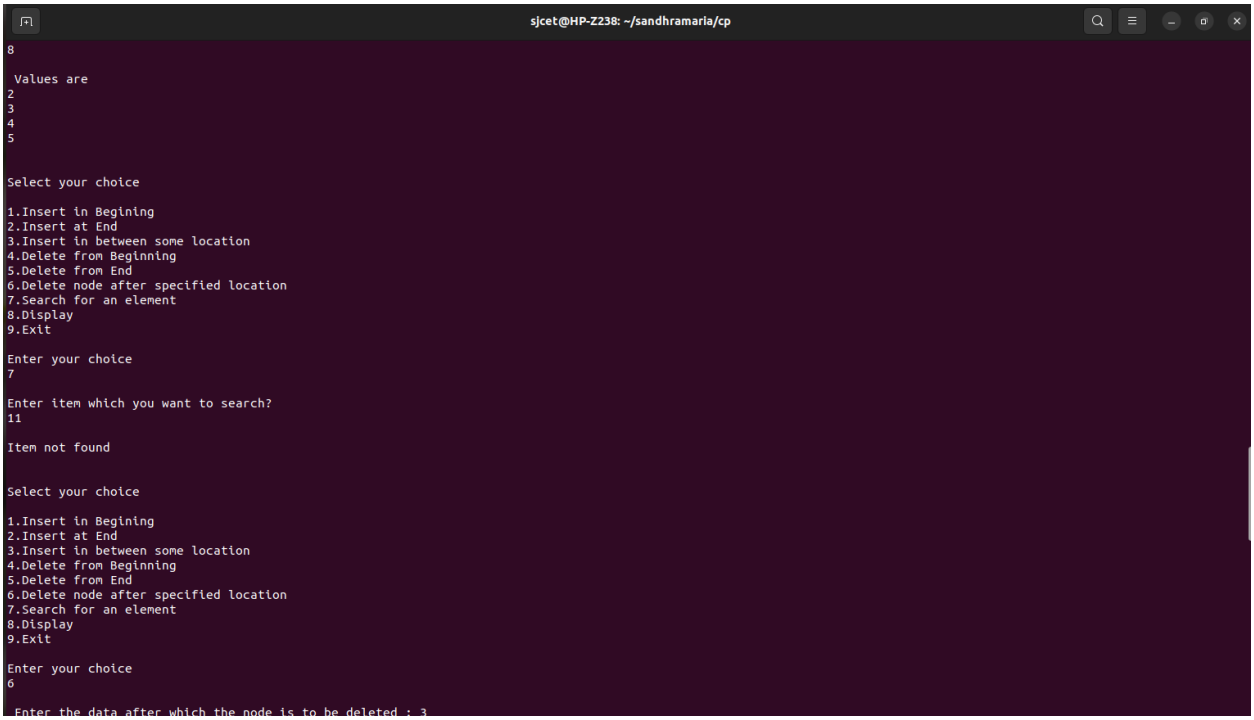

sjcet@HP-Z238: ~/sandhramaria/cp
sjcet@HP-Z238:~$ cd sandhramaria
sjcet@HP-Z238:~/sandhramaria$ cd cp
sjcet@HP-Z238:~/sandhramaria/cp$ gcc doublylink.c
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
1
Enter Item value1
Node inserted
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
2
Enter value2
node inserted

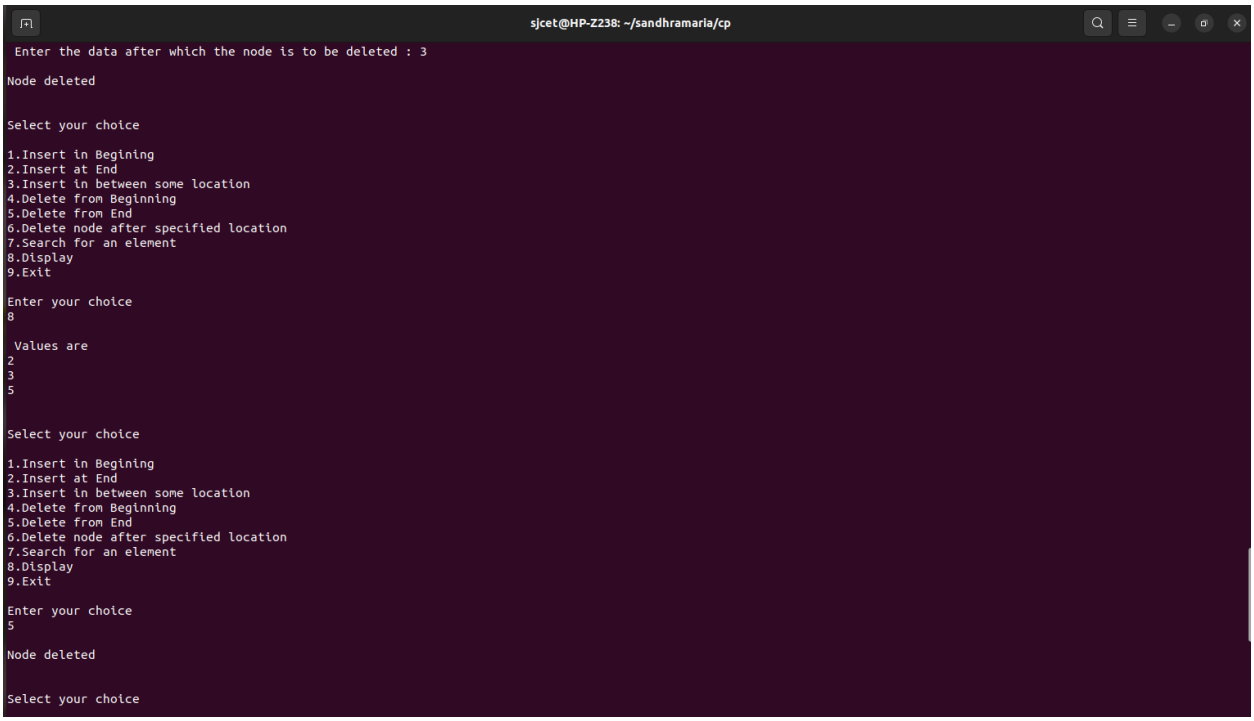

sjcet@HP-Z238: ~/sandhramaria/cp
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
2
Enter value3
node inserted
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
2
Enter value4
node inserted
Select your choice
1.Insert in Begining
2.Insert at End

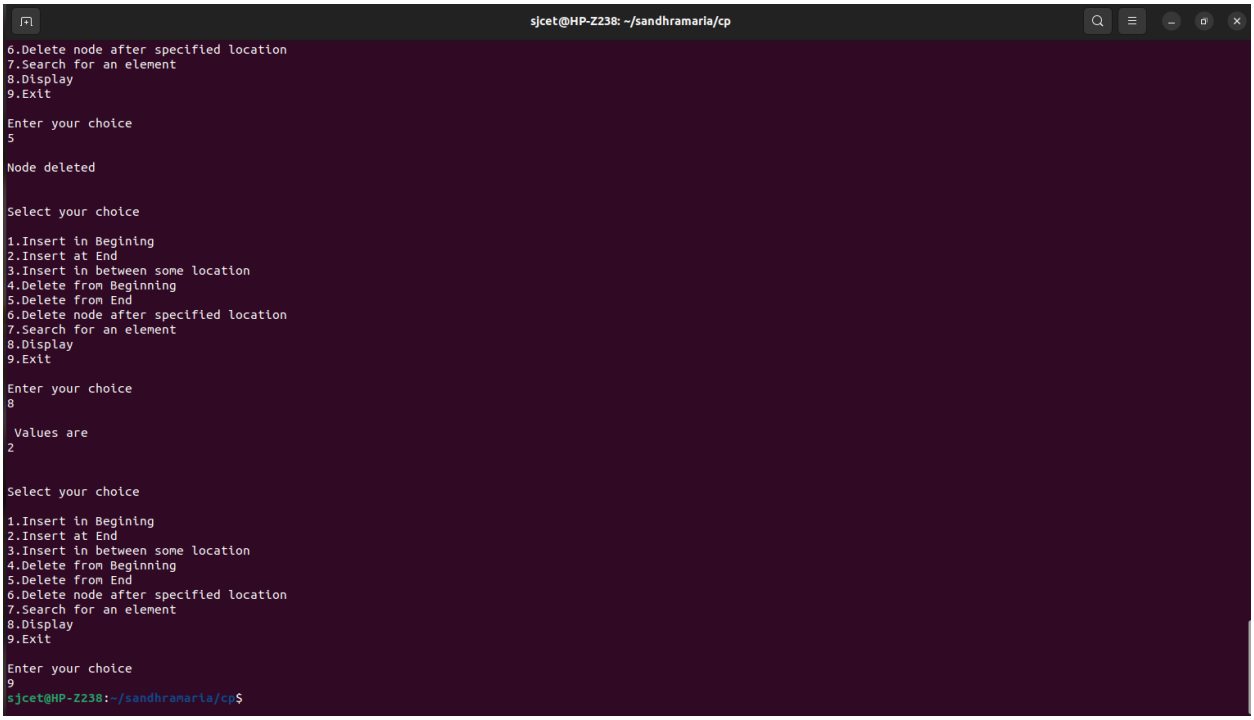

sjcet@HP-Z238: ~/sandhramaria/cp
Enter your choice
8
Values are
1
2
3
4
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
3
Enter the location5
Enter value5
node inserted
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
8


sjcet@HP-Z238: ~/sandhramaria/cp
Enter your choice
8
Values are
1
2
3
4
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
3
Enter the location5
Enter value5
node inserted
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
8


sjcet@HP-Z238: ~/sandhramaria/cp
Values are
1
2
3
4
5
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
4
node deleted
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
8
Values are
2
3
4


sjcet@HP-Z238: ~/sandhramaria/cp
8
Values are
2
3
4
5
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
7
Enter item which you want to search?
11
Item not found
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
6
Enter the data after which the node is to be deleted : 3


sjcet@HP-Z238: ~/sandhramaria/cp
Enter the data after which the node is to be deleted : 3
Node deleted
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
8
Values are
2
3
5
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
5
Node deleted
Select your choice


sjcet@HP-Z238: ~/sandhramaria/cp
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
5
Node deleted
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
8
Values are
2
Select your choice
1.Insert in Begining
2.Insert at End
3.Insert in between some location
4.Delete from Beginning
5.Delete from End
6.Delete node after specified location
7.Search for an element
8.Display
9.Exit
Enter your choice
9
sjcet@HP-Z238:~/sandhramaria/cp$

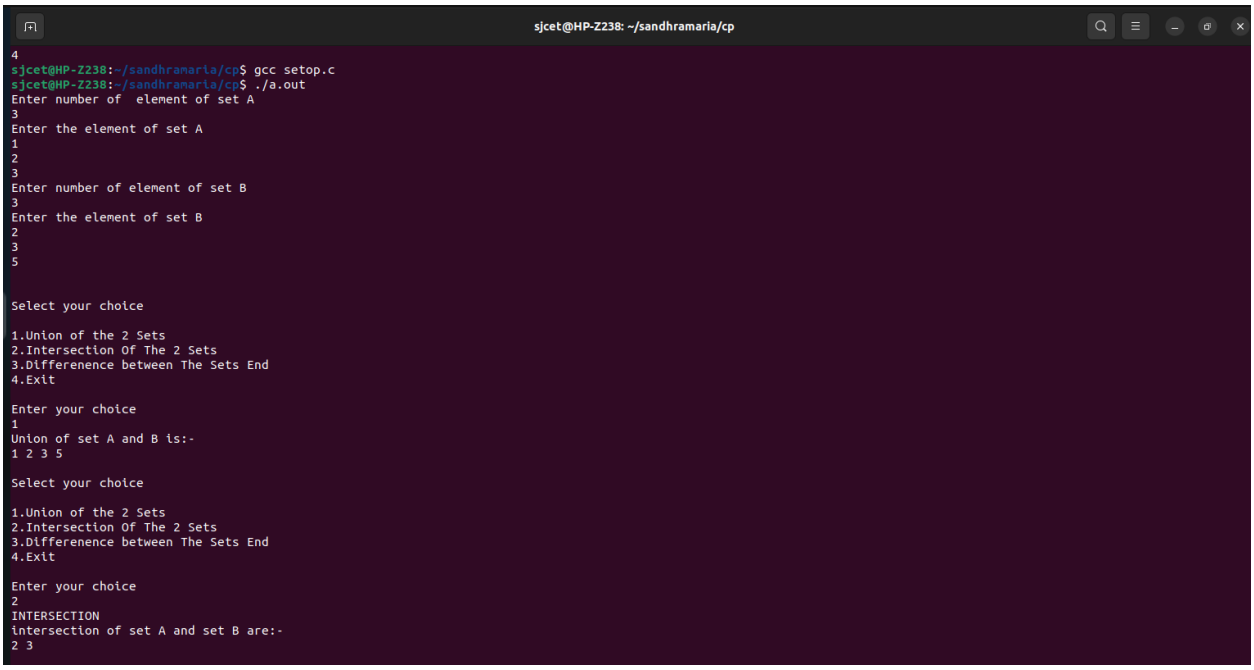
13.Set operations - Union, Intersection, Difference

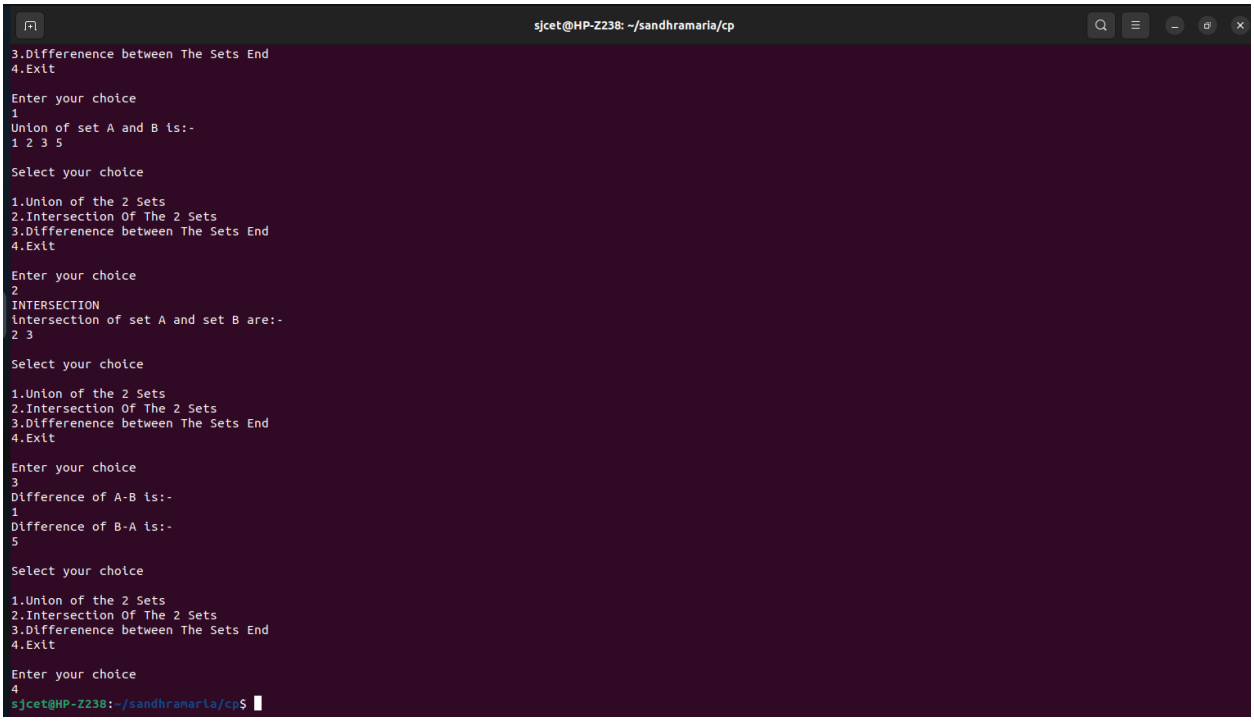
```
#include<stdio.h>
void main()
{
   int a[10],b[10],i,c[10],d[10],e[10],f[10],j,k=0,n1,l,n2,ch,m=0,n=0,p=0;
   printf("Enter number of  element of set A\n");
   scanf("%d",&n1);
   printf("Enter the element of set A \n");
   for(i=0;i<n1;i++)
      scanf("%d",&a[i]);
   printf("Enter number of element of set B\n");
   scanf("%d",&n2);
   printf("Enter the element of set B \n");
   for(i=0;i<n2;i++)
     scanf("%d",&b[i]);

   while(ch!=4)
{
printf("\n\nSelect your choice\n");
printf("\n1.Union of the 2 Sets \n2.Intersection Of The 2 Sets\n3.Differenence
between The Sets\n4.Exit\n");
printf("\nEnter your choice\n");
scanf("\n%d",&ch);
switch(ch)
{
case 1:
  for(i=0;i<n1;i++)
  {
     for(j=0;j<k;j++)
     {
if(c[j]==a[i])
   break;
      }
```

```
      if(j==k)
      {
 c[k]=a[i];
 k++;
      }
   }
  for(i=0;i<n2;i++)
  {
     for(j=0;j<k;j++)
     {
if(c[j]==b[i])
  break;
     }
    if(j==k)
    {
      c[k]=b[i];
      k++;
    }
  }
  printf("Union of set A and B is:-\n");
   for(i=0;i<k;i++)
     printf("%d ",c[i]);
break;


case 2:
printf("INTERSECTION \n");
          for( i=0;i<n1;i++)
          {
                    for(j=0;j<n2;j++)
                    {
                              if(a[i]==b[j])
                              {
                                        d[n]=a[i];
                                        n++;
```

```
                    }
                }

        }
        printf("intersection of set A and set B are:-\n");
        for(i=0;i<n;i++)
        printf("%d ",d[i]);
        break;

case 3:

    for( i=0;i<n1;i++)
    {
        for(j=0;j<n2;j++)
        {
            if(b[j]==a[i])
             break;
        }
        if(j==n2)
        {
            for(l=0;l<m;l++)
            {
                if(e[l]==a[i])
                 break;
            }
            if(l==m)
            {
                e[m]=a[i];
                m++;
            }
        }

    }

    for( i=0;i<n2;i++)
```

```
    {
       for(j=0;j<n1;j++)
       {
          if(b[i]==a[j])
           break;
       }
       if(j==n1)
       {

          for(l=0;l<p;l++)
          {
             if(d[l]==b[i])
              break;
          }
          if(l==p)
          {
             d[p]=b[i];
             p++;
          }
       }
    }
     printf("Difference of A-B is:-\n");
     for(i=0;i<m;i++)
     {
        printf("%d ",e[i]);
     }
     printf("\n");
     printf("Difference of B-A is:-\n");
     for(i=0;i<p;i++)
     {
        printf("%d ",d[i]);
     }
   }
}
}
```


sjcet@HP-Z238: ~/sandhramaria/cp
4
sjcet@HP-Z238:~/sandhramaria/cp$ gcc setop.c
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
Enter number of element of set A
3
Enter the element of set A
1
2
3
Enter number of element of set B
3
Enter the element of set B
2
3
5
Select your choice
1.Union of the 2 Sets
2.Intersection Of The 2 Sets
3.Differenence between The Sets End
4.Exit
Enter your choice
1
Union of set A and B is:-
1 2 3 5
Select your choice
1.Union of the 2 Sets
2.Intersection Of The 2 Sets
3.Differenence between The Sets End
4.Exit
Enter your choice
2
INTERSECTION
intersection of set A and set B are:-
2 3


sjcet@HP-Z238: ~/sandhramaria/cp
3.Differenence between The Sets End
4.Exit
Enter your choice
1
Union of set A and B is:-
1 2 3 5
Select your choice
1.Union of the 2 Sets
2.Intersection Of The 2 Sets
3.Differenence between The Sets End
4.Exit
Enter your choice
2
INTERSECTION
intersection of set A and set B are:-
2 3
Select your choice
1.Union of the 2 Sets
2.Intersection Of The 2 Sets
3.Differenence between The Sets End
4.Exit
Enter your choice
3
Difference of A-B is:-
1
Difference of B-A is:-
5
Select your choice
1.Union of the 2 Sets
2.Intersection Of The 2 Sets
3.Differenence between The Sets End
4.Exit
Enter your choice
4
sjcet@HP-Z238:~/sandhramaria/cp$

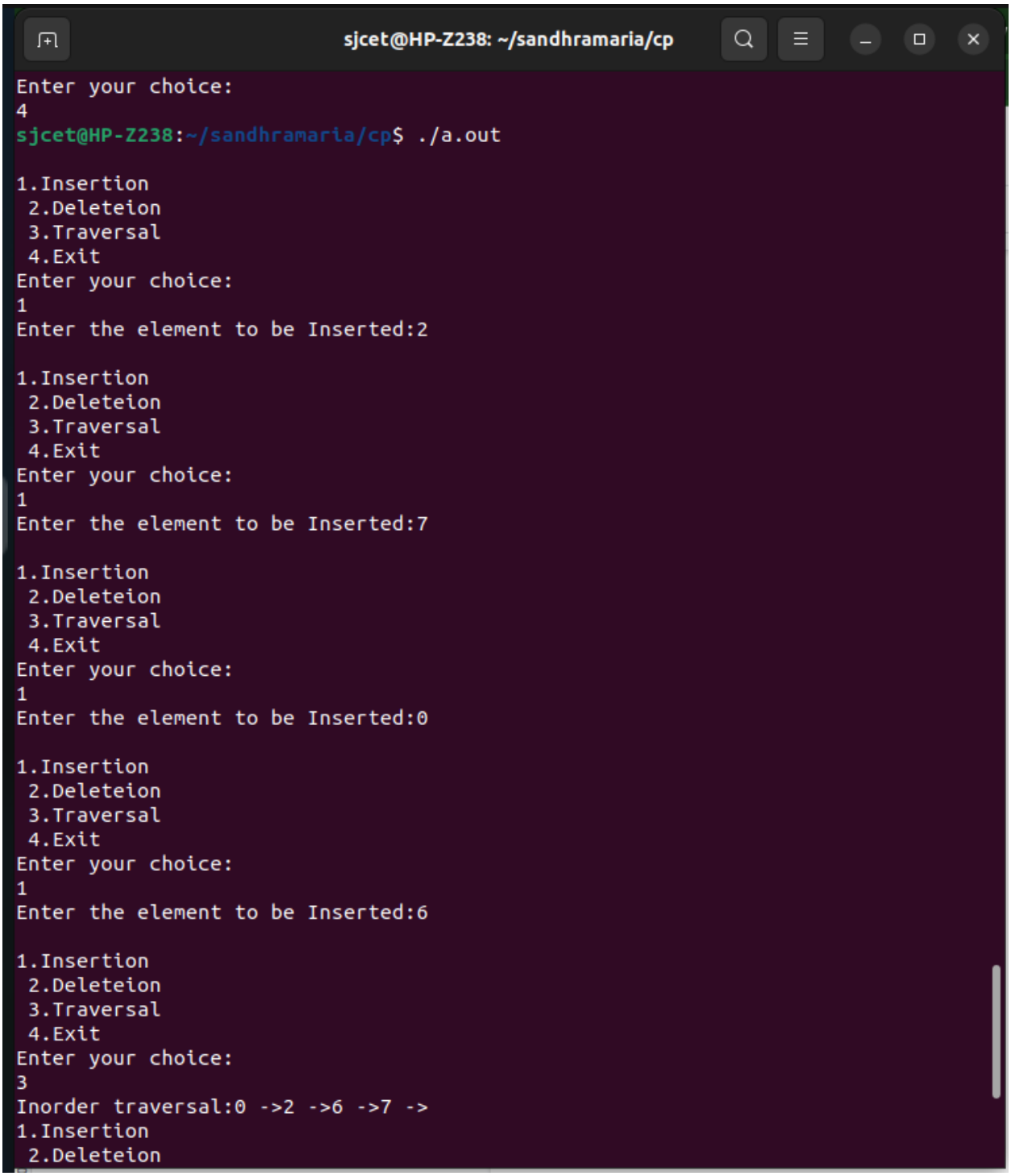
14. Implementation of binary search tree

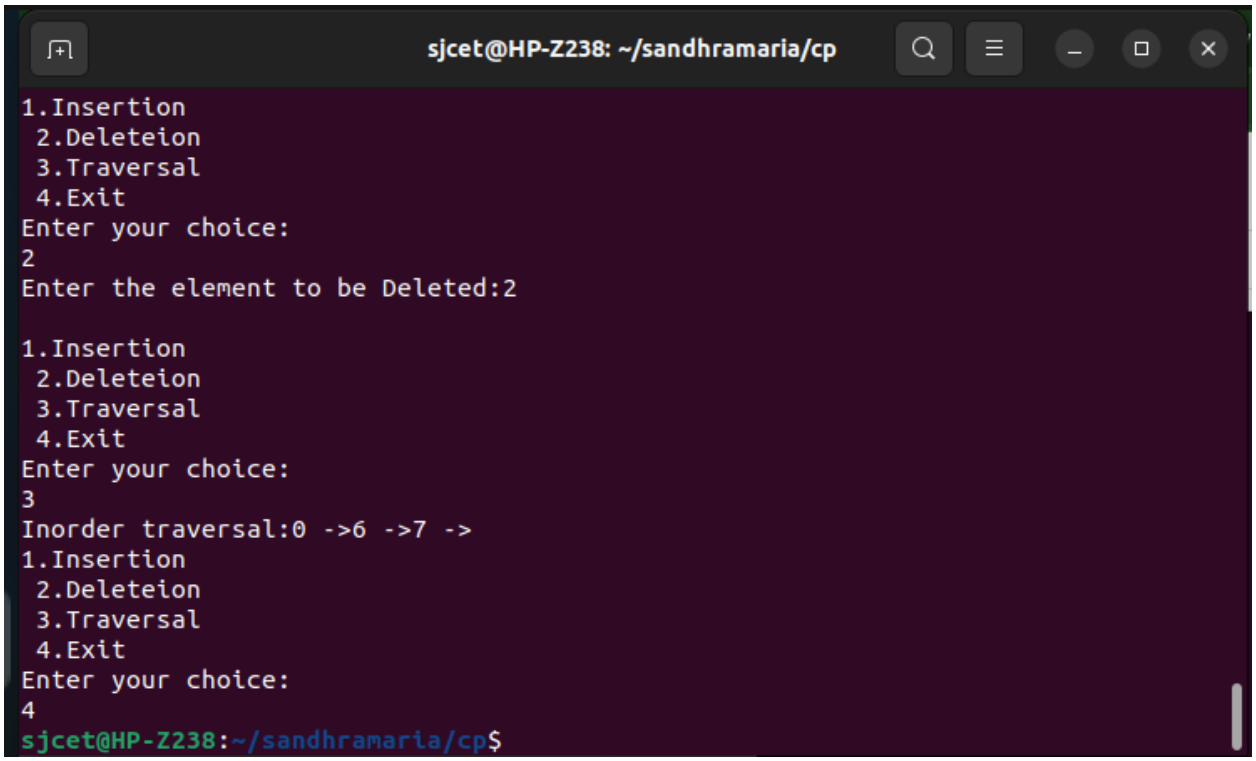
```
#include <stdio.h>
#include <stdlib.h>

struct node {
  int key;
  struct node *left, *right;
};

struct node *newNode(int item) {
  struct node *temp = (struct node *)malloc(sizeof(struct node));
  temp->key = item;
  temp->left = temp->right = NULL;
  return temp;
}

void inorder(struct node *root) {
  if (root != NULL) {
    inorder(root->left);
    printf("%d ->", root->key);
    inorder(root->right);
  }
}

struct node *insert(struct node *node, int key) {
  if (node == NULL) return newNode(key);
  if (key < node->key)
    node->left = insert(node->left, key);
  else
    node->right = insert(node->right, key);

  return node;
}
```

```
struct node *minValueNode(struct node *node) {
  struct node *current = node;


  while (current && current->left != NULL)
    current = current->left;

  return current;
}

struct node *deleteNode(struct node *root, int key) {

  if (root == NULL) return root;

  if (key < root->key)
    root->left = deleteNode(root->left, key);
  else if (key > root->key)
    root->right = deleteNode(root->right, key);

  else {

    if (root->left == NULL) {
      struct node *temp = root->right;
      free(root);
      return temp;
    } else if (root->right == NULL) {
      struct node *temp = root->left;
      free(root);
      return temp;
    }

    struct node *temp = minValueNode(root->right);

    root->key = temp->key;
```

```
    root->right = deleteNode(root->right, temp->key);
  }
  return root;
}
void main()
{

struct node *root = NULL;
int choice, n;
while(1){
printf(" \n1.Insertion");
printf("\n 2.Deleteion");
printf("\n 3.Traversal");
printf("\n 4.Exit");
printf("\nEnter your choice:\n");
scanf("%d",&choice);
switch(choice)
{
case 1: printf("Enter the element to be Inserted:");
scanf("%d",&n);
root = insert(root, n);
break;
case 2: printf("Enter the element to be Deleted:");
scanf("%d",&n);
root = deleteNode(root, n);
break;
case 3: printf("Inorder traversal:");
inorder(root);
break;
case 4:
exit(0);
break;
default:
printf("n Wrong Choice:n");
```

```
break;
}
}
}
```


sjcet@HP-Z238: ~/sandhramaria/cp
Enter your choice:
4
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
1.Insertion
2.Deleteion
3.Traversal
4.Exit
Enter your choice:
1
Enter the element to be Inserted:2
1.Insertion
2.Deleteion
3.Traversal
4.Exit
Enter your choice:
1
Enter the element to be Inserted:7
1.Insertion
2.Deleteion
3.Traversal
4.Exit
Enter your choice:
1
Enter the element to be Inserted:0
1.Insertion
2.Deleteion
3.Traversal
4.Exit
Enter your choice:
1
Enter the element to be Inserted:6
1.Insertion
2.Deleteion
3.Traversal
4.Exit
Enter your choice:
3
Inorder traversal:0 ->2 ->6 ->7 ->
1.Insertion
2.Deleteion


sjcet@HP-Z238: ~/sandhramaria/cp
1.Insertion
2.Deleteion
3.Traversal
4.Exit
Enter your choice:
2
Enter the element to be Deleted:2
1.Insertion
2.Deleteion
3.Traversal
4.Exit
Enter your choice:
3
Inorder traversal:0 ->6 ->7 ->
1.Insertion
2.Deleteion
3.Traversal
4.Exit
Enter your choice:
4
sjcet@HP-Z238:~/sandhramaria/cp$

15.Implementation of B-tree

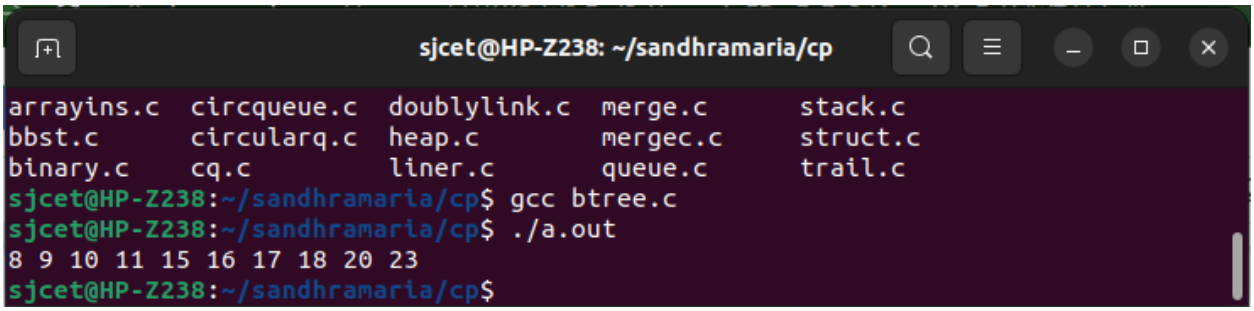
```
#include <stdio.h>
#include <stdlib.h>
#define MAX 3
#define MIN 2
struct BTreeNode {
  int val[MAX + 1], count;
  struct BTreeNode *link[MAX + 1];
};
struct BTreeNode *root;
struct BTreeNode *createNode(int val, struct BTreeNode *child)
{
  struct BTreeNode *newNode;
  newNode = (struct BTreeNode *)malloc(sizeof(struct BTreeNode));
  newNode->val[1] = val;
  newNode->count = 1;
  newNode->link[0] = root;
  newNode->link[1] = child;
  return newNode;
  }
void insertNode(int val, int pos, struct BTreeNode *node,
        struct BTreeNode *child) {
  int j = node->count;
  while (j > pos) {
    node->val[j + 1] = node->val[j];
    node->link[j + 1] = node->link[j];
    j--;
  }
  node->val[j + 1] = val;
  node->link[j + 1] = child;
  node->count++;
}
void splitNode(int val, int *pval, int pos, struct BTreeNode *node,
         struct BTreeNode *child, struct BTreeNode **newNode) {
```

```
  int median, j;
  if (pos > MIN)
    median = MIN + 1;
  else
    median = MIN;
  *newNode = (struct BTreeNode *)malloc(sizeof(struct BTreeNode));
  j = median + 1;
  while (j <= MAX) {
    (*newNode)->val[j - median] = node->val[j];
    (*newNode)->link[j - median] = node->link[j];
    j++;
  }
  node->count = median;
  (*newNode)->count = MAX - median;

  if (pos <= MIN) {
    insertNode(val, pos, node, child);
  } else {
    insertNode(val, pos - median, *newNode, child);
  }
  *pval = node->val[node->count];
  (*newNode)->link[0] = node->link[node->count];
  node->count--;
}
int setValue(int val, int *pval,
        struct BTreeNode *node, struct BTreeNode **child) {
  int pos;
  if (!node) {
    *pval = val;
    *child = NULL;
    return 1;
  }
  if (val < node->val[1]) {
    pos = 0;
  } else {
```

```
    for (pos = node->count;
       (val < node->val[pos] && pos > 1); pos--)
      ;
    if (val == node->val[pos]) {
      printf("Duplicates are not permitted\n");
      return 0;
    }
  }
  if (setValue(val, pval, node->link[pos], child)) {
    if (node->count < MAX) {
      insertNode(*pval, pos, node, *child);
    } else {
      splitNode(*pval, pval, pos, node, *child, child);
      return 1;
    }
  }
  return 0;
}
void insert(int val) {
  int flag, i;
  struct BTreeNode *child;

  flag = setValue(val, &i, root, &child);
  if (flag)
    root = createNode(i, child);
}
void search(int val, int *pos, struct BTreeNode *myNode) {
  if (!myNode) {
    return;
  }
  if (val < myNode->val[1]) {
    *pos = 0;
  } else {
    for (*pos = myNode->count;
       (val < myNode->val[*pos] && *pos > 1); (*pos)--)
```

```
    ;
   if (val == myNode->val[*pos]) {
    printf("%d is found", val);
    return;
   }
  }
  search(val, pos, myNode->link[*pos]);

  return;
}
void traversal(struct BTreeNode *myNode) {
  int i;
  if (myNode) {
   for (i = 0; i < myNode->count; i++) {
    traversal(myNode->link[i]);
    printf("%d ", myNode->val[i + 1]);
   }
   traversal(myNode->link[i]);
  }
}
int main() {
  int val, ch;
  insert(8);
  insert(9);
  insert(10);
  insert(11);
  insert(15);
  insert(16);
  insert(17);
  insert(18);
  insert(20);
  insert(23);
  traversal(root);
  printf("\n");
  search(11, &ch, root);
```

}

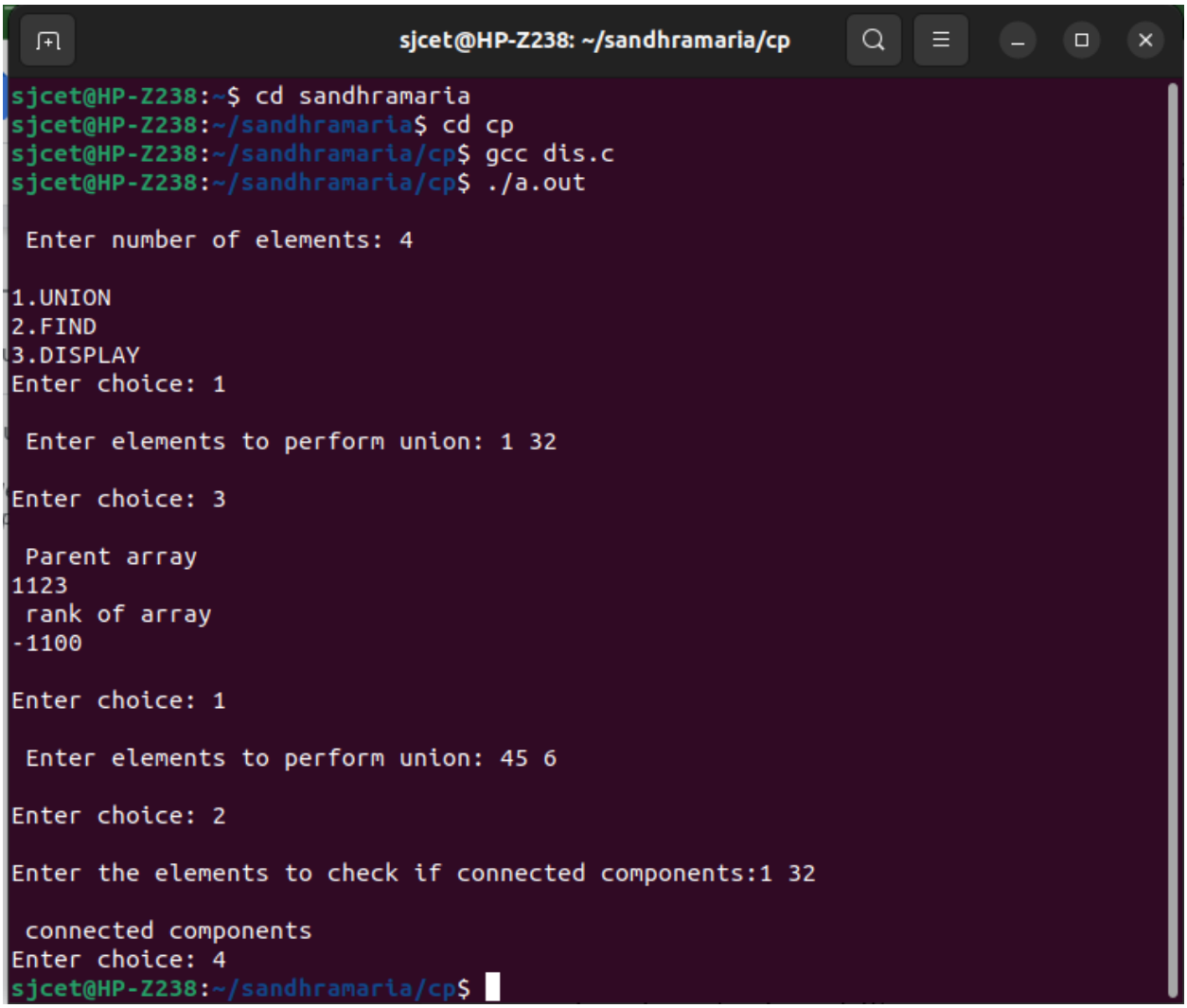

sjcet@HP-Z238: ~/sandhramaria/cp
arrayins.c circqueue.c doublylink.c merge.c stack.c
bbst.c circularq.c heap.c mergec.c struct.c
binary.c cq.c liner.c queue.c trail.c
sjcet@HP-Z238:~/sandhramaria/cp$ gcc btree.c
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
8 9 10 11 15 16 17 18 20 23
sjcet@HP-Z238:~/sandhramaria/cp$

16. Implementation of disjoint set

```
#include<stdio.h>
struct disjointset
{
int parent[20];
int rank[10];
int n;
};
struct disjointset dis;

void makeset()
{
int i;
for(i=0;i<dis.n;i++)
{
dis.parent[i]=i;
dis.rank[i]=0;
}
}

void displayset()
{
int i;
printf("\n Parent array\n");
for(i=0;i<dis.n;i++)
{
printf("%d",dis.parent[i]);
}
printf("\n rank of array\n");
for(i=0;i<dis.n;i++)
printf("%d",dis.rank[i]);
printf("\n");
}
```

```
int find(int x)
{
if(dis.parent[x]!=x)
dis.parent[x]=find(dis.parent[x]);
return dis.parent[x];
}


void Union(int x,int y)
{
int xset=find(x),yset=find(y);
if(xset==yset)
return;
if(dis.rank[xset]<dis.rank[yset])
{dis.parent[xset]=yset;
dis.rank[xset]=-1;
}
else if(dis.rank[xset]>dis.rank[yset])
{
dis.parent[yset]=xset;
dis.rank[yset]=-1;
}
else
{
dis.parent[yset]=xset;
dis.rank[xset]=dis.rank[xset]+1;
dis.rank[yset]=-1;
}
}

int main()
{
int x,y,n;
printf("\n Enter number of elements: ");
scanf("%d",&dis.n);
```

```
makeset();
int ch,w;
printf("\n1.UNION\n2.FIND\n3.DISPLAY");
do{
printf("\nEnter choice: ");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\n Enter elements to perform union: ");
scanf("%d%d",&x,&y);
Union(x,y);
break;

case 2:
printf("\nEnter the elements to check if connected components:");
scanf("%d%d",&x,&y);
if(find(x)==find(y))
printf("\n connected components");
else
printf("\n no connected components");
break;

case 3:
displayset();
break;

case 4:
printf("EXIT");
break;
}
}while(ch!=4);
return 0;
}
```

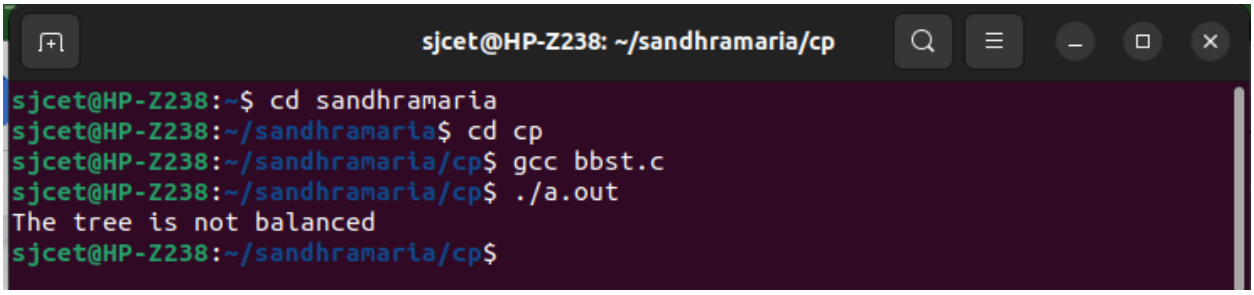

sjcet@HP-Z238: ~/sandhramaria/cp
sjcet@HP-Z238:~$ cd sandhramaria
sjcet@HP-Z238:~/sandhramaria$ cd cp
sjcet@HP-Z238:~/sandhramaria/cp$ gcc dis.c
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
Enter number of elements: 4
1.UNION
2.FIND
3.DISPLAY
Enter choice: 1
Enter elements to perform union: 1 32
Enter choice: 3
Parent array
1123
rank of array
-1100
Enter choice: 1
Enter elements to perform union: 45 6
Enter choice: 2
Enter the elements to check if connected components:1 32
connected components
Enter choice: 4
sjcet@HP-Z238:~/sandhramaria/cp$

17. Implementation of Balanced Binary search Tree

```
#include <stdio.h>
#include <stdlib.h>
#define bool int
struct node {
  int item;
  struct node *left;
  struct node *right;
};

struct node *newNode(int item)
{
  struct node *node = (struct node *)malloc(sizeof(struct node));
  node->item = item;
  node->left = NULL;
  node->right = NULL;
  return (node);
}

bool checkHeightBalance(struct node *root, int *height) {
  int leftHeight = 0, rightHeight = 0;
  int l = 0, r = 0;
  if (root == NULL)
  {
    *height = 0;
    return 1;
  }
  l = checkHeightBalance(root->left, &leftHeight);
  r = checkHeightBalance(root->right, &rightHeight);
  *height = (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;
  if ((leftHeight - rightHeight >= 2) || (rightHeight - leftHeight >= 2))
    return 0;
  else
    return l && r;
```

```
}

int main()
{
  int height = 0;
  struct node *root = newNode(1);
  root->left = newNode(2);
  root->right = newNode(3);
  root->left->left = newNode(4);
  root->left->right = newNode(5);
  root->left->right->right = newNode(6);
  if (checkHeightBalance(root, &height))
    printf("The tree is balanced");
  else
    printf("The tree is not balanced");
}
```

OBJ OBJ OBJ

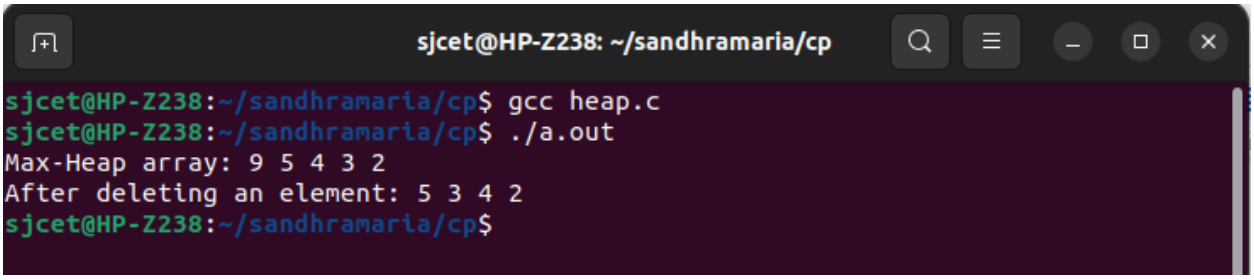


18.Max-Heap implementation

```
#include <stdio.h>
int size = 0;
void swap(int *a, int *b)
{
  int temp = *b;
  *b = *a;
  *a = temp;
}
void heapify(int array[], int size, int i)
{
  if (size == 1)
  {
    printf("Single element in the heap");
  }
  else
  {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < size && array[l] > array[largest])
      largest = l;
    if (r < size && array[r] > array[largest])
      largest = r;
    if (largest != i)
    {
      swap(&array[i], &array[largest]);
      heapify(array, size, largest);
    }
  }
}
void insert(int array[], int newNum)
{
  if (size == 0)
```

```
  {
    array[0] = newNum;
    size += 1;
  }
  else
  {
    array[size] = newNum;
    size += 1;
    for (int i = size / 2 - 1; i >= 0; i--)
    {
      heapify(array, size, i);
    }
  }
}
void deleteRoot(int array[], int num)
{
  int i;
  for (i = 0; i < size; i++)
  {
    if (num == array[i])
      break;
  }
  swap(&array[i], &array[size - 1]);
  size -= 1;
  for (int i = size / 2 - 1; i >= 0; i--)
  {
    heapify(array, size, i);
  }
}
void printArray(int array[], int size)
{
  for (int i = 0; i < size; ++i)
    printf("%d ", array[i]);
  printf("\n");
}
```

```
int main()
{
  int array[10];
  insert(array, 3);
  insert(array, 4);
  insert(array, 9);
  insert(array, 5);
  insert(array, 2);
  printf("Max-Heap array: ");
  printArray(array, size);
  deleteRoot(array, 9);
  printf("After deleting an element: ");
  printArray(array, size);
}
```



19. Red-Black tree implementation

```
#include <stdio.h>
#include <stdlib.h>
enum nodeColor {
  RED,BLACK
};
struct rbNode {
  int data, color;
  struct rbNode *link[2];
};
struct rbNode *root = NULL;
struct rbNode *createNode(int data) {
  struct rbNode *newnode;
  newnode = (struct rbNode *)malloc(sizeof(struct rbNode));
  newnode->data = data;
  newnode->color = RED;
  newnode->link[0] = newnode->link[1] = NULL;
  return newnode;
}
void insertion(int data) {
  struct rbNode *stack[98], *ptr, *newnode, *xPtr, *yPtr;
  int dir[98], ht = 0, index;
  ptr = root;
  if (!root) {
    root = createNode(data);
    return;
  }
  stack[ht] = root;
  dir[ht++] = 0;
  while (ptr != NULL) {
    if (ptr->data == data) {
      printf("Duplicates Not Allowed!!\n");
      return;
    }
```

```
  index = (data - ptr->data) > 0 ? 1 : 0;
  stack[ht] = ptr;
  ptr = ptr->link[index];
  dir[ht++] = index;
 }
stack[ht - 1]->link[index] = newnode = createNode(data);
while ((ht >= 3) && (stack[ht - 1]->color == RED)) {
  if (dir[ht - 2] == 0) {
    yPtr = stack[ht - 2]->link[1];
    if (yPtr != NULL && yPtr->color == RED) {
      stack[ht - 2]->color = RED;
      stack[ht - 1]->color = yPtr->color = BLACK;
      ht = ht - 2;
    } else {
      if (dir[ht - 1] == 0) {
        yPtr = stack[ht - 1];
      } else {
        xPtr = stack[ht - 1];
        yPtr = xPtr->link[1];
        xPtr->link[1] = yPtr->link[0];
        yPtr->link[0] = xPtr;
        stack[ht - 2]->link[0] = yPtr;
      }
      xPtr = stack[ht - 2];
      xPtr->color = RED;
      yPtr->color = BLACK;
      xPtr->link[0] = yPtr->link[1];
      yPtr->link[1] = xPtr;
      if (xPtr == root) {
        root = yPtr;
      } else {
        stack[ht - 3]->link[dir[ht - 3]] = yPtr;
      }
      break;
    }
```

```
    } else {
      yPtr = stack[ht - 2]->link[0];
      if ((yPtr != NULL) && (yPtr->color == RED)) {
        stack[ht - 2]->color = RED;
        stack[ht - 1]->color = yPtr->color = BLACK;
        ht = ht - 2;
      } else {
        if (dir[ht - 1] == 1) {
          yPtr = stack[ht - 1];
        } else {
          xPtr = stack[ht - 1];
          yPtr = xPtr->link[0];
          xPtr->link[0] = yPtr->link[1];
          yPtr->link[1] = xPtr;
          stack[ht - 2]->link[1] = yPtr;
        }
        xPtr = stack[ht - 2];
        yPtr->color = BLACK;
        xPtr->color = RED;
        xPtr->link[1] = yPtr->link[0];
        yPtr->link[0] = xPtr;
        if (xPtr == root) {
          root = yPtr;
        } else {
          stack[ht - 3]->link[dir[ht - 3]] = yPtr;
        }
        break;
      }
    }
  }
  root->color = BLACK;
}
void deletion(int data) {
  struct rbNode *stack[98], *ptr, *xPtr, *yPtr;
  struct rbNode *pPtr, *qPtr, *rPtr;
```

```
int dir[98], ht = 0, diff, i;
enum nodeColor color;

if (!root) {
  printf("Tree not available\n");
  return;
}

ptr = root;
while (ptr != NULL) {
  if ((data - ptr->data) == 0)
    break;
  diff = (data - ptr->data) > 0 ? 1 : 0;
  stack[ht] = ptr;
  dir[ht++] = diff;
  ptr = ptr->link[diff];
}

if (ptr->link[1] == NULL) {
  if ((ptr == root) && (ptr->link[0] == NULL)) {
    free(ptr);
    root = NULL;
  } else if (ptr == root) {
    root = ptr->link[0];
    free(ptr);
  } else {
    stack[ht - 1]->link[dir[ht - 1]] = ptr->link[0];
  }
} else {
  xPtr = ptr->link[1];
  if (xPtr->link[0] == NULL) {
    xPtr->link[0] = ptr->link[0];
    color = xPtr->color;
    xPtr->color = ptr->color;
    ptr->color = color;
```

```
  if (ptr == root) {
    root = xPtr;
  } else {
    stack[ht - 1]->link[dir[ht - 1]] = xPtr;
  }

  dir[ht] = 1;
  stack[ht++] = xPtr;
} else {
  i = ht++;
  while (1) {
    dir[ht] = 0;
    stack[ht++] = xPtr;
    yPtr = xPtr->link[0];
    if (!yPtr->link[0])
      break;
    xPtr = yPtr;
  }

  dir[i] = 1;
  stack[i] = yPtr;
  if (i > 0)
    stack[i - 1]->link[dir[i - 1]] = yPtr;

  yPtr->link[0] = ptr->link[0];

  xPtr->link[0] = yPtr->link[1];
  yPtr->link[1] = ptr->link[1];

  if (ptr == root) {
    root = yPtr;
  }

  color = yPtr->color;
```

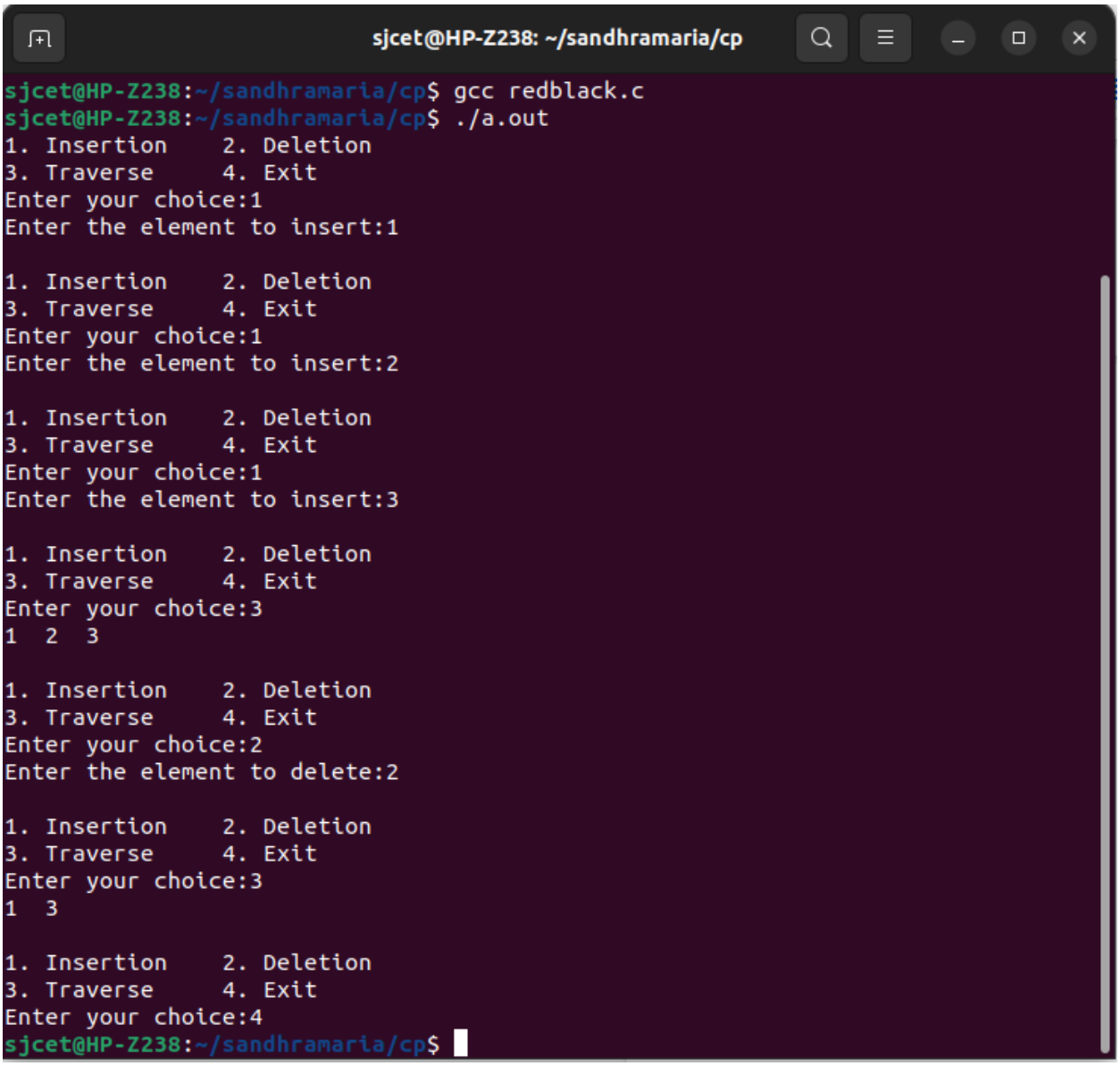
```
    yPtr->color = ptr->color;
    ptr->color = color;
  }
}

if (ht < 1)
  return;

if (ptr->color == BLACK) {
  while (1) {
    pPtr = stack[ht - 1]->link[dir[ht - 1]];
    if (pPtr && pPtr->color == RED) {
      pPtr->color = BLACK;
      break;
    }

    if (ht < 2)
      break;

    if (dir[ht - 2] == 0) {
      rPtr = stack[ht - 1]->link[1];

      if (!rPtr)
        break;

      if (rPtr->color == RED) {
        stack[ht - 1]->color = RED;
        rPtr->color = BLACK;
        stack[ht - 1]->link[1] = rPtr->link[0];
        rPtr->link[0] = stack[ht - 1];

        if (stack[ht - 1] == root) {
          root = rPtr;
        } else {
          stack[ht - 2]->link[dir[ht - 2]] = rPtr;
```

```
    }
    dir[ht] = 0;
    stack[ht] = stack[ht - 1];
    stack[ht - 1] = rPtr;
    ht++;

    rPtr = stack[ht - 1]->link[1];
   }

  if ((!rPtr->link[0] || rPtr->link[0]->color == BLACK) &&
    (!rPtr->link[1] || rPtr->link[1]->color == BLACK)) {
    rPtr->color = RED;
  } else {
    if (!rPtr->link[1] || rPtr->link[1]->color == BLACK) {
      qPtr = rPtr->link[0];
      rPtr->color = RED;
      qPtr->color = BLACK;
      rPtr->link[0] = qPtr->link[1];
      qPtr->link[1] = rPtr;
      rPtr = stack[ht - 1]->link[1] = qPtr;
    }
    rPtr->color = stack[ht - 1]->color;
    stack[ht - 1]->color = BLACK;
    rPtr->link[1]->color = BLACK;
    stack[ht - 1]->link[1] = rPtr->link[0];
    rPtr->link[0] = stack[ht - 1];
    if (stack[ht - 1] == root) {
      root = rPtr;
    } else {
      stack[ht - 2]->link[dir[ht - 2]] = rPtr;
    }
    break;
  }
 } else {
  rPtr = stack[ht - 1]->link[0];
```

```
if (!rPtr)
  break;

if (rPtr->color == RED) {
  stack[ht - 1]->color = RED;
  rPtr->color = BLACK;
  stack[ht - 1]->link[0] = rPtr->link[1];
  rPtr->link[1] = stack[ht - 1];

  if (stack[ht - 1] == root) {
    root = rPtr;
  } else {
    stack[ht - 2]->link[dir[ht - 2]] = rPtr;
  }
  dir[ht] = 1;
  stack[ht] = stack[ht - 1];
  stack[ht - 1] = rPtr;
  ht++;

  rPtr = stack[ht - 1]->link[0];
}
if ((!rPtr->link[0] || rPtr->link[0]->color == BLACK) &&
  (!rPtr->link[1] || rPtr->link[1]->color == BLACK)) {
  rPtr->color = RED;
} else {
  if (!rPtr->link[0] || rPtr->link[0]->color == BLACK) {
    qPtr = rPtr->link[1];
    rPtr->color = RED;
    qPtr->color = BLACK;
    rPtr->link[1] = qPtr->link[0];
    qPtr->link[0] = rPtr;
    rPtr = stack[ht - 1]->link[0] = qPtr;
  }
  rPtr->color = stack[ht - 1]->color;
  stack[ht - 1]->color = BLACK;
```

```
          rPtr->link[0]->color = BLACK;
          stack[ht - 1]->link[0] = rPtr->link[1];
          rPtr->link[1] = stack[ht - 1];
          if (stack[ht - 1] == root) {
            root = rPtr;
          } else {
            stack[ht - 2]->link[dir[ht - 2]] = rPtr;
          }
          break;
        }
      }
      ht--;
    }
  }
}
void inorderTraversal(struct rbNode *node) {
  if (node) {
    inorderTraversal(node->link[0]);
    printf("%d  ", node->data);
    inorderTraversal(node->link[1]);
  }
  return;
}
int main() {
  int ch, data;
  while (1) {
    printf("1. Insertion\t2. Deletion\n");
    printf("3. Traverse\t4. Exit");
    printf("\nEnter your choice:");
    scanf("%d", &ch);
    switch (ch) {
      case 1:
        printf("Enter the element to insert:");
        scanf("%d", &data);
        insertion(data);
```

```
      break;
    case 2:
      printf("Enter the element to delete:");
      scanf("%d", &data);
      deletion(data);
      break;
    case 3:
      inorderTraversal(root);
      printf("\n");
      break;
    case 4:
      exit(0);
    default:
      printf("Not available\n");
      break;
   }
   printf("\n");
  }
  return 0;
}
```


sjcet@HP-Z238: ~/sandhramaria/cp
sjcet@HP-Z238:~/sandhramaria/cp$ gcc redblack.c
sjcet@HP-Z238:~/sandhramaria/cp$ ./a.out
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:1
Enter the element to insert:1
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:1
Enter the element to insert:2
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:1
Enter the element to insert:3
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:3
1 2 3
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:2
Enter the element to delete:2
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:3
1 3
1. Insertion 2. Deletion
3. Traverse 4. Exit
Enter your choice:4
sjcet@HP-Z238:~/sandhramaria/cp$

0

20. Implementation of binomial heap

```
#include<stdio.h>
#include<malloc.h>

struct node {
   int n;
   int degree;
   struct node* parent;
   struct node* child;
   struct node* sibling;
};

struct node* MAKE_bin_HEAP();
int bin_LINK(struct node*, struct node*);
struct node* CREATE_NODE(int);
struct node* bin_HEAP_UNION(struct node*, struct node*);
struct node* bin_HEAP_INSERT(struct node*, struct node*);
struct node* bin_HEAP_MERGE(struct node*, struct node*);
struct node* bin_HEAP_EXTRACT_MIN(struct node*);
int REVERT_LIST(struct node*);
int DISPLAY(struct node*);
struct node* FIND_NODE(struct node*, int);
int bin_HEAP_DECREASE_KEY(struct node*, int, int);
int bin_HEAP_DELETE(struct node*, int);

int count = 1;

struct node* MAKE_bin_HEAP() {
   struct node* np;
   np = NULL;
   return np;
}

struct node * H = NULL;
```

```
struct node *Hr = NULL;

int bin_LINK(struct node* y, struct node* z) {
   y->parent = z;
   y->sibling = z->child;
   z->child = y;
   z->degree = z->degree + 1;
}

struct node* CREATE_NODE(int k) {
   struct node* p;//new node;
   p = (struct node*) malloc(sizeof(struct node));
   p->n = k;
   return p;
}

struct node* bin_HEAP_UNION(struct node* H1, struct node* H2) {
   struct node* prev_x;
   struct node* next_x;
   struct node* x;
   struct node* H = MAKE_bin_HEAP();
   H = bin_HEAP_MERGE(H1, H2);
   if (H == NULL)
      return H;
   prev_x = NULL;
   x = H;
   next_x = x->sibling;
   while (next_x != NULL) {
      if ((x->degree != next_x->degree) || ((next_x->sibling != NULL)
            && (next_x->sibling)->degree == x->degree)) {
         prev_x = x;
         x = next_x;
      } else {
         if (x->n <= next_x->n) {
            x->sibling = next_x->sibling;
```

```
            bin_LINK(next_x, x);
        } else {
            if (prev_x == NULL)
                H = next_x;
            else
                prev_x->sibling = next_x;
            bin_LINK(x, next_x);
            x = next_x;
        }
    }
    next_x = x->sibling;
  }
  return H;
}

struct node* bin_HEAP_INSERT(struct node* H, struct node* x) {
  struct node* H1 = MAKE_bin_HEAP();
  x->parent = NULL;
  x->child = NULL;
  x->sibling = NULL;
  x->degree = 0;
  H1 = x;
  H = bin_HEAP_UNION(H, H1);
  return H;
}

struct node* bin_HEAP_MERGE(struct node* H1, struct node* H2) {
  struct node* H = MAKE_bin_HEAP();
  struct node* y;
  struct node* z;
  struct node* a;
  struct node* b;
  y = H1;
  z = H2;
  if (y != NULL) {
```

```
        if (z != NULL && y->degree <= z->degree)
            H = y;
        else if (z != NULL && y->degree > z->degree)
            /* need some modifications here;the first and the else conditions can be
merged together!!!! */
            H = z;
        else
            H = y;
    } else
        H = z;
    while (y != NULL && z != NULL) {
        if (y->degree < z->degree) {
            y = y->sibling;
        } else if (y->degree == z->degree) {
            a = y->sibling;
            y->sibling = z;
            y = a;
        } else {
            b = z->sibling;
            z->sibling = y;
            z = b;
        }
    }
    return H;
}

int DISPLAY(struct node* H) {
    struct node* p;
    if (H == NULL) {
        printf("\nHEAP EMPTY");
        return 0;
    }
    printf("\nTHE ROOT NODES ARE:-\n");
    p = H;
    while (p != NULL) {
```

```
        printf("%d", p->n);
        if (p->sibling != NULL)
            printf("-->");
        p = p->sibling;
    }
    printf("\n");
}

struct node* bin_HEAP_EXTRACT_MIN(struct node* H1) {
    int min;
    struct node* t = NULL;
    struct node* x = H1;
    struct node *Hr;
    struct node* p;
    Hr = NULL;
    if (x == NULL) {
        printf("\nNOTHING TO EXTRACT");
        return x;
    }
    //    int min=x->n;
    p = x;
    while (p->sibling != NULL) {
        if ((p->sibling)->n < min) {
            min = (p->sibling)->n;
            t = p;
            x = p->sibling;
        }
        p = p->sibling;
    }
    if (t == NULL && x->sibling == NULL)
        H1 = NULL;
    else if (t == NULL)
        H1 = x->sibling;
    else if (t->sibling == NULL)
        t = NULL;
```

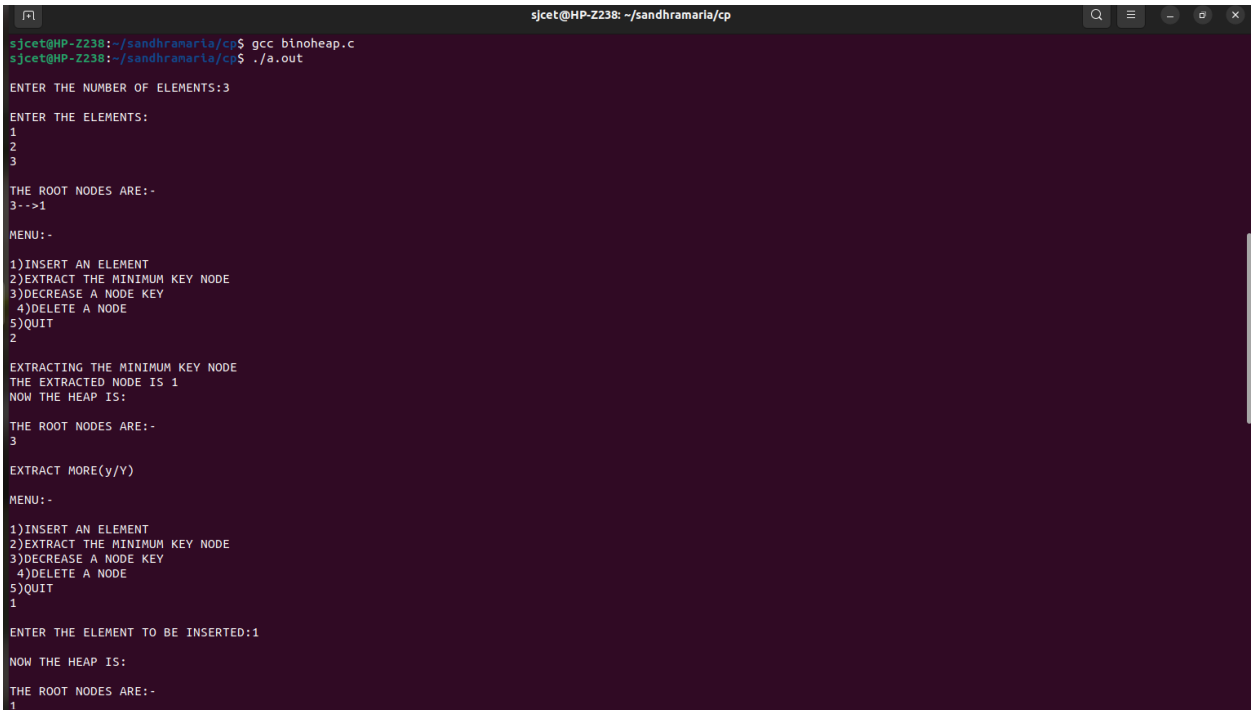
```
    else
        t->sibling = x->sibling;
    if (x->child != NULL) {
        REVERT_LIST(x->child);
        (x->child)->sibling = NULL;
    }
    H = bin_HEAP_UNION(H1, Hr);
    return x;
}

int REVERT_LIST(struct node* y) {
    if (y->sibling != NULL) {
        REVERT_LIST(y->sibling);
        (y->sibling)->sibling = y;
    } else {
        Hr = y;
    }
}

struct node* FIND_NODE(struct node* H, int k) {
    struct node* x = H;
    struct node* p = NULL;
    if (x->n == k) {
        p = x;
        return p;
    }
    if (x->child != NULL && p == NULL) {
        p = FIND_NODE(x->child, k);
    }

    if (x->sibling != NULL && p == NULL) {
        p = FIND_NODE(x->sibling, k);
    }
    return p;
}
```

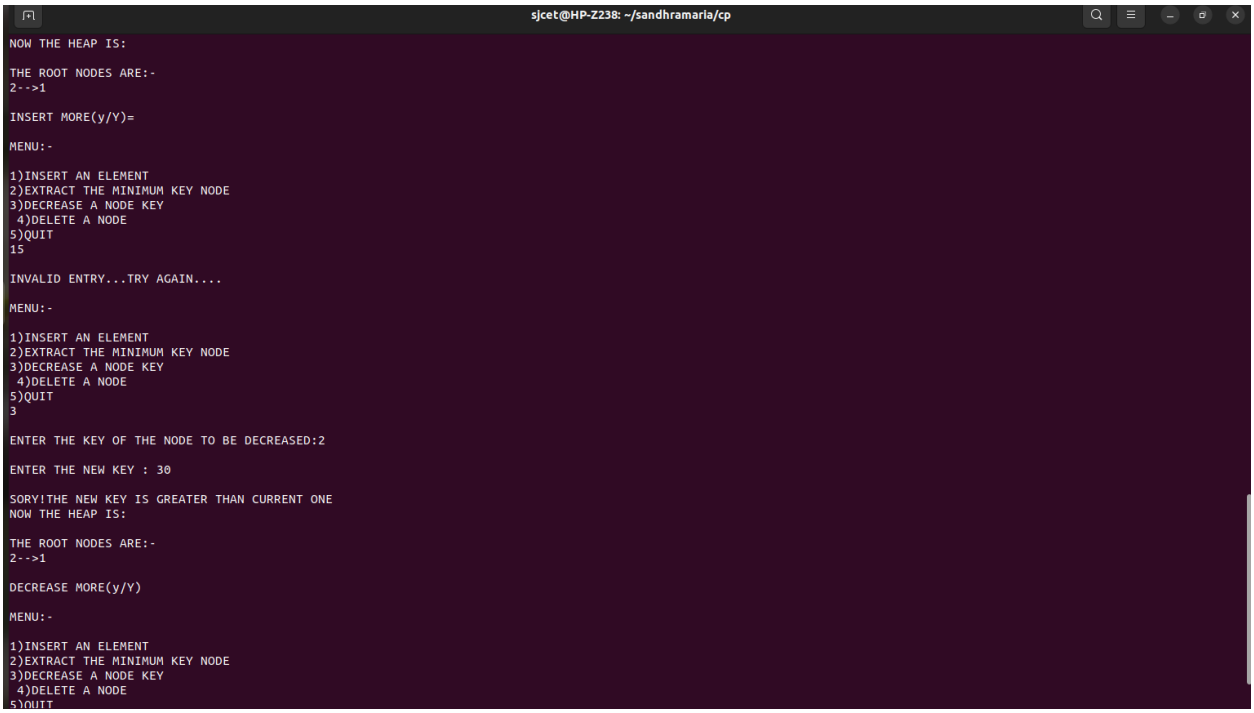
```
int bin_HEAP_DECREASE_KEY(struct node* H, int i, int k) {
    int temp;
    struct node* p;
    struct node* y;
    struct node* z;
    p = FIND_NODE(H, i);
    if (p == NULL) {
        printf("\nINVALID CHOICE OF KEY TO BE REDUCED");
        return 0;
    }
    if (k > p->n) {
        printf("\nSORY!THE NEW KEY IS GREATER THAN CURRENT ONE");
        return 0;
    }
    p->n = k;
    y = p;
    z = p->parent;
    while (z != NULL && y->n < z->n) {
        temp = y->n;
        y->n = z->n;
        z->n = temp;
        y = z;
        z = z->parent;
    }
    printf("\nKEY REDUCED SUCCESSFULLY!");
}

int bin_HEAP_DELETE(struct node* H, int k) {
    struct node* np;
    if (H == NULL) {
        printf("\nHEAP EMPTY");
        return 0;
    }
```

```
    bin_HEAP_DECREASE_KEY(H, k, -1000);
    np = bin_HEAP_EXTRACT_MIN(H);
    if (np != NULL)
        printf("\nNODE DELETED SUCCESSFULLY");
}

int main() {
    int i, n, m, l;
    struct node* p;
    struct node* np;
    char ch;
    printf("\nENTER THE NUMBER OF ELEMENTS:");
    scanf("%d", &n);
    printf("\nENTER THE ELEMENTS:\n");
    for (i = 1; i <= n; i++) {
        scanf("%d", &m);
        np = CREATE_NODE(m);
        H = bin_HEAP_INSERT(H, np);
    }
    DISPLAY(H);
    do {
        printf("\nMENU:-\n");
        printf(
                "\n1)INSERT AN ELEMENT\n2)EXTRACT THE MINIMUM KEY
NODE\n3)DECREASE A NODE KEY\n 4)DELETE A NODE\n5)QUIT\n");
        scanf("%d", &l);
        switch (l) {
        case 1:
            do {
                printf("\nENTER THE ELEMENT TO BE INSERTED:");
                scanf("%d", &m);
                p = CREATE_NODE(m);
                H = bin_HEAP_INSERT(H, p);
                printf("\nNOW THE HEAP IS:\n");
                DISPLAY(H);
```

```
        printf("\nINSERT MORE(y/Y)= \n");
        fflush(stdin);
        scanf("%c", &ch);
    } while (ch == 'Y' || ch == 'y');
    break;
case 2:
    do {
        printf("\nEXTRACTING THE MINIMUM KEY NODE");
        p = bin_HEAP_EXTRACT_MIN(H);
        if (p != NULL)
            printf("\nTHE EXTRACTED NODE IS %d", p->n);
        printf("\nNOW THE HEAP IS:\n");
        DISPLAY(H);
        printf("\nEXTRACT MORE(y/Y)\n");
        fflush(stdin);
        scanf("%c", &ch);
    } while (ch == 'Y' || ch == 'y');
    break;
case 3:
    do {
        printf("\nENTER THE KEY OF THE NODE TO BE DECREASED:");
        scanf("%d", &m);
        printf("\nENTER THE NEW KEY : ");
        scanf("%d", &l);
        bin_HEAP_DECREASE_KEY(H, m, l);
        printf("\nNOW THE HEAP IS:\n");
        DISPLAY(H);
        printf("\nDECREASE MORE(y/Y)\n");
        fflush(stdin);
        scanf("%c", &ch);
    } while (ch == 'Y' || ch == 'y');
    break;
case 4:
    do {
        printf("\nENTER THE KEY TO BE DELETED: ");
```

```
            scanf("%d", &m);
            bin_HEAP_DELETE(H, m);
            printf("\nDELETE MORE(y/Y)\n");
            fflush(stdin);
            scanf("%c", &ch);
         } while (ch == 'y' || ch == 'Y');
         break;
      case 5:
         printf("\nEXIT\
n");
         break;
      default:
         printf("\nINVALID ENTRY...TRY AGAIN....\n");
      }
   } while (l != 5);
}
```




sjcet@HP-Z238: ~/sandhramaria/cp
NOW THE HEAP IS:
THE ROOT NODES ARE:-
2-->1
INSERT MORE(y/Y)=
MENU:-
1)INSERT AN ELEMENT
2)EXTRACT THE MINIMUM KEY NODE
3)DECREASE A NODE KEY
4)DELETE A NODE
5)QUIT
15
INVALID ENTRY...TRY AGAIN....
MENU:-
1)INSERT AN ELEMENT
2)EXTRACT THE MINIMUM KEY NODE
3)DECREASE A NODE KEY
4)DELETE A NODE
5)QUIT
3
ENTER THE KEY OF THE NODE TO BE DECREASED:2
ENTER THE NEW KEY : 30
SORY!THE NEW KEY IS GREATER THAN CURRENT ONE
NOW THE HEAP IS:
THE ROOT NODES ARE:-
2-->1
DECREASE MORE(y/Y)
MENU:-
1)INSERT AN ELEMENT
2)EXTRACT THE MINIMUM KEY NODE
3)DECREASE A NODE KEY
4)DELETE A NODE
5)QUIT

21. Min Heap implementation

```
#include <stdio.h>
#define HEAP_CAPACITY 10
#define SUCCESS_VAL 99999
#define FAIL_VAL -99999
int size = 0;
int i;
int heap[HEAP_CAPACITY];
void swap(int *a,int *b)
{
   int temp = *b;
   *b = *a;
   *a = temp;
}
void heapify(int i)
{
   if (size == 1)
   {
      return;
   }
   else{
      int smallest = i;
      int left = 2 * i + 1;
      int right = 2 * i + 2;
      if(left < size && heap[left] < heap[smallest])
         smallest = left;
      if(right < size && heap[right] < heap[smallest])
         smallest = right;
      if (smallest != i)
      {
         swap(&heap[i], &heap[smallest]);
         heapify(smallest);
      }
   }
```

```
}
int insert(int newNum)
{
   if(size==0)
   {
      heap[0] = newNum;
      size += 1;
      return SUCCESS_VAL;
   }
   else if(size < HEAP_CAPACITY)
   {
      heap[size] = newNum;
      size += 1;
      for(i =(size-1)/2;i>=0;i--)
      {
         heapify(i);
      }
      return SUCCESS_VAL;
   }
   else
   {
      printf("Heap capacity reached. Insertion failed.\n");
      return FAIL_VAL;
   }
}
int delete(int number)
{
   int i,index=-1;
   if(size <=0)
   {
      printf("Empty min heap");
      return FAIL_VAL;
   }
   for(i=0;i<size;i++)
   {
```

```
        if(number == heap[i])
        {
            index = i;
            break;
        }
    }
    if(index == -1)
    {
        printf("Key is not found\n");
        return FAIL_VAL;
    }
    swap(&heap[i],&heap[size-1]);
    size -= 1;
    for(i=(size-1)/2; i>=0;i--)
    {
        heapify(i);
    }
    return SUCCESS_VAL;
}
void printHeap()
{
    for( i=0;i<size;++i)
    {
        if(i==0)
            printf("%d(root) ", heap[i]);
        else
            printf("%d(%d's child) ",heap[i],heap[(i-1)/2]);
    }
    printf("\n");
}
int main()
{
    while(1)
    {
```

```
        printf("\n___MENU___\n1.Insert Element \n2.Print MinHeap \n3.Delete
Element \n4.Exit \n");
        int choice;
        scanf("%d",&choice);
        if(choice==1)
        {
            printf("Enter the element to be inserted\n");
            int item;
            scanf("%d",&item);
            int res=insert(item);
            if(res==SUCCESS_VAL)
                printf("inserted successfully\n");
        }
        else if(choice==2)
        {
            printHeap();
        }
        else if(choice==3)
        {
            int res = delete(heap[0]);
            if(res==SUCCESS_VAL)
                printf("Delete Successfully\n");
            else
                printf("Deleted Unsuccessfully\n");
            }
        else if(choice==4)
        {
            break;
        }
    }
}
```

OBJ OBJ

OBJ

22.Implementation Of prim's algorithm

```
#include<stdio.h>
#include <string.h>
#include<stdbool.h>
#define INF 9999999
#define V 5
int G[V][V] = {
{0, 9, 75, 0, 0},
{9, 0, 95, 19, 42},
{75, 95, 0, 51, 66},
{0, 19, 51, 0, 31},
{0, 42, 66, 31, 0}};
int main()
{
int no_edge; // number of edge
int selected[V];
memset(selected, false, sizeof(selected));
no_edge = 0;
selected[0] = true;
int x;
int y;
printf("Edge : Weight\n");
while (no_edge < V - 1)
{
int min = INF;
x = 0;
y = 0;
for (int i = 0; i < V; i++)
{
if (selected[i])
{
for (int j = 0; j < V; j++)
{
if (!selected[j] && G[i][j])
```

```
{
if (min > G[i][j]) {
min = G[i][j];
x = i;
y = j;
}
}
}
}
}
printf("%d - %d : %d\n", x, y, G[x][y]);
selected[y] = true;
no_edge++;
}
return 0;
}
```

```
Edge : Weight
0 - 1 : 9
1 - 3 : 19
3 - 4 : 31
3 - 2 : 51
```