# Ex 1: SINGLY LINKED LIST

Program:

```
#include <stdio.h>
#include<malloc.h>
void createfnode(int ele);
void insertfront(int ele);
void insertend(int ele);
void display();
struct node
{
   int data;
   struct node* next;
};
struct node* head = NULL;
struct node *newnode;
void insertfront(int ele)
 {
 newnode=(struct node*)malloc(sizeof(struct node));
 if(newnode!=NULL)
 { newnode->data=ele;
    if(head!=NULL)
    {
       newnode->next=head;
```

```
    head=newnode;
   }
   else
   {

      newnode->next=NULL;
      head=newnode;
   }
}
}

 void insertend(int ele)
{
 newnode=(struct node*)malloc(sizeof(struct node));
 if(newnode!=NULL)
 {
    newnode->data=ele;
    newnode->next=NULL;
  if(head!=NULL)
  {
      struct node *t;
      t=head;
      while(t->next!=NULL)
      {
```

```
            t=t->next;
        }
      newnode->next=NULL;
      t->next=newnode;
    }
   else
    {
       head=newnode;
    }
}


}

int listsize()
{
    int c=0;
    struct node *t;
    t=head;
    while(t!=NULL)
    {
        c=c+1;
        t=t->next;
    }
    printf("\n The size of the list is %d:\n",c);
```

```
    return c;
}
void insertpos(int ele,int pos)
{
   int ls=0;
   ls=listsize();
   if(head == NULL && (pos <= 0 || pos > 1))
   {
      printf("\nInvalid position to insert a node\n");
      return;
   }
  if(head != NULL && (pos <= 0 || pos > ls))
   {
      printf("\nInvalid position to insert a node\n");
      return;
   }
   struct node* newnode = NULL;
 newnode=(struct node*)malloc(sizeof(struct node));

  if(newnode != NULL)
   {
      newnode->data=ele;
      struct node* temp = head;
      int count = 1;
```

```
        while(count < pos-1)
        {
            temp = temp -> next;
            count += 1;
        }
        if(pos == 1)
        {
            newnode->next = head;
            head = newnode;
        }
        else
        {
            newnode->next = temp->next;
            temp->next = newnode;
        }
    }
}
void findnext(int s)
{
    struct node *temp;
    temp=head;
    if(temp==NULL&&temp->next==NULL)
    {
        printf("No next element ");
```

```
    }
    else
    {
        while(temp->data!=s)
        {
            temp=temp->next;

        }
                printf("\nNext Element of %d is %d\n",s,temp->next->data);
    }

}

void findprev(int s)
{
    struct node *temp;
    temp=head;
    if(temp==NULL)
    {
        printf("List is empty ");
    }
    else
    {
        while(temp->next->data!=s)
```

```
        {
          temp=temp->next;
         }
         printf("\n The previous ele of %d is %d\n",s,temp->data);
    }
}
void find(int s)
{
    struct node *temp;
    temp=head;
    if(head==NULL)
    {
       printf("\n List is empty");
    }
    else
    {

          while(temp->data!=s && temp->next!=NULL)
          {
             temp=temp->next;
          }
          if(temp!=NULL && temp->data==s)
          {
      printf("\n Searching ele %d is present in the addr of %p",temp->data,temp);
```

```
    }
    else
    {
        printf("\n Searching elem %d is not present",s);
    }

}
}

void isempty()
{
    if(head==NULL)
    {
        printf("\nList is empty\n");
    }
    else
    {
        printf("\nList is not empty\n");
    }
}

void deleteAtBeginning()
{
    struct node *t;
```

```
    t=head;
    head=t->next;
}

void deleteAtEnd()
{
   struct node *temp;
   temp=head;
   if(head==NULL)
   {
      printf("\n List is empty");
   }
   else
   {
         while(temp->next->next!=NULL)
         {
            temp=temp->next;
         }
        temp->next=NULL;
   }

}
  void display()
  {
```

```
    struct node *t;
    t=head;
    while(t!=NULL)
    {
        printf("%d\t",t->data);
        t=t->next;
    }
 }
 void delete(int ele)
{
   struct node *t;
   t=head;
   if(t->data==ele)
   {
      head=t->next;
   }
   else
   {
   while(t->next->data!=ele)
   {
      t=t->next;
   }

   t->next=t->next->next;
```

```
 }
}
int main()
{

   do
   {
   int ch,a,pos;
   printf("\n Choose any one operation that you would like to perform\n");
   printf("\n 1.Insert the element at the beginning");
   printf("\n 2.Insert the element at the end");
   printf("\n 3. To insert at the specified position");
   printf("\n 4. To view list");
   printf("\n 5.To view list size");
   printf("\n 6.To delete first element");
   printf("\n 7.To delete last element");
   printf("\n 8.To find next element");
   printf("\n 9. To find previous element");
   printf("\n 10. To find search for an element");
   printf("\n 11. To quit");
   printf("\n Enter your choice\n");
   scanf("%d",&ch);
      switch(ch)
```

```
{
case 1:
printf("\n Insert an element to be inserted at the beginning\n");
scanf("%d",&a);
insertfront(a);
break;
case 2:
 printf("\n Insert an element to be inserted at the End\n");
scanf("%d",&a);
insertend(a);
break;
case 3:
 printf("\n Insert an element and the position to insert in the list\n");
scanf("%d%d",&a,&pos);
insertpos(a,pos);
break;
case 4:
display();
break;
case 5:
listsize();
break;
case 6:
printf("\n Delete an element to be in the beginning\n");
```

```
deleteAtBeginning();
break;
case 7:
printf("\n Delete an element to be at the end\n");
deleteAtEnd();
break;
case  8:
printf("\n enter the element to which you need to find next ele in the list\n");;
scanf("%d",&a);
findnext(a);
break;
case 9:
printf("\n enter the element to which you need to find prev ele in the list\n");;
scanf("%d",&a);
findprev(a);
break;
case 10:
printf("\n enter the element to find the address of it\n");;
scanf("%d",&a);
find(a);
break;
case 11:
printf("Ended");
```

```
            exit(0);
        default:
        printf("Invalid option is chosen so the process is quit");
        }
    }while(1);
 return 0;
}
```

OUTPUT:

```
aiml2315011129@cselab:~$ gcc sll.c
aiml2315011129@cselab:~$ ./a.out

 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
1

 Insert an element to be inserted at the beginning
2

 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
1

 Insert an element to be inserted at the beginning
1

 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
2

 Insert an element to be inserted at the End
3

 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
```

```
 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
2

 Insert an element to be inserted at the End
4

 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
4
1       2       3       4
 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
3

 Insert an element and the position to insert in the list
10 2

 The size of the list is 4:

 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
```

```
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
4
1       10      2       3       4
 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
5

 The size of the list is 5:

 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
6

 Delete an element to be in the beginning

 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
4
10      2       3       4
```

```
10      2       3       4
 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
7

 Delete an element to be at the end

 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
4
10      2       3
 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
8

 enter the element to which you need to find next ele in the list
2

Next Element of 2 is 3

 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
```

```
 enter the element to which you need to find next ele in the list
2

Next Element of 2 is 3

 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
9

 enter the element to which you need to find prev ele in the list
2

 The previous ele of 2 is 10

 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
10

 enter the element to find the address of it
10

 Searching ele 10 is present in the addr of 0x55d6cef30b40
 Choose any one operation that you would like to perform

 1.Insert the element at the beginning
 2.Insert the element at the end
 3. To insert at the specified position
 4. To view list
 5.To view list size
 6.To delete first element
 7.To delete last element
 8.To find next element
 9. To find previous element
 10. To find search for an element
 11. To quit
 Enter your choice
11
Ended
aiml231501129@cselab:~$
```

# Ex 2: Doubly Linked List

Program:

```
#include <stdio.h>
#include<malloc.h>
struct node
{
   int data;
   struct node *next;
   struct node *prev;
};
struct node *newnode;
struct node *head=NULL;

void insertfront(int ele)
{
   newnode=(struct node*)malloc(sizeof(struct node));
  if(head==NULL)
  {
     newnode->data=ele;
     newnode->next=NULL;
     newnode->prev=NULL;
     head=newnode;
  }
  else
```

```
  {
     newnode->data=ele;
     newnode->next=head;
     head->prev=newnode;
     head=newnode;
  }

}

void insertend(int ele)
{
   newnode=(struct node*)malloc(sizeof(struct node));
   newnode->data=ele;
   newnode->next=NULL;
    if(head!=NULL)
    {
       struct node *t;
       t=head;
       while(t->next!=NULL)
       {
          t=t->next;
       }
      newnode->next=NULL;
      t->next=newnode;
```

```
      newnode->prev=t;
    }
    else
    {
      newnode->prev=NULL;
       head=newnode;

    }
  }
int listsize()
  {
     int c=0;
     struct node *t;
     t=head;
     while(t!=NULL)
     {
        c=c+1;
        t=t->next;
     }
     printf("\n The size of the list is %d:\n",c);
     return c;
  }
  void insertpos(int ele,int pos)
  {
```

```
 int ls=0;
 struct node *temp;
 ls=listsize();
 if(head == NULL)
 {
    printf("\nInvalid position to insert a node\n");
    return;
 }
 if(head != NULL && (pos <= 0 || pos > ls))
 {
    printf("\nInvalid position to insert a node\n");
    return;
 }

newnode=(struct node*)malloc(sizeof(struct node));

 if(newnode != NULL)
 {
    newnode->data=ele;
    temp = head;
    int count = 1;
    while(count < pos-1)
    {
        temp = temp -> next;
```

```
            count += 1;
                    }
        if(pos == 1)
        {
            newnode->next = head;
            head->prev=newnode;
            head = newnode;
        }
        else
        {
            newnode->next = temp->next;
            temp->next->prev = newnode;
            temp->next=newnode;
            newnode->prev=temp;
        }
    }
}

void find(int s)
{
    int c=1;
    struct node *temp;
    temp=head;
    if(head==NULL)
```

```
    {
       printf("\n List is empty");
    }
    else
    {


          while(temp->data!=s && temp->next!=NULL)
          {
             temp=temp->next;
             c++;
          }
          if(temp!=NULL && temp->data==s)
          {
      printf("\n Searching ele %d is present in the addr of %p in the pos%d",temp-
>data,temp,c);
    }
    else
    {
       printf("\n Searching elem %d is not present",s);
    }


}
}

void findnext(int s)
```

```
{
   struct node *temp;
   temp=head;
   if(temp==NULL&&temp->next==NULL)
   {
      printf("No next element ");
   }
   else
   {
      while(temp->data!=s)
      {
         temp=temp->next;

      }
            printf("\nNext Element of %d is %d\n",s,temp->next->data);
   }

}

void findprev(int s)
{
   struct node *temp;
   temp=head;
   if(temp==NULL)
```

```
    {
        printf("List is empty ");
    }
    else
    {
        while(temp->data!=s)
        {
            temp=temp->next;


        }
          printf("\n The previous ele of %d is %d\n",s,temp->prev->data);
    }

}
void deleteAtBeginning()
{
    struct node *t;
     // t=head;
      head->next->prev=NULL;
      head=head->next;
}
void deleteAtEnd()
{
    struct node *temp;
```

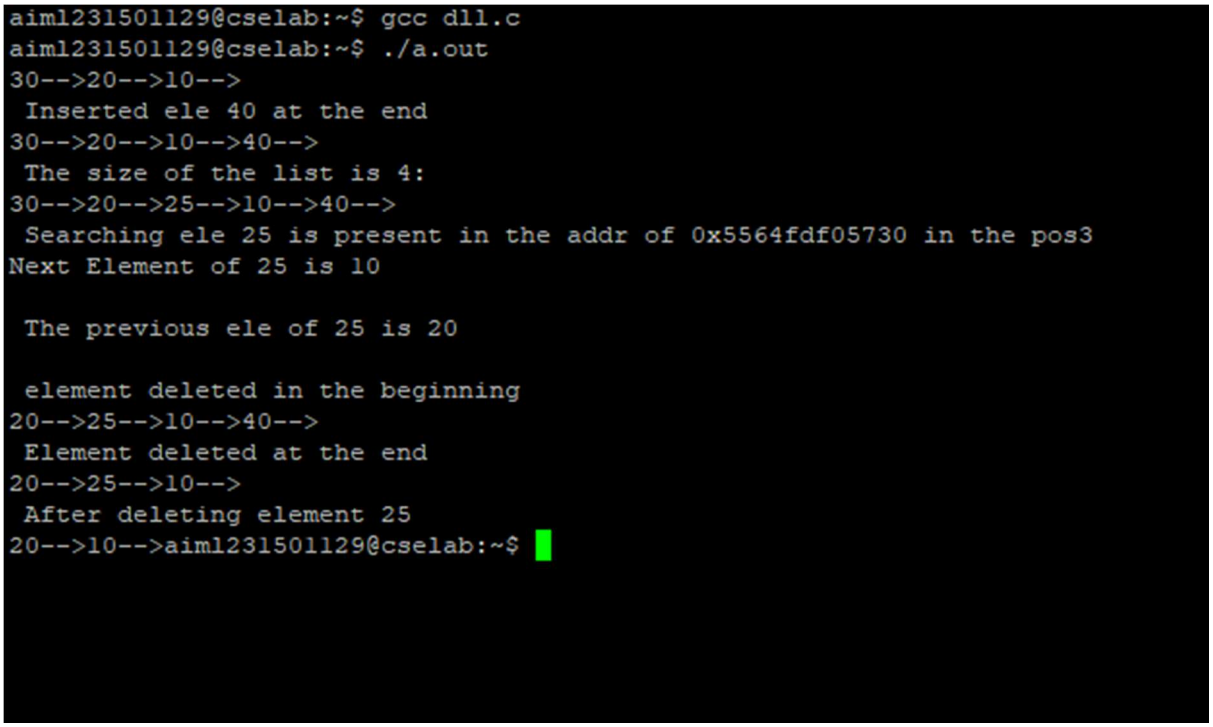
```
    temp=head;
    if(head==NULL)
    {
       printf("\n List is empty");
    }
    else
    {
          while(temp->next!=NULL)
          {
             temp=temp->next;
          }
         temp->prev->next=NULL;
    }

}

void delete(int ele)
{
    struct node *t;
    t=head;
    if(t->data==ele)
    {
       head=t->next;
    }
```

```
    else
    {
    while(t->data!=ele)
    {
        t=t->next;
    }

    t->prev->next=t->next;
    t->next->prev=t->prev;

 }
}


void display()
{
    struct node *temp;
    temp=head;
    while(temp!=NULL)
    {
        printf("%d-->",temp->data);
        temp=temp->next;
    }
}
```

```
int main()
{
   insertfront(10);
   insertfront(20);
   insertfront(30);
   display();
   printf("\n Inserted ele 40 at the end\n");
insertend(40);
display();
insertpos(25,3);
display();
find(25);
findnext(25);
findprev(25);
printf("\n element deleted in the beginning\n");
deleteAtBeginning();
display();
deleteAtEnd();
printf("\n Element deleted at the end\n");
display();
printf("\n After deleting element 25\n");
delete(25);
```

```
display();
    return 0;
}
```

OUTPUT:



# Ex 3: Polynomial Addition

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int coeff;
   int expo;
   struct node *next;
};

struct node* insert(struct node *head,int co,int exp)
{
   struct node *temp;
   struct node *newnode=malloc(sizeof(struct node));
   newnode->coeff=co;
   newnode->expo=exp;
   newnode->next=NULL;

   if(head==NULL || exp>head->expo)
   {
      newnode->next=head;
      head=newnode;
   }
   else
   {
```

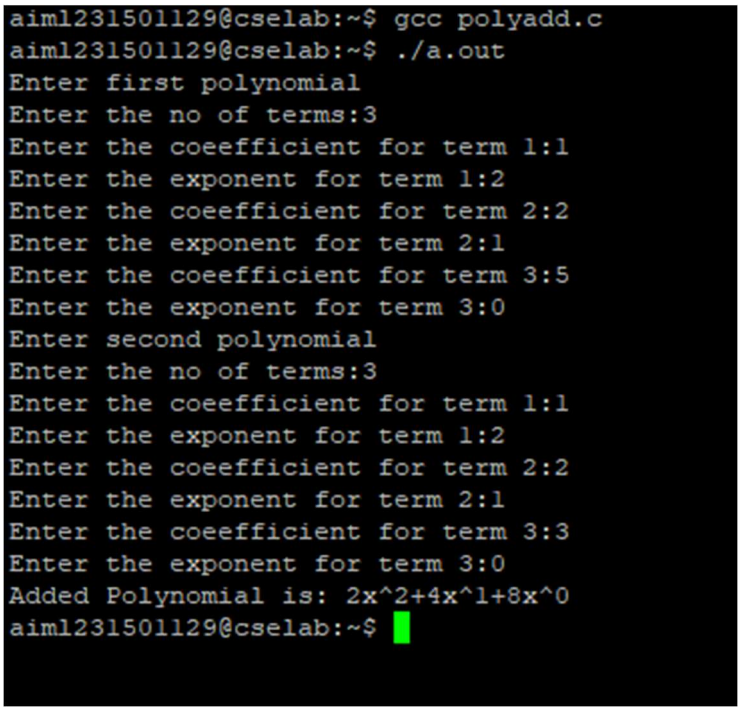
```
        temp=head;
        while(temp->next!=NULL &&temp->next->expo>=exp)
            temp=temp->next;
        newnode->next=temp->next;
        temp->next=newnode;
    }
    return head;
}
struct node* create(struct node *head)
{
    int n,i;
    int coeff;
    int expo;
    printf("Enter the no of terms:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the coeefficient for term %d:",i+1);
        scanf("%d",&coeff);

        printf("Enter the exponent for term %d:",i+1);
        scanf("%d",&expo);

        head=insert(head,coeff,expo);
    }
    return head;
}
```

```
void print(struct node* head)
{
   if(head==NULL)
      printf("No Polynomial");
   else
   {
      struct node *temp=head;
      while(temp!=NULL)
      {
         printf("%dx^%d",temp->coeff,temp->expo);
         temp=temp->next;
         if(temp!=NULL)
            printf("+");
         else
            printf("\n");
      }
   }
}

void polyAdd(struct node *head1, struct node *head2)
{
   struct node *ptr1=head1;
   struct node *ptr2=head2;
   struct node *head3=NULL;
   while(ptr1!=NULL && ptr2!=NULL)
   {
      if(ptr1->expo == ptr2->expo)
```

```
    {
    head3=insert(head3,ptr1->coeff+ptr2->coeff,ptr1->expo);
       ptr1=ptr1->next;
       ptr2=ptr2->next;
    }
    else if(ptr1->expo > ptr2->expo)
    {
       head3=insert(head3,ptr1->coeff,ptr1->expo);
       ptr1=ptr1->next;
    }
    else if(ptr1->expo < ptr2->expo)
    {
       head3=insert(head3,ptr2->coeff,ptr2->expo);
       ptr2=ptr2->next;
    }
 }
 while(ptr1!=NULL)
 {
    head3=insert(head3,ptr1->coeff,ptr1->expo);
    ptr1=ptr1->next;
 }
 while(ptr2!=NULL)
 {
    head3=insert(head3,ptr2->coeff,ptr2->expo);
    ptr2=ptr2->next;
 }
printf("Added Polynomial is: ") ;
```

```
    print(head3);
}
int main()
{
    struct node *head1=NULL;
    struct node *head2=NULL;
    printf("Enter first polynomial\n");
    head1=create(head1);
    printf("Enter second polynomial\n");
    head2=create(head2);
    polyAdd(head1,head2);
    return 0;

}
```

OUTPUT:



# Ex 4: Stack Implementation

PROGRAM :

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
int Data;
struct Node *next;
}*top;
void popStack()
{
struct Node *temp, *var=top;
if(var==top)
{
top = top->next;
free(var);
}
else
printf("\nStack Empty");
}
void push(int value)
{
struct Node *temp;
temp=(struct Node *)malloc(sizeof(struct Node));
temp->Data=value;
if (top == NULL)
```

```
{
top=temp;
top->next=NULL;
}
else
{
temp->next=top;
top=temp;
}
}
void display()
{
struct Node *var=top;
if(var!=NULL)
{
printf("\nElements are as:\n");
while(var!=NULL)
{
printf("\t%d\n",var->Data);
var=var->next;
}
printf("\n");
}
else
printf("\nStack is Empty");
}
int main()
```

```
{
int i=0;
top=NULL;
clrscr();
printf(" \n1. Push to stack");
printf(" \n2. Pop from Stack");
printf(" \n3. Display data of Stack");
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 Stack: ");
scanf("%d",&value);
push(value);
break;
}
case 2:
{
popStack();
printf("\n The last element is popped");
break;
```

```
}
case 3:
{
display();
break;
}
case 4:
{
struct Node *temp;
while(top!=NULL)
{
temp = top->next;
free(top);
top=temp;
}
exit(0);
}
default:
{
printf("\nwrong choice for operation");
}}}}
```

OUTPUT:

```
aiml2315011129@cselab:~$ gcc ex4.c
aiml2315011129@cselab:~$ ./a.out

1. Push to stack
2. Pop from Stack
3. Display data of Stack
4. Exit

Choose Option: 1

Enter a value to push into Stack: 1

Choose Option: 1

Enter a value to push into Stack: 1

Choose Option:
4
aiml2315011129@cselab:~$ gcc ex4.c
aiml2315011129@cselab:~$ ./a.out

1. Push to stack
2. Pop from Stack
3. Display data of Stack
4. Exit

Choose Option: 1

Enter a value to push into Stack: 1

Choose Option: 1

Enter a value to push into Stack: 2

Choose Option: 1

Enter a value to push into Stack: 3

Choose Option: 1

Enter a value to push into Stack: 4

Choose Option: 1

Enter a value to push into Stack: 5

Choose Option: 2

 The last element is popped
Choose Option: 3

Elements are as:
        4
        3
        2
        1


Choose Option: 4
aiml2315011129@cselab:~$
```

# Ex 5: Infix to Postfix

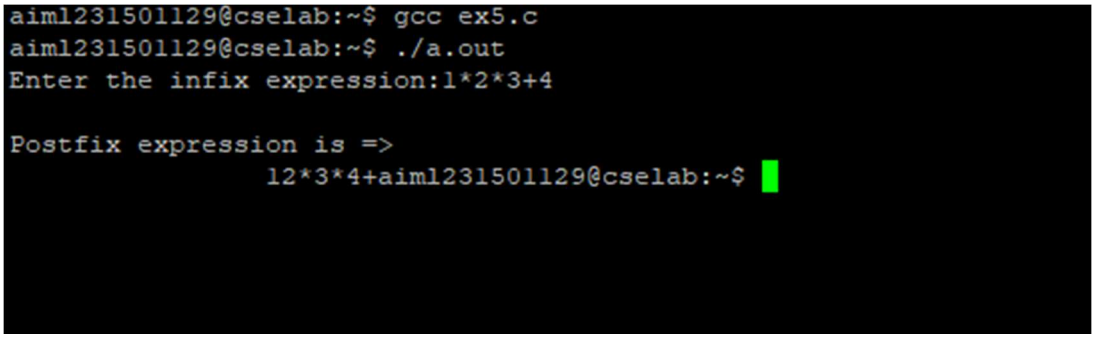
PROGRAM:

```
#include<stdio.h>
#include<malloc.h>
int top=0,st[20];
char inf[40],post[40];
void postfix();
void push(int);
char pop();
void main()
{
printf("Enter the infix expression:");
scanf("%s",inf);
postfix();
}
void postfix()
{int i,j=0;
for(i=0;inf[i]!=0;i++)
{switch(inf[i])
{
case '+':while(st[top]>=1)
post[j++]=pop();
push(1);
break;
case '-':while(st[top]>=1)
post[j++]=pop();
```

```
push(2);
break;
case '*':while(st[top]>=3)
post[j++]=pop();
push(3);
break;
case '/':while(st[top]>=4)
post[j++]=pop();
push(4);
break;
case '^':
post[j++]=pop();
push(5);
break;
case '(':push(0);
break;
case ')':while(st[top]!=0)
post[j++]=pop();
top--;
break;
default:
post[j++]=inf[i];
}}
while(top>0)
post[j++]=pop();
printf("\nPostfix expression is =>\n\t\t%s",post);
}void push(int ele)
```

```
{
top++;
st[top]=ele;
}char pop()
{int el;
char e;
el=st[top];
top--;
switch(el)
{case 1:
e='+';
break;
case 2:
e='-';
break;
case 3:
e='*';
break;
case 4:
e='/';
break;
case 5:
e='^';
break;
}return(e);
}
```

OUTPUT:


aiml231501129@cselab:~$ gcc ex5.c
aiml231501129@cselab:~$ ./a.out
Enter the infix expression:1*2*3+4
Postfix expression is =>
12*3*4+aiml231501129@cselab:~$

# Ex 6: Evaluation Arithmetic Expression

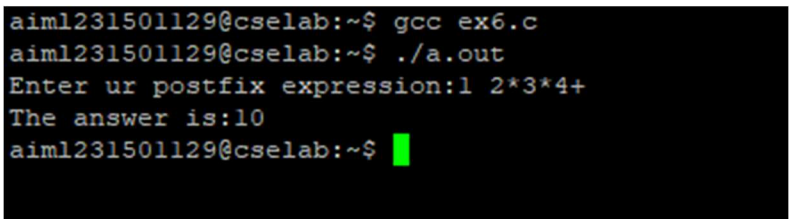
PROGRAM:

```
#include <stdio.h>

#include <string.h>

#include <ctype.h>

int top = -1;

int stack[100];

void push (int data) {

stack[++top] = data;

}

int pop () {

int data;

if (top == -1)

return -1;

data = stack[top];

stack[top] = 0;

top--;

return (data);

}

int main()

{

char str[100];

int i, data = -1, operand1, operand2, result;

printf("Enter ur postfix expression:");

fgets(str, 100, stdin);

for (i = 0; i < strlen(str); i++){
```

```
if(isdigit(str[i]))
{
data = (data == -1) ? 0 : data;
data = (data * 10) + (str[i] - 48);
continue;
}
if (data != -1){
   push(data);
}
if (str[i] == '+' || str[i] == '-'|| str[i] =='*'|| str[i] == '/'){
operand2 = pop();
operand1 = pop();
if (operand1 == -1 || operand2 == -1)
break;
switch (str[i])
{
case '+':
result = operand1 + operand2;
push(result);
break;
case '-':
result = operand1 - operand2;
push(result);
break;
case '*':
result = operand1 * operand2;
push(result);
```

```
break;
case '/':
result = operand1 / operand2;
push(result);
break;
}
}
data = -1;
}
if (top == 0)
printf("The answer is:%d\n", stack[top]);
else
printf("u have given wrong postfix expression\n");
return 0;
}
```

OUTPUT:



# Ex 7: Implementation of Queue using Linked List

PROGRAM :

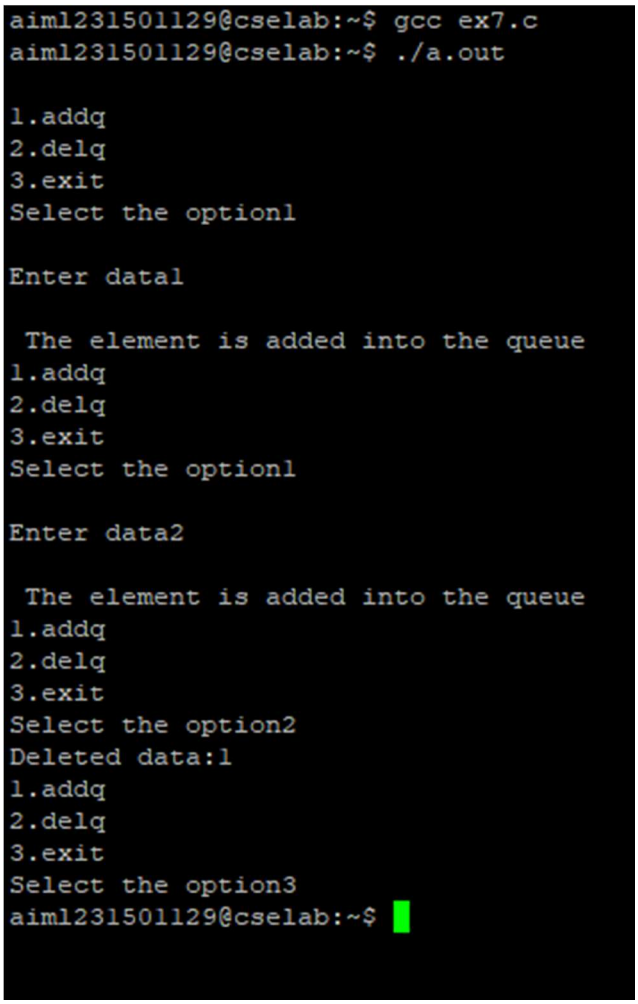
```
#include<stdio.h >
#include<malloc.h >
struct queue
{
int data;
struct queue *next;
};
struct queue *addq(struct queue *front);
struct queue *delq(struct queue *front);
void main()
{
struct queue *front;
int reply,option,data;
front=NULL;
do
{
printf("\n1.addq");
printf("\n2.delq");
printf("\n3.exit");
printf("\nSelect the option");
scanf("%d",&option);
switch(option)
{
```

```
case 1 : //addq
front=addq(front);
printf("\n The element is added into the queue");
break;
case 2 : //delq
front=delq(front);
break;
case 3 : exit(0);
}
}while(1);
}
struct queue *addq(struct queue *front)
{
struct queue *c,*r;
//create new node
c=(struct queue*)malloc(sizeof(struct queue));
if(c==NULL)
{
printf("Insufficient memory");
return(front);
}
//read an insert value from console
printf("\nEnter data");
scanf("%d",&c->data);
c->next=NULL;
if(front==NULL)
{
```

```
front=c;
}
else
{
//insert new node after last node
r=front;
while(r->next!=NULL)
{
r=r->next;
}}
return(front);
}
struct queue *delq(struct queue *front)
{
struct queue *c;
if(front==NULL)
{
printf("Queue is empty");
return(front);
}
//print the content of first node
printf("Deleted data:%d",front->data);
//delete first node
c=front;
front=front->next;
free(c);
return(front);
```

}

OUTPUT:


aiml2315011129@cselab:~$ gcc ex7.c
aiml2315011129@cselab:~$ ./a.out
1.addq
2.delq
3.exit
Select the option1
Enter data1
The element is added into the queue
1.addq
2.delq
3.exit
Select the option1
Enter data2
The element is added into the queue
1.addq
2.delq
3.exit
Select the option2
Deleted data:1
1.addq
2.delq
3.exit
Select the option3
aiml2315011129@cselab:~$

# Ex 8: Tree Traversal

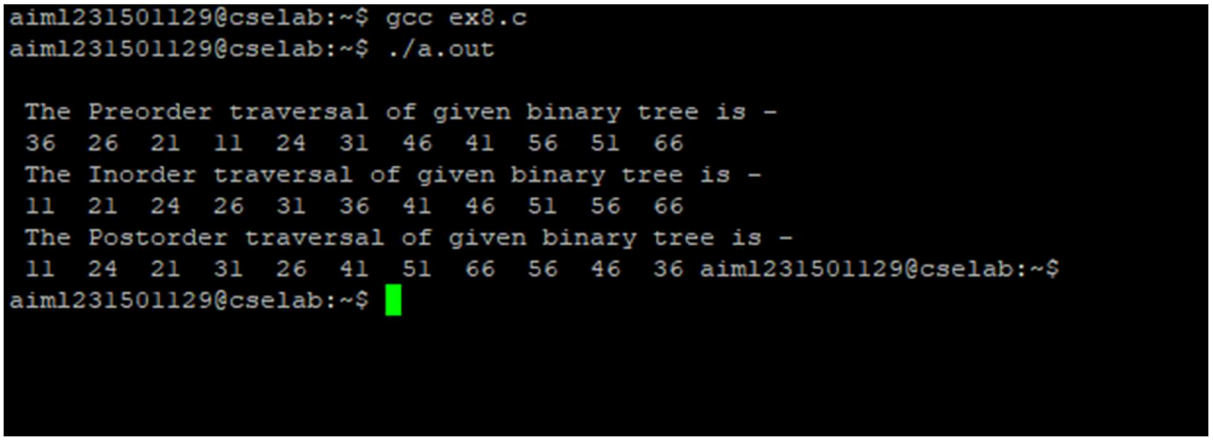
PROGRAM :

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


struct node {
   int element;
   struct node* left;
   struct node* right;
};
struct node* createNode(int val)
{
   struct node* Node = (struct node*)malloc(sizeof(struct node));
   Node->element = val;
   Node->left = NULL;
   Node->right = NULL;


   return (Node);
}
void traversePreorder(struct node* root)
{
   if (root == NULL)
      return;
   printf(" %d ", root->element);
   traversePreorder(root->left);
   traversePreorder(root->right);
```

```
}

void traverseInorder(struct node* root)
{
   if (root == NULL)
      return;
   traverseInorder(root->left);
   printf(" %d ", root->element);
   traverseInorder(root->right);
}

void traversePostorder(struct node* root)
{
   if (root == NULL)
      return;
   traversePostorder(root->left);
   traversePostorder(root->right);
   printf(" %d ", root->element);
}


int main()
{
   struct node* root = createNode(36);
   root->left = createNode(26);
   root->right = createNode(46);
   root->left->left = createNode(21);
```

```
    root->left->right = createNode(31);
    root->left->left->left = createNode(11);
    root->left->left->right = createNode(24);
    root->right->left = createNode(41);
    root->right->right = createNode(56);
    root->right->right->left = createNode(51);
    root->right->right->right = createNode(66);

    printf("\n The Preorder traversal of given binary tree is -\n");
    traversePreorder(root);

    printf("\n The Inorder traversal of given binary tree is -\n");
    traverseInorder(root);

    printf("\n The Postorder traversal of given binary tree is -\n");
    traversePostorder(root);

    return 0;
}
```

OUTPUT:



# Ex 9: Binary Search Tree

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
struct BinaryTreeNode {
   int key;
   struct BinaryTreeNode *left, *right;
};


struct BinaryTreeNode* newNodeCreate(int value)
{
   struct BinaryTreeNode* temp
      = (struct BinaryTreeNode*)malloc(
         sizeof(struct BinaryTreeNode));
   temp->key = value;
   temp->left = temp->right = NULL;
   return temp;
}
struct BinaryTreeNode*
searchNode(struct BinaryTreeNode* root, int target)
{
   if (root == NULL || root->key == target) {
      return root;
   }
   if (root->key < target) {
      return searchNode(root->right, target);
```

```
    }
    return searchNode(root->left, target);
}
struct BinaryTreeNode*
insertNode(struct BinaryTreeNode* node, int value)
{
    if (node == NULL) {
        return newNodeCreate(value);
    }
    if (value < node->key) {
        node->left = insertNode(node->left, value);
    }
    else if (value > node->key) {
        node->right = insertNode(node->right, value);
    }
    return node;
}
void postOrder(struct BinaryTreeNode* root)
{
    if (root != NULL) {
        postOrder(root->left);
        postOrder(root->right);
        printf(" %d ", root->key);
    }
}
void inOrder(struct BinaryTreeNode* root)
{
```

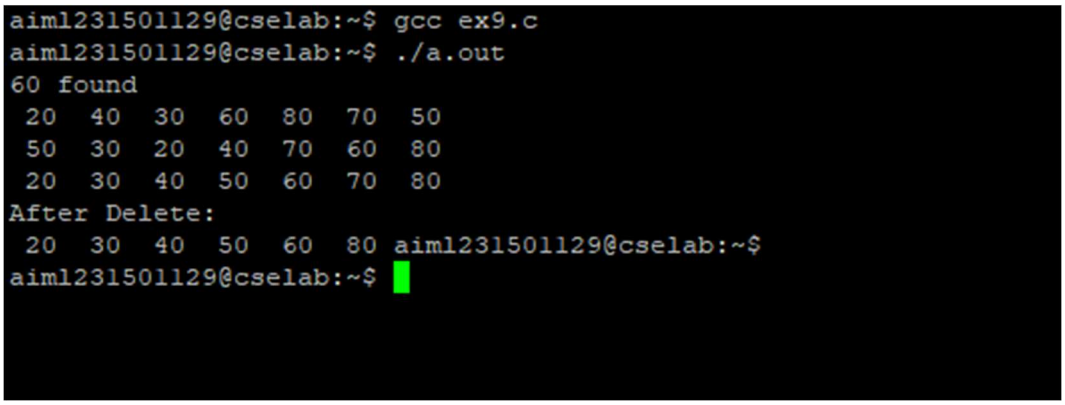
```
    if (root != NULL) {
        inOrder(root->left);
        printf(" %d ", root->key);
        inOrder(root->right);
    }
}
void preOrder(struct BinaryTreeNode* root)
{
    if (root != NULL) {
        printf(" %d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}
struct BinaryTreeNode* findMin(struct BinaryTreeNode* root)
{
    if (root == NULL) {
        return NULL;
    }
    else if (root->left != NULL) {
        return findMin(root->left);
    }
    return root;
}
struct BinaryTreeNode* delete (struct BinaryTreeNode* root,
                               int x)
{
```

```
if (root == NULL)
   return NULL;


if (x > root->key) {
   root->right = delete (root->right, x);
}
else if (x < root->key) {
   root->left = delete (root->left, x);
}
else {
   if (root->left == NULL && root->right == NULL) {
      free(root);
      return NULL;
   }
   else if (root->left == NULL
         || root->right == NULL) {
      struct BinaryTreeNode* temp;
      if (root->left == NULL) {
         temp = root->right;
      }
      else {
         temp = root->left;
      }
      free(root);
      return temp;
   }
   else {
```

```
            struct BinaryTreeNode* temp
               = findMin(root->right);
            root->key = temp->key;
            root->right = delete (root->right, temp->key);
        }
    }
    return root;
}

int main()
{
    struct BinaryTreeNode* root = NULL;
    root = insertNode(root, 50);
    insertNode(root, 30);
    insertNode(root, 20);
    insertNode(root, 40);
    insertNode(root, 70);
    insertNode(root, 60);
    insertNode(root, 80);
    if (searchNode(root, 60) != NULL) {
        printf("60 found");
    }
    else {
        printf("60 not found");
    }

    printf("\n");
```

```
    postOrder(root);
    printf("\n");
    preOrder(root);
    printf("\n");
    inOrder(root);
    printf("\n");
    struct BinaryTreeNode* temp = delete (root, 70);
    printf("After Delete: \n");
    inOrder(root);


    return 0;
}
```

OUTPUT:



# Ex 10: AVL Tree

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node* left;
   struct node* right;
   int ht;
};
struct node* root = NULL;
struct node* create(int);
struct node* insert(struct node*, int);
struct node* delete(struct node*, int);
struct node* search(struct node*, int);
struct node* rotate_left(struct node*);
struct node* rotate_right(struct node*);
int balance_factor(struct node*);
int height(struct node*);
void inorder(struct node*);
void preorder(struct node*);
void postorder(struct node*);

int main()
{
```

```
int user_choice, data;
char user_continue = 'y';
struct node* result = NULL;

while (user_continue == 'y' || user_continue == 'Y')
{
   printf("\n\n------- AVL TREE --------\n");
   printf("\n1. Insert");
   printf("\n2. Delete");
   printf("\n3. Search");
   printf("\n4. Inorder");
   printf("\n5. Preorder");
   printf("\n6. Postorder");
   printf("\n7. EXIT");

   printf("\n\nEnter Your Choice: ");
   scanf("%d", &user_choice);

   switch(user_choice)
   {
      case 1:
         printf("\nEnter data: ");
         scanf("%d", &data);
         root = insert(root, data);
         break;

      case 2:
```

```
    printf("\nEnter data: ");
    scanf("%d", &data);
    root = delete(root, data);
    break;

case 3:
    printf("\nEnter data: ");
    scanf("%d", &data);
    result = search(root, data);
    if (result == NULL)
    {
        printf("\nNode not found!");
    }
    else
    {
        printf("\n Node found");
    }
    break;
case 4:
    inorder(root);
    break;

case 5:
    preorder(root);
    break;

case 6:
```

```
            postorder(root);
            break;

        case 7:
            printf("\n\tProgram Terminated\n");
            return 1;

        default:
            printf("\n\tInvalid Choice\n");
      }

      printf("\n\nDo you want to continue? ");
      scanf(" %c", &user_continue);
   }

   return 0;
}
struct node* create(int data)
{
   struct node* new_node = (struct node*) malloc (sizeof(struct node));
   if (new_node == NULL)
   {
      printf("\nMemory can't be allocated\n");
      return NULL;
   }
   new_node->data = data;
   new_node->left = NULL;
```

```
    new_node->right = NULL;
    return new_node;
}
struct node* rotate_left(struct node* root)
{
    struct node* right_child = root->right;
    root->right = right_child->left;
    right_child->left = root;
    root->ht = height(root);
    right_child->ht = height(right_child);
    return right_child;
}
struct node* rotate_right(struct node* root)
{
    struct node* left_child = root->left;
    root->left = left_child->right;
    left_child->right = root;
    root->ht = height(root);
    left_child->ht = height(left_child);
    return left_child;
}
int balance_factor(struct node* root)
{
    int lh, rh;
    if (root == NULL)
        return 0;
    if (root->left == NULL)
```

```
        lh = 0;
    else
        lh = 1 + root->left->ht;
    if (root->right == NULL)
        rh = 0;
    else
        rh = 1 + root->right->ht;
    return lh - rh;
}
int height(struct node* root)
{
    int lh, rh;
    if (root == NULL)
    {
        return 0;
    }
    if (root->left == NULL)
        lh = 0;
    else
        lh = 1 + root->left->ht;
    if (root->right == NULL)
        rh = 0;
    else
        rh = 1 + root->right->ht;

    if (lh > rh)
        return (lh);
```

```
    return (rh);
}
struct node* insert(struct node* root, int data)
{
    if (root == NULL)
    {
        struct node* new_node = create(data);
        if (new_node == NULL)
        {
            return NULL;
        }
        root = new_node;
    }
    else if (data > root->data)
    {
        root->right = insert(root->right, data);
        if (balance_factor(root) == -2)
        {
            if (data > root->right->data)
            {
                root = rotate_left(root);
            }
            else
            {
                root->right = rotate_right(root->right);
                root = rotate_left(root);
            }
```

```
        }
    }
    else
    {
        root->left = insert(root->left, data);
        if (balance_factor(root) == 2)
        {
            if (data < root->left->data)
            {
                root = rotate_right(root);
            }
            else
            {
                root->left = rotate_left(root->left);
                root = rotate_right(root);
            }
        }
    }
    root->ht = height(root);
    return root;
}
struct node * delete(struct node *root, int x)
{
    struct node * temp = NULL;

    if (root == NULL)
    {
```

```
    return NULL;
}

if (x > root->data)
{
    root->right = delete(root->right, x);
    if (balance_factor(root) == 2)
    {
        if (balance_factor(root->left) >= 0)
        {
            root = rotate_right(root);
        }
        else
        {
            root->left = rotate_left(root->left);
            root = rotate_right(root);
        }
    }
}
else if (x < root->data)
{
    root->left = delete(root->left, x);
    if (balance_factor(root) == -2)
    {
        if (balance_factor(root->right) <= 0)
        {
            root = rotate_left(root);
```

```
        }
        else
        {
            root->right = rotate_right(root->right);
            root = rotate_left(root);
        }
    }
}
else
{
    if (root->right != NULL)
    {
        temp = root->right;
        while (temp->left != NULL)
            temp = temp->left;

        root->data = temp->data;
        root->right = delete(root->right, temp->data);
        if (balance_factor(root) == 2)
        {
            if (balance_factor(root->left) >= 0)
            {
                root = rotate_right(root);
            }
            else
            {
                root->left = rotate_left(root->left);
```

```
                root = rotate_right(root);
            }
          }
       }
       else
       {
          return (root->left);
       }
    }
    root->ht = height(root);
    return (root);
}
struct node* search(struct node* root, int key)
{
    if (root == NULL)
    {
       return NULL;
    }

    if(root->data == key)
    {
       return root;
    }

    if(key > root->data)
    {
       search(root->right, key);
```

```
    }
    else
    {
        search(root->left, key);
    }
}
void inorder(struct node* root)
{
    if (root == NULL)
    {
        return;
    }

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

    printf("%d ", root->data);
    preorder(root->left);
    preorder(root->right);
```

```
}
void postorder(struct node* root)
{
   if (root == NULL)
   {
      return;
   }

   postorder(root->left);
   postorder(root->right);
   printf("%d ", root->data);
}
```

OUTPUT:

```
aiml231501129@cselab:~$ gcc ex10.c
aiml231501129@cselab:~$ ./a.out


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. Inorder
5. Preorder
6. Postorder
7. EXIT

Enter Your Choice: 1

Enter data: 1


Do you want to continue? y


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. Inorder
5. Preorder
6. Postorder
7. EXIT

Enter Your Choice: 1

Enter data: 2


Do you want to continue? y


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. Inorder
5. Preorder
6. Postorder
7. EXIT

Enter Your Choice: 1

Enter data: 3


Do you want to continue? y


------- AVL TREE --------
```

```
------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. Inorder
5. Preorder
6. Postorder
7. EXIT

Enter Your Choice: 1

Enter data: 4


Do you want to continue? y


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. Inorder
5. Preorder
6. Postorder
7. EXIT

Enter Your Choice: 2

Enter data: 4


Do you want to continue? y


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. Inorder
5. Preorder
6. Postorder
7. EXIT

Enter Your Choice: 4
1 2 3

Do you want to continue? y


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. Inorder
5. Preorder
6. Postorder
7. EXIT
```

```
1. Insert
2. Delete
3. Search
4. Inorder
5. Preorder
6. Postorder
7. EXIT

Enter Your Choice: 3

Enter data: 2

 Node found

Do you want to continue? y


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. Inorder
5. Preorder
6. Postorder
7. EXIT

Enter Your Choice: 5
2 1 3

Do you want to continue? y


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. Inorder
5. Preorder
6. Postorder
7. EXIT

Enter Your Choice: 6
1 3 2

Do you want to continue? y


------- AVL TREE --------

1. Insert
2. Delete
3. Search
4. Inorder
5. Preorder
6. Postorder
7. EXIT

Enter Your Choice: 7

        Program Terminated
aiml231501129@cselab:~$
```

# Ex 11: Topological Sorting

PROGRAM:

```
#include<stdio.h>

#include<stdlib.h>


int s[100], j, res[100];


void AdjacencyMatrix(int a[][100], int n) {


   int i, j;

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

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

         a[i][j] = 0;

      }

   }

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

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

         a[i][j] = rand() % 2;

         a[j][i] = 0;

      }

   }

}


void dfs(int u, int n, int a[][100]) {


   int v;
```

```
    s[u] = 1;
    for (v = 0; v < n - 1; v++) {
        if (a[u][v] == 1 && s[v] == 0) {
            dfs(v, n, a);
        }
    }
    j += 1;
    res[j] = u;
}

void topological_order(int n, int a[][100]) {

    int i, u;
    for (i = 0; i < n; i++) {
        s[i] = 0;
    }
    j = 0;
    for (u = 0; u < n; u++) {
        if (s[u] == 0) {
            dfs(u, n, a);
        }
    }
    return;
}
int main() {
    int a[100][100], n, i, j;
```

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

    AdjacencyMatrix(a, n);

    printf("\t\tAdjacency Matrix of the graph\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            printf("\t%d", a[i][j]);
        }
        printf("\n");
    }
    printf("\nTopological order:\n");

    topological_order(n, a);

    for (i = n; i >= 1; i--) {
        printf("-->%d", res[i]);
    }
    return 0;
}
```

OUTPUT:

```
aiml231501129@cselab:~$ gcc ex12.c
aiml231501129@cselab:~$ ./a.out
Enter number of vertices
2
                Adjacency Matrix of the graph
        0       0
        1       0

Topological order:
-->1-->0aiml231501129@cselab:~$
```

# Ex 12: BFS AND DFS

BREATH FIRST SEARCH:

PROGRAM:

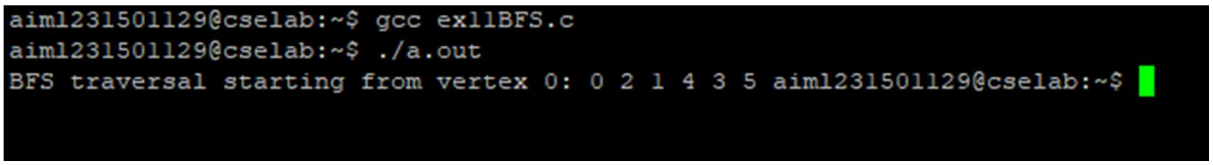
```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int vertex;
   struct node* next;
};
struct adj_list {
   struct node* head;
};
struct graph {
   int num_vertices;
   struct adj_list* adj_lists;
   int* visited;
};
struct node* new_node(int vertex) {
   struct node* new_node = (struct node*)malloc(sizeof(struct node));
new_node->vertex = vertex;
new_node->next = NULL;
   return new_node;
}
struct graph* create_graph(int n) {
   struct graph* graph = (struct graph*)malloc(sizeof(struct graph));
   graph->num_vertices = n;
```

```
    graph->adj_lists = (struct adj_list*)malloc(n * sizeof(struct adj_list));
    graph->visited = (int*)malloc(n * sizeof(int));

    int i;
    for (i = 0; i< n; i++) {
        graph->adj_lists[i].head = NULL;
        graph->visited[i] = 0;
    }

    return graph;
}
void add_edge(struct graph* graph, int src, int dest) {
    struct node* new_node1 = new_node(dest);
    new_node1->next = graph->adj_lists[src].head;
    graph->adj_lists[src].head = new_node1;
    struct node* new_node2 = new_node(src);
    new_node2->next = graph->adj_lists[dest].head;
    graph->adj_lists[dest].head = new_node2;
}
void bfs(struct graph* graph, int v) {
    int queue[1000];
    int front = -1;
    int rear = -1;
    graph->visited[v] = 1;
    queue[++rear] = v;
    while (front != rear) {
        int current_vertex = queue[++front];
```

```
printf("%d ", current_vertex);
      struct node* temp = graph->adj_lists[current_vertex].head;
      while (temp != NULL) {
         int adj_vertex = temp->vertex;
         if (graph->visited[adj_vertex] == 0) {
            graph->visited[adj_vertex] = 1;
            queue[++rear] = adj_vertex;
         }

         temp = temp->next;
      }
   }
}

int main() {
   struct graph* graph = create_graph(6);



add_edge(graph, 0, 1);
add_edge(graph, 0, 2);
add_edge(graph, 1, 3);
add_edge(graph, 1, 4);
add_edge(graph, 2, 4);
add_edge(graph, 3, 4);
add_edge(graph, 3, 5);
add_edge(graph, 4,5);
```

```
printf("BFS traversal starting from vertex 0: ");
bfs(graph, 0);

    return 0;
}
```

OUTPUT:




DEPTH FIRST SEARCH:

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
int vis[100];
struct Graph {
    int V;
    int E;
    int** Adj;
};
struct Graph* adjMatrix()
{
    struct Graph* G = (struct Graph*)
        malloc(sizeof(struct Graph));
    if (!G) {
        printf("Memory Error\n");
        return NULL;
```

```
    }
    G->V = 7;
    G->E = 7;

    G->Adj = (int**)malloc((G->V) * sizeof(int*));
    for (int k = 0; k < G->V; k++) {
        G->Adj[k] = (int*)malloc((G->V) * sizeof(int));
    }

    for (int u = 0; u < G->V; u++) {
        for (int v = 0; v < G->V; v++) {
            G->Adj[u][v] = 0;
        }
    }
    G->Adj[0][1] = G->Adj[1][0] = 1;
    G->Adj[0][2] = G->Adj[2][0] = 1;
    G->Adj[1][3] = G->Adj[3][1] = 1;
    G->Adj[1][4] = G->Adj[4][1] = 1;
    G->Adj[1][5] = G->Adj[5][1] = 1;
    G->Adj[1][6] = G->Adj[6][1] = 1;
    G->Adj[6][2] = G->Adj[2][6] = 1;

    return G;
}
void DFS(struct Graph* G, int u)
{
    vis[u] = 1;
```

```
    printf("%d ", u);
    for (int v = 0; v < G->V; v++) {
        if (!vis[v] && G->Adj[u][v]) {
            DFS(G, v);
        }
    }
}
void DFStraversal(struct Graph* G)
{
    for (int i = 0; i < 100; i++) {
        vis[i] = 0;
    }
    for (int i = 0; i < G->V; i++) {
        if (!vis[i]) {
            DFS(G, i);
        }
    }
}
void main()
{
    struct Graph* G;
    G = adjMatrix();
    DFStraversal(G);
}
```

OUTPUT:

```
aiml231501129@cselab:~$ gcc ex11DFS.c
aiml231501129@cselab:~$ ./a.out
0 1 3 4 5 6 2 aiml231501129@cselab:~$
```

# Ex 13: Prims Algorithm

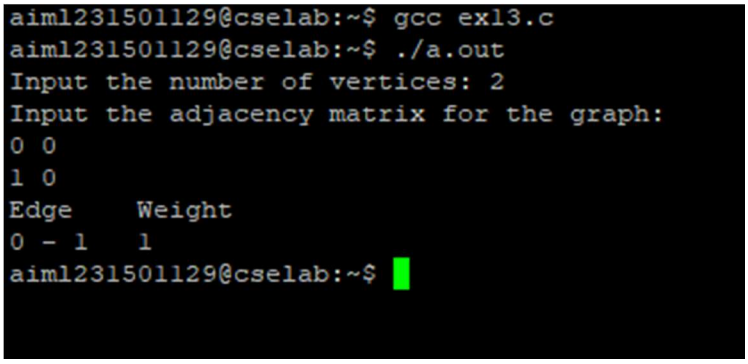
PROGRAM:

```
#include <stdio.h>
#include <limits.h>
#define MAX_VERTICES 100
int minKey(int key[], int mstSet[], int vertices) {
    int min = INT_MAX, minIndex;

    for (int v = 0; v < vertices; v++) {
        if (!mstSet[v] && key[v] < min) {
            min = key[v];
            minIndex = v;
        }
    }

    return minIndex;
}
void printMST(int parent[], int graph[MAX_VERTICES][MAX_VERTICES], int vertices) {
    printf("Edge \tWeight\n");
    for (int i = 1; i < vertices; i++) {
        printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);
    }
}
void primMST(int graph[MAX_VERTICES][MAX_VERTICES], int vertices) {
    int parent[MAX_VERTICES];
    int key[MAX_VERTICES];
```

```
    int mstSet[MAX_VERTICES];
    for (int i = 0; i < vertices; i++) {
        key[i] = INT_MAX;
        mstSet[i] = 0;
    }
    key[0] = 0;
    parent[0] = -1;
    for (int count = 0; count < vertices - 1; count++) {
        int u = minKey(key, mstSet, vertices);
        mstSet[u] = 1;
        for (int v = 0; v < vertices; v++) {
            if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }
    printMST(parent, graph, vertices);
}

int main() {
    int vertices;
    printf("Input the number of vertices: ");
    scanf("%d", &vertices);

    if (vertices <= 0 || vertices > MAX_VERTICES) {
        printf("Invalid number of vertices. Exiting...\n");
```

```
        return 1;
    }

    int graph[MAX_VERTICES][MAX_VERTICES];
    printf("Input the adjacency matrix for the graph:\n");
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    primMST(graph, vertices);

    return 0;
}
```

OUTPUT:



# Ex 14: Dijkstra Algorithm

PROGRAM :

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


#define MAX_VERTICES 100
int minDistance(int dist[], int sptSet[], int vertices) {
   int min = INT_MAX, minIndex;


   for (int v = 0; v < vertices; v++) {
      if (!sptSet[v] && dist[v] < min) {
         min = dist[v];
         minIndex = v;
      }
   }


   return minIndex;
}
void printSolution(int dist[], int vertices) {
   printf("Vertex \tDistance from Source\n");
   for (int i = 0; i < vertices; i++) {
      printf("%d \t%d\n", i, dist[i]);
   }
}
void dijkstra(int graph[MAX_VERTICES][MAX_VERTICES], int src, int vertices) {
   int dist[MAX_VERTICES];
```

```
    int sptSet[MAX_VERTICES];
    for (int i = 0; i < vertices; i++) {
        dist[i] = INT_MAX;
        sptSet[i] = 0;
    }
    dist[src] = 0;
    for (int count = 0; count < vertices - 1; count++) {
        int u = minDistance(dist, sptSet, vertices);
        sptSet[u] = 1;
        for (int v = 0; v < vertices; v++) {
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }
    printSolution(dist, vertices);
}

int main() {
    int vertices;
    printf("Input the number of vertices: ");
    scanf("%d", &vertices);

    if (vertices <= 0 || vertices > MAX_VERTICES) {
        printf("Invalid number of vertices. Exiting...\n");
        return 1;
    }
```

```
    int graph[MAX_VERTICES][MAX_VERTICES];
    printf("Input the adjacency matrix for the graph (use INT_MAX for infinity):\n");
    for (int i = 0; i < vertices; i++) {
        for (int j = 0; j < vertices; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    int source;
    printf("Input the source vertex: ");
    scanf("%d", &source);

    if (source < 0 || source >= vertices) {
        printf("Invalid source vertex. Exiting...\n");
        return 1;
    }
    dijkstra(graph, source, vertices);

    return 0;
}
```

OUTPUT:

```
aiml2315011129@cselab:~$ gcc ex14.c
aiml2315011129@cselab:~$ ./a.out
Input the number of vertices: 4
Input the adjacency matrix for the graph (use INT_MAX for infinity):
0 0 1 0
0 0 1 1
1 1 0 1
1 1 1 0
Input the source vertex: 3
Vertex  Distance from Source
0       1
1       1
2       1
3       0
aiml2315011129@cselab:~$
```

# Ex 15: Sorting Techniques

QUICK SORT:

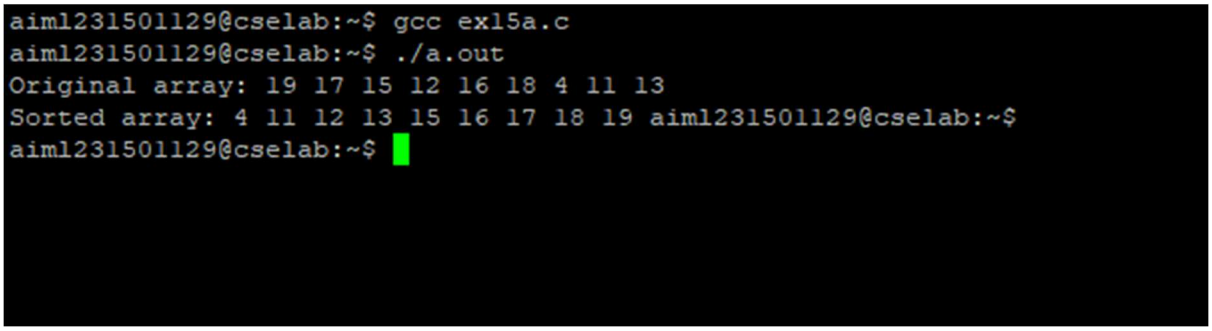
PROGRAM:

```
#include <stdio.h>

void swap(int* a, int* b)

{

	int temp = *a;

	*a = *b;

	*b = temp;

}

int partition(int arr[], int low, int high)

{

	int pivot = arr[low];

	int i = low;

	int j = high;


	while (i < j) {

		while (arr[i] <= pivot && i <= high - 1) {

			i++;

		}

		while (arr[j] > pivot && j >= low + 1) {

			j--;

		}

		if (i < j) {

			swap(&arr[i], &arr[j]);

		}
```

```
        }
        swap(&arr[low], &arr[j]);
        return j;
}
void quickSort(int arr[], int low, int high)
{
        if (low < high) {
                int partitionIndex = partition(arr, low, high);
                quickSort(arr, low, partitionIndex - 1);
                quickSort(arr, partitionIndex + 1, high);
        }
}
int main()
{
        int arr[] = { 19, 17, 15, 12, 16, 18, 4, 11, 13 };
        int n = sizeof(arr) / sizeof(arr[0]);
        printf("Original array: ");
        for (int i = 0; i < n; i++) {
                printf("%d ", arr[i]);
        }
        quickSort(arr, 0, n - 1);
        printf("\nSorted array: ");
        for (int i = 0; i < n; i++) {
                printf("%d ", arr[i]);
        }

        return 0;
```

}

OUTPUT:




MERGE SORT:

PROGRAM:

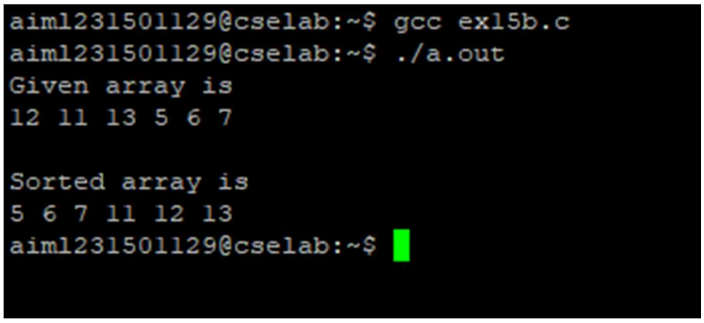
```
#include <stdio.h>

#include <stdlib.h>

void merge(int arr[], int l, int m, int r)

{

        int i, j, k;

        int n1 = m - l + 1;

        int n2 = r - m;

        int L[n1], R[n2];

        for (i = 0; i < n1; i++)

                L[i] = arr[l + i];

        for (j = 0; j < n2; j++)

                R[j] = arr[m + 1 + j];

        i = 0;


        j = 0;
```

```
	k = l;
	while (i < n1 && j < n2) {
		if (L[i] <= R[j]) {
			arr[k] = L[i];
			i++;
		}
		else {
			arr[k] = R[j];
			j++;
		}
		k++;
	}

	while (i < n1) {
		arr[k] = L[i];
		i++;
		k++;
	}

	while (j < n2) {
		arr[k] = R[j];
		j++;
		k++;
	}
}
void mergeSort(int arr[], int l, int r)
{
```

```
	if (l < r) {
		int m = l + (r - l) / 2;
		mergeSort(arr, l, m);
		mergeSort(arr, m + 1, r);

		merge(arr, l, m, r);
	}
}
void printArray(int A[], int size)
{
	int i;
	for (i = 0; i < size; i++)
		printf("%d ", A[i]);
	printf("\n");
}
int main()
{
	int arr[] = { 12, 11, 13, 5, 6, 7 };
	int arr_size = sizeof(arr) / sizeof(arr[0]);

	printf("Given array is \n");
	printArray(arr, arr_size);

	mergeSort(arr, 0, arr_size - 1);

	printf("\nSorted array is \n");
	printArray(arr, arr_size);
```

```
    return 0;
}
```

OUTPUT:



# Ex 16: Hashing

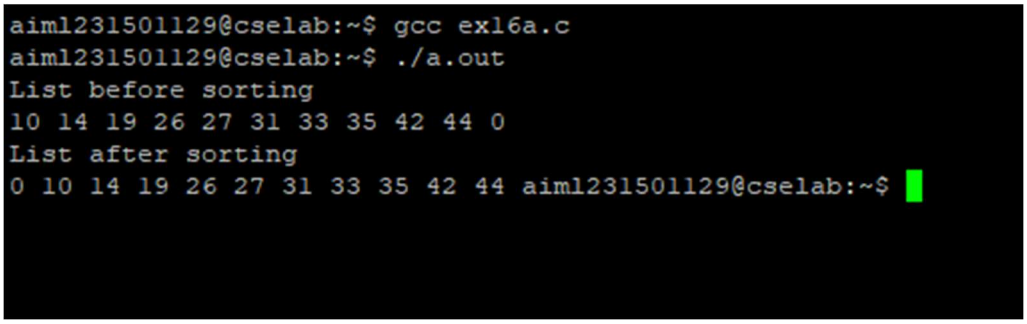
OPEN ADDRESING:

```
#include <stdio.h>


#define max 10


int a[11] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44, 0 };
int b[10];


void merging(int low, int mid, int high) {
  int l1, l2, i;


  for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++) {
    if(a[l1] <= a[l2])
      b[i] = a[l1++];
    else
      b[i] = a[l2++];
  }


  while(l1 <= mid)
    b[i++] = a[l1++];


  while(l2 <= high)
    b[i++] = a[l2++];


  for(i = low; i <= high; i++)
```

```
    a[i] = b[i];
}

void sort(int low, int high) {
  int mid;

  if(low < high) {
    mid = (low + high) / 2;
    sort(low, mid);
    sort(mid+1, high);
    merging(low, mid, high);
  } else {
    return;
  }
}

int main() {
  int i;

  printf("List before sorting\n");

  for(i = 0; i <= max; i++)
    printf("%d ", a[i]);

  sort(0, max);

  printf("\nList after sorting\n");
```

```
  for(i = 0; i <= max; i++)
    printf("%d ", a[i]);
}
```

OUTPUT:




CLOSED ADDRESING:

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node {
   int key;
   int value;
   struct Node* next;
} Node;
typedef struct HashTable {
   int size;
   Node** table;
} HashTable;
Node* createNode(int key, int value) {
   Node* newNode = (Node*)malloc(sizeof(Node));
```

```
    newNode->key = key;
    newNode->value = value;
    newNode->next = NULL;
    return newNode;
}
HashTable* createTable(int size) {
    HashTable* newTable = (HashTable*)malloc(sizeof(HashTable));
    newTable->size = size;
    newTable->table = (Node**)malloc(sizeof(Node*) * size);
    for (int i = 0; i < size; i++) {
        newTable->table[i] = NULL;
    }
    return newTable;
}
int hashFunction(int key, int size) {
    return key % size;
}
void insert(HashTable* hashTable, int key, int value) {
    int hashIndex = hashFunction(key, hashTable->size);
    Node* newNode = createNode(key, value);
    newNode->next = hashTable->table[hashIndex];
    hashTable->table[hashIndex] = newNode;
}
int search(HashTable* hashTable, int key) {
    int hashIndex = hashFunction(key, hashTable->size);
    Node* current = hashTable->table[hashIndex];
    while (current != NULL) {
```

```
        if (current->key == key) {
            return current->value;
        }
        current = current->next;
    }
    return -1;
}
void delete(HashTable* hashTable, int key) {
    int hashIndex = hashFunction(key, hashTable->size);
    Node* current = hashTable->table[hashIndex];
    Node* prev = NULL;
    while (current != NULL && current->key != key) {
        prev = current;
        current = current->next;
    }
    if (current == NULL) {
        return;
    }
    if (prev == NULL) {
        hashTable->table[hashIndex] = current->next;
    } else {
        prev->next = current->next;
    }
    free(current);
}
void freeTable(HashTable* hashTable) {
    for (int i = 0; i < hashTable->size; i++) {
```

```
        Node* current = hashTable->table[i];
        while (current != NULL) {
            Node* temp = current;
            current = current->next;
            free(temp);
        }
    }
    free(hashTable->table);
    free(hashTable);
}
int main() {
    HashTable* hashTable = createTable(10);

    insert(hashTable, 1, 10);
    insert(hashTable, 2, 20);
    insert(hashTable, 12, 30);

    printf("Value for key 1: %d\n", search(hashTable, 1));
    printf("Value for key 2: %d\n", search(hashTable, 2));
    printf("Value for key 12: %d\n", search(hashTable, 12));
    printf("Value for key 3: %d\n", search(hashTable, 3));

    delete(hashTable, 2);
    printf("Value for key 2 after deletion: %d\n", search(hashTable, 2));

    freeTable(hashTable);
    return 0;
```

}

OUTPUT:

```
aiml231501129@cselab:~$ gcc ex16b.c
aiml231501129@cselab:~$ ./a.out
Value for key 1: 10
Value for key 2: 20
Value for key 12: 30
Value for key 3: -1
Value for key 2 after deletion: -1
aiml231501129@cselab:~$
```

REHASHING:

PROGRAM:

```
#include <stdio.h>

#include <stdlib.h>


typedef struct Node {

    int key;

    int value;

    struct Node* next;

} Node;


typedef struct HashTable {

    int size;

    int count;

    Node** table;

} HashTable;


Node* createNode(int key, int value) {

    Node* newNode = (Node*)malloc(sizeof(Node));
```

```
    newNode->key = key;
    newNode->value = value;
    newNode->next = NULL;
    return newNode;
}

HashTable* createTable(int size) {
    HashTable* newTable = (HashTable*)malloc(sizeof(HashTable));
    newTable->size = size;
    newTable->count = 0;
    newTable->table = (Node**)malloc(sizeof(Node*) * size);
    for (int i = 0; i < size; i++) {
        newTable->table[i] = NULL;
    }
    return newTable;
}

int hashFunction(int key, int size) {
    return key % size;
}

void insert(HashTable* hashTable, int key, int value);

void rehash(HashTable* hashTable) {
    int oldSize = hashTable->size;
    Node** oldTable = hashTable->table;
    int newSize = oldSize * 2;
```

```
    hashTable->table = (Node**)malloc(sizeof(Node*) * newSize);
    hashTable->size = newSize;
    hashTable->count = 0;
    for (int i = 0; i < newSize; i++) {
        hashTable->table[i] = NULL;
    }

    for (int i = 0; i < oldSize; i++) {
        Node* current = oldTable[i];
        while (current != NULL) {
            insert(hashTable, current->key, current->value);
            Node* temp = current;
            current = current->next;
            free(temp);
        }
    }

    free(oldTable);
}

void insert(HashTable* hashTable, int key, int value) {
    if ((float)hashTable->count / hashTable->size >= 0.75) {
        rehash(hashTable);
    }

    int hashIndex = hashFunction(key, hashTable->size);
    Node* newNode = createNode(key, value);
```

```
    newNode->next = hashTable->table[hashIndex];
    hashTable->table[hashIndex] = newNode;
    hashTable->count++;
}

int search(HashTable* hashTable, int key) {
    int hashIndex = hashFunction(key, hashTable->size);
    Node* current = hashTable->table[hashIndex];
    while (current != NULL) {
        if (current->key == key) {
            return current->value;
        }
        current = current->next;
    }
    return -1;
}

void delete(HashTable* hashTable, int key) {
    int hashIndex = hashFunction(key, hashTable->size);
    Node* current = hashTable->table[hashIndex];
    Node* prev = NULL;
    while (current != NULL && current->key != key) {
        prev = current;
        current = current->next;
    }
    if (current == NULL) {
        return;
```

```
    }
    if (prev == NULL) {
        hashTable->table[hashIndex] = current->next;
    } else {
        prev->next = current->next;
    }
    free(current);
    hashTable->count--;
}

void freeTable(HashTable* hashTable) {
    for (int i = 0; i < hashTable->size; i++) {
        Node* current = hashTable->table[i];
        while (current != NULL) {
            Node* temp = current;
            current = current->next;
            free(temp);
        }
    }
    free(hashTable->table);
    free(hashTable);
}

int main() {
    HashTable* hashTable = createTable(5);

    insert(hashTable, 1, 10);
```

```
    insert(hashTable, 2, 20);
    insert(hashTable, 3, 30);
    insert(hashTable, 4, 40);
    insert(hashTable, 5, 50);
    insert(hashTable, 6, 60);

    printf("Value for key 1: %d\n", search(hashTable, 1));
    printf("Value for key 2: %d\n", search(hashTable, 2));
    printf("Value for key 3: %d\n", search(hashTable, 3));
    printf("Value for key 4: %d\n", search(hashTable, 4));
    printf("Value for key 5: %d\n", search(hashTable, 5));
    printf("Value for key 6: %d\n", search(hashTable, 6));

    delete(hashTable, 3);
    printf("Value for key 3 after deletion: %d\n", search(hashTable, 3));

    freeTable(hashTable);
    return 0;
}
```

OUTPUT: