1. Program to simulate the working of stack using array

```
#include<stdio.h>
#define MAX 5
int s[5],top = -1;

void display()
{
   int i;
   puts("\n---stack info--");
   if(top == -1)
   {
      puts("\n...Empty stack...");
      return;
   }
   for(i=top; i>=0; i--)
      printf("s[%d]=%d\n",i,s[i]);
}

void pop()
{
   if(top == -1)
   {
      puts("\n...stack underflow...");
      return;
   }
   printf("\npoped element s[%d]=%d",top, s[top]);
   top = top-1;
   display();
}
```

```
void push()
{
   int item;
   if(top == (MAX-1))
   {
      puts("\n...stack overflow...");
      return;
   }
   printf("\nEnter the element:");
   scanf("%d",&item);
   s[++top] = item;
   display();
}

void main()
{
   int ch;
   while(1)
   {
      printf("***Stack Operations***\n");
      printf("1.Push\n");
      printf("2.Pop\n");
      printf("3.Exit\n");
      printf("Enter the choice\t");
      scanf("%d",&ch);
      switch(ch)
      {
      case 1:
         push();
```

```
            break;
        case 2:
            pop();
            break;
        default:
            exit(0);
        }
    }
}
```

2. Program to implement Array operations

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int a[100], n;
int menu()
{
   int choice;
   puts("\n\n\\*... Array operations...*\\\n");
   puts("1. Insertion");
   puts("2. Deletion");
   puts("3. Display ");
   puts("4. Exit");
   printf("Enter your choice :\t");
   scanf("%d",&choice);
   return(choice);
}

void display();
void swap(int *x, int *y)
{
   int temp;
   temp = *x;
   *x = *y;
   *y = temp;
}

void insert()
{
```

```
    int k,i,item;
    printf("Enter the index & Element of to be inserted :\t");
    scanf("%d%d",&k,&item);
    if (k>n)
    {
        printf("Enter k between 0 to %d\t", n);
        scanf("%d", &k);
    }
    for(i=n-1; i>=k; i--)
        a[i+1] = a[i];
    a[k] = item;
    n++;
    display();
}

void deleteEle()
{
    int k,i;
    printf("Enter the index of the element :\t");
    scanf("%d",&k);
    printf("Deleted Element : %d", a[k]);
    for(i=k; i<n-1; i++)
        a[i] = a[i+1];
    n--;
    display();
}

int main()
{
    int i,ch;
```

```
    float f;

    printf("\nEnter how many elements : \t");
    scanf("%d",&n);
    puts("Enter elements of the array : ");
    for(i=0; i<n; i++)
        scanf("%d", &a[i]);
    while(1)
    {
        ch = menu();
        switch(ch)
        {
        case 1:
            insert();
            break;
        case 2:
            deleteEle();
            break;
        case 3:
            display();
            break;
        default:
            puts("\n\nNice you have performed all operations on array\n\t\tPress any key to
Quit..");
            getch();
            exit(0);
        }
    }
}

void display()
```

```
{
   int i;
   puts("elements of the array");
   for(i=0; i<n; i++)
      printf("a[%d] = %d\n", i,a[i]);
   getch();

}
```

3. Program to simulate the working of circular queue using array

```
#include<stdio.h>
#define MAX 5
int Q[5],f=-1,r=-1;
void display()
{
   int i;
   puts("\n--Queue Info--");
   if(f==-1)
   {
      puts("\n...Queue is empty...");
      return;
   }
   if(r<f)
   {
      for(i=f; i<MAX; i++)
         printf("Q[%d]=%d\n",i,Q[i]);
      for(i=f; i<=r; i++)
         printf("Q[%d]=%d\n",i,Q[i]);
   }
   else
      for(i=f; i<=r; i++)
         printf("Q[%d]=%d\n",i,Q[i]);
}

void deleteQ()
{
   if(f==-1)
   {
```

```
        puts("\n...Queue Underflow...");
        return;
    }
    printf("\nelement deleted Q[%d]=%d",f,Q[f]);
    if(f==r)
    {
        f=-1;
        r=-1;
    }
    else if(f==MAX-1)
        f=0;
    else
        f=f+1;
    display();
}

void insert()
{
    int ele;
    if((f==0 && r==MAX-1)||(f==r+1))
    {
        puts("\n...Queue Overflow...");
        return;
    }
    printf("\nEnter the Element:");
    scanf("%d",&ele);
    if(f==-1)
    {
        f=0;
        r=0;
```

```
    }
    else if(r==MAX-1)
        r=0;
    else
        r=r+1;
    Q[r]=ele;
    display();
}

int main()
{
    int ch;
    while(1)
    {
        printf("\n***Queue Operation***\n");
        printf("1.Insert\n");
        printf("2.Delete\n");
        printf("3.Exit\n");
        printf("Enter the choice\t");
        scanf("%d",&ch);
        switch(ch)
        {
        case 1:
            insert();
            break;
        case 2:
            deleteQ();
            break;
        default:
            exit(0);
```

```
        }
    }
}
```

4. Simulate the working of stack using singly linked list

```
#include<stdio.h>
#include<stdlib.h>
typedef struct stack
{
   int data;
   struct stack *next;
} st;
st *top=NULL;

void display()
{
   st *temp = top;
   if (top == NULL)
   {
      puts("\nStack Empty");
      return;
   }
   printf("\n\n***Content of the stack***\n");
   while (temp!=NULL)
   {
      printf("->%d", temp->data);
      temp = temp->next;

   }
}

void pop()
{
```

```
    st *temp;
    if (top == NULL)
    {
        puts("\nStack underflow");
        return;
    }
    printf("\nElement deleted = %d",  top->data);
    temp = top;
    top = top->next;
    free(temp);
    display();
}


void push()
{
    int item;
    st *temp= (st*)malloc(sizeof(st));
    if (temp == NULL)
    {
        printf("\n...Stack Overflow...\n");
        return;
    }
    printf("\nEnter the element\t");
    scanf("%d", &item);
    temp->data = item;
    temp->next = top;
    top = temp;
    display();
```

```
}
int main()
{
   int ch=0;
   while(1)
   {
      printf("\n\n***Stack Operations***\n");
      printf("1. Push\n");
      printf("2. Pop\n");
      printf("3. Exit\n");
      printf("Enter your choice \t");
      scanf("%d", &ch);
      switch(ch)
      {
      case 1:
         push();
         break;
      case 2:
         pop();
         break;
      default:
         exit(0);
      }
   }
}
```

5. Simulate the working queue using singly linked list

```
#include <stdio.h>
typedef struct Queue
{
   int data;
   struct Queue *next;
} Q;

Q *f=NULL, *r = NULL;

void display()
{
   Q *temp = f;
   if (f == NULL)
   {
      puts("\nQueue Empty");
      return;
   }
   printf("\n\n***Content of the Queue***\n");
   while (temp!=NULL)
   {
      printf("%d->", temp->data);
      temp = temp->next;

   }
}

void Delete()
{
```

```
    Q *temp;
    if (f==NULL)
    {
        puts("\nQueue underflow");
        return;
    }
    printf("\nElement deleted = %d",  f->data);
    temp = f;
    if (f == r)
        f = r = NULL;
    else
        f = f->next;
    display();
    free(temp);
}


void Insert()
{
    int item;
    Q *temp= (Q*)malloc(sizeof(Q));
    if (temp == NULL)
    {
        printf("\n...Queue Overflow...\n");
        return;
    }
    printf("\nEnter the element\t");
    scanf("%d", &item);
    temp->data = item;
    temp->next = NULL;
```

```
    if (f == NULL)
        f = r = temp;
    else
    {
        r->next = temp;
        r = temp;
    }
    display();

}
int main()
{
    int ch=0;
    while(1)
    {
        printf("\n\n***Queue Operations***\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Exit\n");
        printf("Enter your choice \t");
        scanf("%d", &ch);
        switch(ch)
        {
        case 1:
            Insert();
            break;
        case 2:
            Delete();
            break;
        default:
```

```
            exit(0);
        }
    }
}
```

6. Simulate the working of circular queue using single linked list

```
#include<stdio.h>
typedef struct Queue
{
   int data;
   struct Queue *next;
} Q;

Q *f = NULL, *r = NULL;

void display()
{
   Q *temp = f;
   if(f == NULL)
   {
      puts("\nQueue Empty");
      return;
   }
   printf("\n\n***Contents of the Queue***\n");
   do
   {
      printf("%d->", temp->data);
      temp = temp->next;
   }
   while(temp!=f);

}

void Delete()
```

```
{
    Q *temp;
    if(f == NULL)
    {
        puts("\nQueue underflow");
        return;
    }

    printf("\nElement deleted = %d", f->data);
    temp = f;
    if(f == r)
        f = r = NULL;
    else
        f = f->next;
    r->next = f;
    display();
    free(temp);
}

void Insert()
{
    int item;
    Q *temp = (Q*)malloc(sizeof(Q));
    printf("\nEnter the element:\t");
    scanf("%d", &item);
    temp->data = item;
    temp->next = NULL;
    if(f == NULL)
        f = r = temp;
    else
```

```
    {
        r->next = temp;
        r = temp;
    }
    r->next = f;
    display();
}

int main()
{
    int ch = 0;
    while(1)
    {
        printf("\n\n***Queue Operations***\n");
        printf("1.Insert\n");
        printf("2.Delete\n");
        printf("3.Exit\n");
        printf("Enter your choice:\t");
        scanf("%d", &ch);
        switch(ch)
        {
        case 1:
            Insert();
            break;
        case 2:
            Delete();
            break;
        case 3:
            exit(0);
        default:
```

```
            puts("Invalid choice");
        }
    }
}
```

7. Simulate the working of stack using double linked list

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

typedef struct stack
{
   int data;
   struct stack *prev;
   struct stack *next;
} st;

st *top = NULL;

void display()
{
   st *temp = top;
   if (top == NULL)
   {
      printf("\nEmpty stack\n");
      return;
   }
   printf("\nElements of the stack:\n");
   while (temp != NULL)
   {
      printf("%d", temp->data);
      if (temp->next != NULL)
      {
         printf("->");  // Print arrow only if there is a next element
      }
```

```
        temp = temp->next;
    }
    printf("\n");  // End the display with a newline
}

void push()
{
    st *temp = (st*) malloc(sizeof(st));
    if (temp == NULL)
    {
        printf("\nOverflow\n");
        return;
    }
    printf("\nEnter the element value: ");
    scanf("%d", &temp->data);
    temp->prev = NULL;
    temp->next = top;
    if (top != NULL)
    {
        top->prev = temp;
    }
    top = temp;
    display();
}

void pop()
{
    st *temp = top;
    if (top == NULL)
    {
```

```
        puts("Underflow");
            return;
    }
    printf("Data popped out from stack = %d\n", top->data);
    top = top->next;
    if (top != NULL)
    {
        top->prev = NULL;
    }
    free(temp);
    display();
}

int main()
{
    int ch;
    clrscr();
    while (1)
    {
        printf("\n\nStack Operations \n1.Push \n2.Pop \n3.Display \n4.Exit \nEnter your choice:
");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            push();
            break;
        case 2:
            pop();
            break;
        case 3:
```

```
            display();
            break;
        default:
            exit(0);
        }
    }
}
```

8. Simulate the working queue using double linked list

```
#include<stdio.h>
#include<stdlib.h>
typedef struct queue
{
   int data;
   struct queue *next;
   struct queue *prev;
} que;

que *front = NULL, *rear = NULL;
void display()
{
   que *temp;
   temp = front;
   if(front == NULL)
   {
      printf("\nEmpty queue \n");
      return;
   }
   printf("\nElements of the queue \n");
   while(temp != NULL)
   {
      printf("%d->",temp->data);
      temp = temp -> next;
   }
}

void insert()
```

```
{
    que *temp = (que*) malloc(sizeof(que));
    if(temp == NULL)
    {
        printf("\nOverflow");
        return;
    }
    temp -> next = NULL;
    temp -> prev = NULL;
    printf("\nEnter the element value:");
    scanf("%d",&temp -> data);
    if(front == NULL)
    {
        front = temp;
        rear = temp;
    }
    else
    {
        rear -> next = temp;
        temp -> prev = rear;
        rear = temp;
    }
    display();
}

void deleteq()
{
    que *temp;
    if(front == NULL)
    {
```

```
        puts("Underflow");
        return;
    }
    temp = front;
    printf("data deleted from queue=%d",front -> data);
    if(front == rear)
    {
        front = NULL;
        rear = NULL;
    }
    else
    {
        front = front -> next;
        front -> prev = NULL;
    }
    free(temp);
    display();
}

int main()
{
    int ch;
    while(1)
    {
        printf("\n\nQueue Operations \n 1.Insert \n 2.Delete \n 3.Display \n 4.Exit \n Enter your
choice:");
        scanf("%d",&ch);
        switch(ch)
        {
        case 1:
            insert();
```

```
            break;
        case 2:
            deleteq();
            break;
        case 3:
            display();
            break;
        default :
            exit(0);
        }
    }
}
```

9. Simulate the working of circular queue using double linked list

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

typedef struct queue
{
   int data;
   struct queue *next;
   struct queue *prev;
} que;

que *front = NULL, *rear = NULL;

void display()
{
   que *temp = front;
   if (front == NULL)
   {
      printf("\nEmpty queue\n");
      return;
   }
   printf("\nElements of the queue:\n");
   do
   {
      printf("%d", temp->data);
      temp = temp->next;
      if (temp != front)
      {
         printf("->");
```

```
        }
    }
    while (temp != front);
    printf("\n");
}

void insert()
{
    que *temp = (que*) malloc(sizeof(que));
    if (temp == NULL)
    {
        printf("\nOverflow\n");
        return;
    }
    temp->next = NULL;
    temp->prev = NULL;
    printf("\nEnter the element value: ");
    scanf("%d", &temp->data);
    if (front == NULL)
    {
        front = temp;
        rear = temp;
        temp->next = front;
        temp->prev = rear;
    }
    else
    {
        rear->next = temp;
        temp->prev = rear;
        temp->next = front;
```

```
        rear = temp;
        front->prev = rear;
    }
    display();
}

void deleteq()
{
    que *temp = front;
    if (front == NULL)
    {
            puts("Underflow");
            return;
    }
    printf("Data deleted from queue = %d\n", front->data);
    if (front == rear)    // Only one element in the queue
    {
        front = NULL;
        rear = NULL;
    }
    else
    {
        front = front->next;
        front->prev = rear;
        rear->next = front;
    }
    free(temp);
    display();
}
```

```
int main()
{
   int ch;
   clrscr();
   while (1)
   {
      printf("\n\nQueue Operation \n1.Insert \n2.Delete \n3.Display \n4.Exit \n Enter your
choice: ");
      scanf("%d", &ch);
      switch (ch)
      {
      case 1:
         insert();
         break;
      case 2:
         deleteq();
         break;
      case 3:
         display();
         break;
      default:
         exit(0);
      }
   }
}
```

10. Implement Quick Sort

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

void quicksort(int a[], int low, int high);

int main()
{
   int a[50];
   int n, i;

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

   printf("Enter the elements to be sorted:\n");
   for (i = 0; i < n; i++)
   {
      scanf("%d", &a[i]);
   }

   quicksort(a, 0, n - 1);

   printf("After applying quick sort\n");
   for (i = 0; i < n; i++)
   {
      printf("a[%d] = %d\n", i, a[i]);
   }

   return 0;
```

```
}

void quicksort(int a[], int low, int high)
{
   int p, i, j, temp;
   if (low < high)
   {
      i = low;
      j = high;
      p = a[low];
      while (i < j)
      {
         while ((a[i] <= p) && (i <= high))
            i++;
         while ((a[j] > p) && (j >= low))
            j--;
         if (i < j)
         {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
         }
      }
      temp = a[j];
      a[j] = a[low];
      a[low] = temp;
      quicksort(a, low, j - 1);
      quicksort(a, j + 1, high);
   }
}
```

11. Merge Sort Implementation

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

void Merge(int *a, int low, int high, int mid) {
    int i, j, k;
    int *temp = (int*)malloc((high - low + 1) * sizeof(int));
    i = low;
    k = 0;
    j = mid + 1;

    while (i <= mid && j <= high) {
        if (a[i] < a[j]) {
            temp[k] = a[i];
            k++;
            i++;
        } else {
            temp[k] = a[j];
            k++;
            j++;
        }
    }

    while (i <= mid) {
        temp[k] = a[i];
        k++;
        i++;
    }
```

```
    while (j <= high) {
        temp[k] = a[j];
        k++;
        j++;
    }

    for (i = low; i <= high; i++) {
        a[i] = temp[i - low];
    }

    free(temp);
}

void MergeSort(int *a, int low, int high) {
    int mid;
    if (low < high) {
        mid = (low + high) / 2;
        MergeSort(a, low, mid);
        MergeSort(a, mid + 1, high);
        Merge(a, low, high, mid);
    }
}

int main() {
    int n, i;

    printf("Enter the number of data elements to be sorted: ");
    scanf("%d", &n);

    int *arr = (int*)malloc(n * sizeof(int));
```

```
    for (i = 0; i < n; i++) {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &arr[i]);
    }

    MergeSort(arr, 0, n - 1);

    printf("\nSorted Data: ");
    for (i = 0; i < n; i++) {
        printf("\narr[%d] = %d", i, arr[i]);
    }

    free(arr);
    return 0;
}
```

12. Create a binary tree and implement the tree traversal techniques

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

struct node
{
   int data;
   struct node *leftChild;
   struct node *rightChild;
};

struct node *root = NULL;

void insert(int data)
{
   struct node *tempNode = (struct node *)malloc(sizeof(struct node));
   struct node *current;
   struct node *parent;

   tempNode->data = data;
   tempNode->leftChild = NULL;
   tempNode->rightChild = NULL;

   if (root == NULL)
   {
      root = tempNode;
   }
   else
   {
```

```
        current = root;
        parent = NULL;
        while (1)
        {
            parent = current;
            if (data < parent->data)
            {
                current = current->leftChild;
                if (current == NULL)
                {
                    parent->leftChild = tempNode;
                    return;
                }
            }
            else
            {
                current = current->rightChild;
                if (current == NULL)
                {
                    parent->rightChild = tempNode;
                    return;
                }
            }
        }
    }
}

void pre_order_traversal(struct node* root)
{
    if (root != NULL)
```

```
    {
        printf("%d\n ", root->data);
        pre_order_traversal(root->leftChild);
        pre_order_traversal(root->rightChild);
    }
}

void inorder_traversal(struct node* root)
{
    if (root != NULL)
    {
        inorder_traversal(root->leftChild);
        printf("%d\n ", root->data);
        inorder_traversal(root->rightChild);
    }
}

void post_order_traversal(struct node* root)
{
    if (root != NULL)
    {
        post_order_traversal(root->leftChild);
        post_order_traversal(root->rightChild);
        printf("%d\n", root->data);
    }
}

int main()
{
    int i, array[7] = { 99, 109, 79, 119, 129, 59, 69 };
```

```
    for (i = 0; i < 7; i++)
    {
        insert(array[i]);
    }

    printf("\nPreorder traversal: ");
    pre_order_traversal(root);
    printf("\nInorder traversal: ");
    inorder_traversal(root);
    printf("\nPostorder traversal: ");
    post_order_traversal(root);
    printf("\n");

    return 0;
}
```

13. Implement search using BST

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

struct node
{
   int data;
   struct node *leftChild;
   struct node *rightChild;
};

struct node *root = NULL;

void insert(int data)
{
   struct node *tempNode = (struct node *)malloc(sizeof(struct node));
   struct node *current;
   struct node *parent;

   tempNode->data = data;
   tempNode->leftChild = NULL;
   tempNode->rightChild = NULL;

   if (root == NULL)
   {
      root = tempNode;
   }
   else
   {
```

```
        current = root;
        parent = NULL;
        while (1)
        {
            parent = current;
            if (data < parent->data)
            {
                current = current->leftChild;
                if (current == NULL)
                {
                    parent->leftChild = tempNode;
                    return;
                }
            }
            else
            {
                current = current->rightChild;
                if (current == NULL)
                {
                    parent->rightChild = tempNode;
                    return;
                }
            }
        }
    }
}

struct node* search(int data)
{
    struct node *current = root;
```

```
    printf("\n\nVisiting elements: \n");
    while (current != NULL && current->data != data)
    {
        if (current->data > data)
        {
            current = current->leftChild;
        }
        else
        {
            current = current->rightChild;
        }
    }
    return current;
}

int main()
{
    int key, i;
    int array[7] = { 27, 14, 35, 10, 19, 31, 42 };

    for (i = 0; i < 7; i++)
    {
        insert(array[i]);
    }

    printf("\nEnter key to be searched: ");
    scanf("%d", &key);

    struct node *temp = search(key);
```

```
    if (temp != NULL)
    {
        printf("Element found.\n");
    }
    else
    {
        printf("%d Element not found.\n", key);
    }

    return 0;
}
```

14. Compute the transitive closure of a given directed graph using Warshall's algorithm

```
#include <stdio.h>


void warshall(int a[25][25], int n)
{
   int i, j, k;
   for (k = 1; k <= n; k++)
   {
      for (i = 1; i <= n; i++)
      {
         for (j = 1; j <= n; j++)
         {
            if (a[i][j] != 1)
            {
               a[i][j] = a[i][k] && a[k][j];
            }
         }
      }
      printf("%d. Transitive Closure:\n", k);
      for (i = 1; i <= n; i++)
      {
         for (j = 1; j <= n; j++)
         {
            printf("%d\t", a[i][j]);
         }
         printf("\n");
      }
   }
}
```

```
int main()
{
    int a[25][25], n, i, j;
    printf("\n\n\t\tWarshall's ALGORITHM TO FIND THE TRANSITIVE CLOSURE\n");
    printf("\nEnter the number of Vertices: ");
    scanf("%d", &n);
    printf("\nEnter the Adjacency Matrix:\n");
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {

            scanf("%d", &a[i][j]);
        }
    }
    printf("\nAdjacency Matrix:\n");
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    warshall(a, n);
    return 0;
}
```

15. Implement Floyd's algorithm for the All-Pairs- Shortest-Paths Algorithm.

```
#include <stdio.h>


void floyd(int a[25][25], int n)
{
   int i, j, k;
   for (k = 1; k <= n; k++)
   {
      for (i = 1; i <= n; i++)
      {
         for (j = 1; j <= n; j++)
         {
            if ((a[i][j]) < ((a[i][k]) + (a[k][j])))
            {
               a[i][j] = a[i][j];
            }
            else
            {
               a[i][j] = a[i][k] + a[k][j];
            }
         }
      }
      printf("%d. Distance Matrix:\n", k);
      for (i = 1; i <= n; i++)
      {
         for (j = 1; j <= n; j++)
         {
            printf("%d\t", a[i][j]);
         }
```

```
            printf("\n");
        }
    }
}

int main()
{
    int a[25][25], n, i, j;
    printf("\n\n\t\tFLOYDS ALGORITHM TO FIND ALL PAIRS SHORTEST PATH\n");
    printf("\nEnter the number of Vertices: ");
    scanf("%d", &n);
    printf("\nEnter the Cost Adjacency Matrix:\n");
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            scanf("%d", &a[i][j]);
        }
    }
    printf("\nCost Matrix:\n");
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    printf("\n");
    floyd(a, n);
```

```
    return 0;
}
```

16. Find the Binomial Co-efficient using Dynamic Programming.

```
#include <stdio.h>

int c[20][20], n, k;

void input()
{
   printf("\nEnter value of n: ");
   scanf("%d", &n);
   printf("\nEnter value of k: ");
   scanf("%d", &k);
}

int min(int a, int b)
{
   return (a < b ? a : b);
}

void Binomial()
{
   int i, j;
   for (i = 0; i <= n; i++)
   {
      for (j = 0; j <= min(i, k); j++)
      {
         if (j == 0 || j == i)
         {
            c[i][j] = 1;
         }
```

```
            else
            {
                c[i][j] = c[i-1][j] + c[i-1][j-1];
            }
        }
    }

    for (i = 0; i <= n; i++)
    {
        for (j = 0; j <= min(i, k); j++)
        {
            printf("%d\t", c[i][j]);
        }
        printf("\n\n");
    }

    printf("\nBinomial Coefficient:\n");
    printf("\nC[%d, %d] = %d\n", n, k, c[n][k]);
}

int main()
{
    input();
    Binomial();
    return 0;
}
```