1.Linked List.c

#include<stdlib.h>

#include <stdio.h>

void create();

void display();

struct node

{

int info;

struct node \*next;

};

struct node \*start=NULL;

int main()

{

int choice;

while(1){

printf("\n 1.Create \n");

printf("\n 2.Display \n");

printf("\n 3.Exit \n");

printf("Enter your choice:\t");

scanf("%d",&choice);

switch(choice)

{

case 1:

create();

break;

case 2:

display();

break;

case 3:

exit(0);

break;

default:

printf("\n Wrong Choice:n");

break;

}

}

return 0;

}

void create()

{

struct node \*temp,\*ptr;

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

/\* if(temp==NULL)

{

printf("\n Out of Memory Space:\n");

exit(0);

}\*/

printf("\nEnter the data value for the node:\t");

scanf("%d",&temp->info);

temp->next=NULL;

if(start==NULL)

{

start=temp;

}

else

{

ptr=start;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=temp;

}

}

void display()

{

struct node \*ptr;

/\* if(start==NULL)

{

printf("\nList is empty:\n");

return;

}

else

{

ptr=start;

printf("\n The List elements are:\t");

while(ptr!=NULL)

{

printf("%d ",ptr->info );

ptr=ptr->next ;

}

}\*/

printf("\n The List elements are:\t");

for(ptr=start;ptr!=NULL;ptr=ptr->next)

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

}

2. .Linked List op.c

#include<stdlib.h>

#include <stdio.h>

void create();

void display();

void insert\_begin();

void insert\_end();

void insert\_pos();

void delete\_begin();

void delete\_end();

void delete\_pos();

struct node

{

int info;

struct node \*next;

};

struct node \*start=NULL;

int main()

{

int choice;

while(1){

printf("\n MENU n");

printf("\n 1.Create \n");

printf("\n 2.Display \n");

printf("\n 3.Insert at the beginning \n");

printf("\n 4.Insert at the end \n");

printf("\n 5.Insert at specified position \n");

printf("\n 6.Delete from beginning \n");

printf("\n 7.Delete from the end \n");

printf("\n 8.Delete from specified position \n");

printf("\n 9.Exit \n");

printf("n--------------------------------------n");

printf("Enter your choice: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

create();

break;

case 2:

display();

break;

case 3:

insert\_begin();

break;

case 4:

insert\_end();

break;

case 5:

insert\_pos();

break;

case 6:

delete\_begin();

break;

case 7:

delete\_end();

break;

case 8:

delete\_pos();

break;

case 9:

exit(0);

break;

default:

printf("n Wrong Choice:n");

break;

}

}

return 0;

}

void create()

{

struct node \*temp,\*ptr;

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

if(temp==NULL)

{

printf("\n Out of Memory Space:\n");

exit(0);

}

printf("\n Enter the data value for the node:\t");

scanf("%d",&temp->info);

temp->next=NULL;

if(start==NULL)

{

start=temp;

}

else

{

ptr=start;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=temp;

}

}

void display()

{

struct node \*ptr;

if(start==NULL)

{

printf("\n List is empty:\n");

return;

}

else

{

ptr=start;

printf("\n The List elements are:\n");

while(ptr!=NULL)

{

printf("%d ",ptr->info );

ptr=ptr->next ;

}

}

}

void insert\_begin()

{

struct node \*temp;

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

if(temp==NULL)

{

printf("\n Out of Memory Space:\n ");

return;

}

printf("\n Enter the data value for the node: " );

scanf("%d",&temp->info);

temp->next =NULL;

if(start==NULL)

{

start=temp;

}

else

{

temp->next=start;

start=temp;

}

}

void insert\_end()

{

struct node \*temp,\*ptr;

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

if(temp==NULL)

{

printf("\n Out of Memory Space:\n");

return;

}

printf("\n Enter the data value for the node: " );

scanf("%d",&temp->info );

temp->next =NULL;

if(start==NULL)

{

start=temp;

}

else

{

ptr=start;

while(ptr->next !=NULL)

{

ptr=ptr->next ;

}

ptr->next =temp;

}

}

void insert\_pos()

{

struct node \*ptr,\*temp;

int i,pos;

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

if(temp==NULL)

{

printf("\n Out of Memory Space:\n");

return;

}

printf("\n Enter the position for the new node to be inserted: ");

scanf("%d",&pos);

printf("\n Enter the data value of the node: ");

scanf("%d",&temp->info) ;

temp->next=NULL;

if(pos==0)

{

temp->next=start;

start=temp;

}

else

{

for(i=0,ptr=start;i<pos-1;i++) { ptr=ptr->next;

if(ptr==NULL)

{

printf("\n Position not found:[Handle with care]\n");

return;

}

}

temp->next =ptr->next ;

ptr->next=temp;

}

}

void delete\_begin()

{

struct node \*ptr;

if(ptr==NULL)

{

printf("\n List is Empty:\n");

return;

}

else

{

ptr=start;

start=start->next ;

printf("\n The deleted element is :%d ",ptr->info);

free(ptr);

}

}

void delete\_end()

{

struct node \*temp,\*ptr;

if(start==NULL)

{

printf("\n List is Empty:");

exit(0);

}

else if(start->next ==NULL)

{

ptr=start;

start=NULL;

printf("\n The deleted element is:%d ",ptr->info);

free(ptr);

}

else

{

ptr=start;

while(ptr->next!=NULL)

{

temp=ptr;

ptr=ptr->next;

}

temp->next=NULL;

printf("\n The deleted element is:%d ",ptr->info);

free(ptr);

}

}

void delete\_pos()

{

int i,pos;

struct node \*temp,\*ptr;

if(start==NULL)

{

printf("\n The List is Empty:\n");

exit(0);

}

else

{

printf("\n Enter the position of the node to be deleted: ");

scanf("%d",&pos);

if(pos==0)

{

ptr=start;

start=start->next ;

printf("\n The deleted element is:%d ",ptr->info );

free(ptr);

}

else

{

ptr=start;

for(i=0;i<pos;i++) { temp=ptr; ptr=ptr->next ;

if(ptr==NULL)

{

printf("\n Position not Found:\n");

return;

}

}

temp->next =ptr->next ;

printf("\n The deleted element is:%d ",ptr->info );

free(ptr);

}

}

}

3. **CircularlinkedCircular List.c**

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

struct node \*head;

void beginsert ();

void lastinsert ();

void randominsert();

void begin\_delete();

void last\_delete();

void random\_delete();

void display();

void search();

void main ()

{

int choice =0;

while(choice != 7)

{

printf("\n\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\n");

printf("\nChoose one option from the following list ...\n");

printf("\n===============================================\n");

printf("\n1.Insert in begining\n2.Insert at last\n3.Delete from Beginning\n4.Delete from last\n5.Search for an element\n6.Show\n7.Exit\n");

printf("\nEnter your choice?\n");

scanf("\n%d",&choice);

switch(choice)

{

case 1:

beginsert();

break;

case 2:

lastinsert();

break;

case 3:

begin\_delete();

break;

case 4:

last\_delete();

break;

case 5:

search();

break;

case 6:

display();

break;

case 7:

exit(0);

break;

default:

printf("Please enter valid choice..");

}

}

}

void beginsert()

{

struct node \*ptr,\*temp;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW");

}

else

{

printf("\nEnter the node data?");

scanf("%d",&item);

ptr -> data = item;

if(head == NULL)

{

head = ptr;

ptr -> next = head;

}

else

{

temp = head;

while(temp->next != head)

temp = temp->next;

ptr->next = head;

temp -> next = ptr;

head = ptr;

}

printf("\nnode inserted\n");

}

}

void lastinsert()

{

struct node \*ptr,\*temp;

int item;

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

if(ptr == NULL)

{

printf("\nOVERFLOW\n");

}

else

{

printf("\nEnter Data?");

scanf("%d",&item);

ptr->data = item;

if(head == NULL)

{

head = ptr;

ptr -> next = head;

}

else

{

temp = head;

while(temp -> next != head)

{

temp = temp -> next;

}

temp -> next = ptr;

ptr -> next = head;

}

printf("\nnode inserted\n");

}

}

void begin\_delete()

{

struct node \*ptr;

if(head == NULL)

{

printf("\nUNDERFLOW");

}

else if(head->next == head)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{ ptr = head;

while(ptr -> next != head)

ptr = ptr -> next;

ptr->next = head->next;

free(head);

head = ptr->next;

printf("\nnode deleted\n");

}

}

void last\_delete()

{

struct node \*ptr, \*preptr;

if(head==NULL)

{

printf("\nUNDERFLOW");

}

else if (head ->next == head)

{

head = NULL;

free(head);

printf("\nnode deleted\n");

}

else

{

ptr = head;

while(ptr ->next != head)

{

preptr=ptr;

ptr = ptr->next;

}

preptr->next = ptr -> next;

free(ptr);

printf("\nnode deleted\n");

}

}

void search()

{

struct node \*ptr;

int item,i=0,flag=1;

ptr = head;

if(ptr == NULL)

{

printf("\nEmpty List\n");

}

else

{

printf("\nEnter item which you want to search?\n");

scanf("%d",&item);

if(head ->data == item)

{

printf("item found at location %d",i+1);

flag=0;

}

else

{

while (ptr->next != head)

{

if(ptr->data == item)

{

printf("item found at location %d ",i+1);

flag=0;

break;

}

else

{

flag=1;

}

i++;

ptr = ptr -> next;

}

}

if(flag != 0)

{

printf("Item not found\n");

}

}

}

void display()

{

struct node \*ptr;

ptr=head;

if(head == NULL)

{

printf("\nnothing to print");

}

else

{

printf("\n printing values ... \n");

while(ptr -> next != head)

{

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

ptr = ptr -> next;

}

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

}

}

4.Stack List.c

#include <stdio.h>

#include <stdlib.h>

void push();

void pop();

void display();

struct node

{

int val;

struct node \*next;

};

struct node \*head;

void main ()

{

int choice=0;

printf("\n\*\*\*\*\*\*\*\*\*Stack operations using linked list\*\*\*\*\*\*\*\*\*\n");

printf("\n----------------------------------------------\n");

while(choice != 4)

{

printf("\n\nChose one from the below options...\n");

printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");

printf("\n Enter your choice \n");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

display();

break;

}

case 4:

{

printf("Exiting....");

break;

}

default:

{

printf("Please Enter valid choice ");

}

};

}

}

void push ()

{

int val;

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

if(ptr == NULL)

{

printf("not able to push the element");

}

else

{

printf("Enter the value");

scanf("%d",&val);

if(head==NULL)

{

ptr->val = val;

ptr -> next = NULL;

head=ptr;

}

else

{

ptr->val = val;

ptr->next = head;

head=ptr;

}

printf("Item pushed");

}

}

void pop()

{

int item;

struct node \*ptr;

if (head == NULL)

{

printf("Underflow");

}

else

{

item = head->val;

ptr = head;

head = head->next;

free(ptr);

printf("Item popped");

}

}

void display()

{

int i;

struct node \*ptr;

ptr=head;

if(ptr == NULL)

{

printf("Stack is empty\n");

}

else

{

printf("Printing Stack elements \n");

while(ptr!=NULL)

{

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

ptr = ptr->next;

}

}

}

**5.Double Linked.c**

#include<stdio.h>

#include<stdlib.h>

struct node

{

struct node \*prev;

struct node \*next;

int data;

};

struct node \*head;

void insertion\_beginning();

void insertion\_last();

void insertion\_specified();

void deletion\_beginning();

void deletion\_last();

void deletion\_specified();

void display();

void search();

void main ()

{

int choice =0;

while(choice != 9)

{

printf("\n\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\n");

printf("\nChoose one option from the following list ...\n");

printf("\n===============================================\n");

printf("\n1.Insert in begining\n2.Insert at last\n3.Insert at any random location\n4.Delete from Beginning\n 5.Delete from last\n6.Delete the node after the given data\n7.Search\n8.Show\n9.Exit\n");

printf("\nEnter your choice?\n");

scanf("\n%d",&choice);

switch(choice)

{

case 1:

insertion\_beginning();

break;

case 2:

insertion\_last();

break;

case 3:

insertion\_specified();

break;

case 4:

deletion\_beginning();

break;

case 5:

deletion\_last();

break;

case 6:

deletion\_specified();

break;

case 7:

search();

break;

case 8:

display();

break;

case 9:

exit(0);

break;

default:

printf("Please enter valid choice..");

}

}

}

void insertion\_beginning()

{

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 insertion\_last()

{

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 insertion\_specified()

{

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=2;i<loc;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 deletion\_beginning()

{

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 deletion\_last()

{

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 deletion\_specified()

{

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;

printf("\n printing values...\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");

}

}

}

**6.Queue Linked.c**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct node**

**{**

**int data;**

**struct node \*next;**

**};**

**struct node \*front;**

**struct node \*rear;**

**void insert();**

**void delete();**

**void display();**

**void main ()**

**{**

**int choice;**

**while(choice != 4)**

**{**

**printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");**

**printf("\n=================================================================\n");**

**printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");**

**printf("\nEnter your choice ?");**

**scanf("%d",& choice);**

**switch(choice)**

**{**

**case 1:**

**insert();**

**break;**

**case 2:**

**delete();**

**break;**

**case 3:**

**display();**

**break;**

**case 4:**

**exit(0);**

**break;**

**default:**

**printf("\nEnter valid choice??\n");**

**}**

**}**

**}**

**void insert()**

**{**

**struct node \*ptr;**

**int item;**

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

**if(ptr == NULL)**

**{**

**printf("\nOVERFLOW\n");**

**return;**

**}**

**else**

**{**

**printf("\nEnter value?\n");**

**scanf("%d",&item);**

**ptr -> data = item;**

**if(front == NULL)**

**{**

**front = ptr;**

**rear = ptr;**

**front -> next = NULL;**

**rear -> next = NULL;**

**}**

**else**

**{**

**rear -> next = ptr;**

**rear = ptr;**

**rear->next = NULL;**

**}**

**}**

**}**

**void delete ()**

**{**

**struct node \*ptr;**

**if(front == NULL)**

**{**

**printf("\nUNDERFLOW\n");**

**return;**

**}**

**else**

**{**

**ptr = front;**

**front = front -> next;**

**free(ptr);**

**}**

**}**

**void display()**

**{**

**struct node \*ptr;**

**ptr = front;**

**if(front == NULL)**

**{**

**printf("\nEmpty queue\n");**

**}**

**else**

**{ printf("\nprinting values .....\n");**

**while(ptr != NULL)**

**{**

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

**ptr = ptr -> next;**

**}**

**}**

**}**

**7.Queue.c**

#include<stdio.h>

#include<stdlib.h>

#define maxsize 5

void insert();

void delete();

void display();

int front = -1, rear = -1;

int queue[maxsize];

void main ()

{

int choice;

while(choice != 4)

{

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n=================================================================\n");

printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");

printf("\nEnter your choice ?");

scanf("%d",&choice);

switch(choice)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

exit(0);

break;

default:

printf("\nEnter valid choice??\n");

}

}

}

void insert()

{

int item;

printf("\nEnter the element\n");

scanf("\n%d",&item);

if(rear == maxsize-1)

{

printf("\nOVERFLOW\n");

return;

}

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

{

front = 0;

rear = 0;

}

else

{

rear = rear+1;

}

queue[rear] = item;

printf("\nValue inserted ");

}

void delete()

{

int item;

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

{

printf("\nUNDERFLOW\n");

return;

}

else

{

item = queue[front];

if(front == rear)

{

front = -1;

rear = -1 ;

}

else

{

front = front + 1;

}

printf("\nvalue deleted ");

}

}

void display()

{

int i;

if(rear == -1)

{

printf("\nEmpty queue\n");

}

else

{ printf("\nprinting values .....\n");

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

{

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

}

}

}

**8.Circular Queue.c**

#include<stdio.h>

#include<stdlib.h>

#define maxsize 5

void insert();

void delete();

void display();

int front = -1, rear = -1;

int queue[maxsize];

void main ()

{

int choice;

while(choice != 4)

{

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\n=================================================================\n");

printf("\n1.insert an element\n2.Delete an element\n3.Display the queue\n4.Exit\n");

printf("\nEnter your choice ?");

scanf("%d",&choice);

switch(choice)

{

case 1:

insert();

break;

case 2:

delete();

break;

case 3:

display();

break;

case 4:

exit(0);

break;

default:

printf("\nEnter valid choice??\n");

}

}

}

void insert()

{

int item;

printf("\nEnter the element\n");

scanf("%d",&item);

if((rear+1)%maxsize == front)

{

printf("\nOVERFLOW");

return;

}

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

{

front = 0;

rear = 0;

}

else if(rear == maxsize -1 && front != 0)

{

rear = 0;

}

else

{

rear = (rear+1)%maxsize;

}

queue[rear] = item;

printf("\nValue inserted ");

}

void delete()

{

int item;

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

{

printf("\nUNDERFLOW\n");

return;

}

else if(front == rear)

{

front = -1;

rear = -1;

}

else if(front == maxsize -1)

{

front = 0;

}

else

front = front + 1;

}

void display()

{

int i;

if(front == -1)

printf("\nCircular Queue is Empty!!!\n");

else

{

i = front;

printf("\nCircular Queue Elements are : \n");

if(front <= rear){

while(i <= rear)

printf("%d %d %d\n",queue[i++],front,rear);

}

else{

while(i <= maxsize - 1)

printf("%d %d %d\n", queue[i++],front,rear);

i = 0;

while(i <= rear)

printf("%d %d %d\n",queue[i++],front,rear);

}

}

}

**9.Poly.c**

#include<stdio.h>

#include<malloc.h>

#include<conio.h>

struct link{

int coeff;

int pow;

struct link \*next;

};

struct link \*poly1=NULL,\*poly2=NULL,\*poly=NULL;

void create(struct link \*node)

{

char ch;

do

{

printf("\n enter coeff:");

scanf("%d",&node->coeff);

printf("\n enter power:");

scanf("%d",&node->pow);

node->next=(struct link\*)malloc(sizeof(struct link));

node=node->next;

node->next=NULL;

printf("\n continue(y/n):");

ch=getch();

}

while(ch=='y' || ch=='Y');

}

void show(struct link \*node)

{

while(node->next!=NULL)

{

printf("%dx^%d",node->coeff,node->pow);

node=node->next;

if(node->next!=NULL)

printf("+");

}

}

void polyadd(struct link \*poly1,struct link \*poly2,struct link \*poly)

{

while(poly1->next && poly2->next)

{

if(poly1->pow>poly2->pow)

{

poly->pow=poly1->pow;

poly->coeff=poly1->coeff;

poly1=poly1->next;

}

else if(poly1->pow<poly2->pow)

{

poly->pow=poly2->pow;

poly->coeff=poly2->coeff;

poly2=poly2->next;

}

else

{

poly->pow=poly1->pow;

poly->coeff=poly1->coeff+poly2->coeff;

poly1=poly1->next;

poly2=poly2->next;

}

poly->next=(struct link \*)malloc(sizeof(struct link));

poly=poly->next;

poly->next=NULL;

}

while(poly1->next || poly2->next)

{

if(poly1->next)

{

poly->pow=poly1->pow;

poly->coeff=poly1->coeff;

poly1=poly1->next;

}

if(poly2->next)

{

poly->pow=poly2->pow;

poly->coeff=poly2->coeff;

poly2=poly2->next;

}

poly->next=(struct link \*)malloc(sizeof(struct link));

poly=poly->next;

poly->next=NULL;

}

}

main()

{

char ch;

do{

poly1=(struct link \*)malloc(sizeof(struct link));

poly2=(struct link \*)malloc(sizeof(struct link));

poly=(struct link \*)malloc(sizeof(struct link));

printf("\nenter 1st number:");

create(poly1);

printf("\nenter 2nd number:");

create(poly2);

printf("\n1st Number:");

show(poly1);

printf("\n2nd Number:");

show(poly2);

polyadd(poly1,poly2,poly);

printf("\nAdded polynomial:");

show(poly);

printf("\n add two more numbers:");

ch=getch();

}

while(ch=='y' || ch=='Y');

}

**10.Stack Array.c**

#include <stdio.h>

int stack[100],i,j,choice=0,n,top=-1;

void push();

void pop();

void show();

void main ()

{

printf("Enter the number of elements in the stack ");

scanf("%d",&n);

printf("\*\*\*\*\*\*\*\*\*Stack operations using array\*\*\*\*\*\*\*\*\*");

printf("\n----------------------------------------------\n");

while(choice != 4)

{

printf("Chose one from the below options...\n");

printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");

printf("\n Enter your choice \n");

scanf("%d",&choice);

switch(choice)

{

case 1:

{

push();

break;

}

case 2:

{

pop();

break;

}

case 3:

{

show();

break;

}

case 4:

{

printf("Exiting....");

break;

}

default:

{

printf("Please Enter valid choice ");

}

};

}

}

void push ()

{

int val;

if (top == n )

printf("\n Overflow");

else

{

printf("Enter the value?");

scanf("%d",&val);

top = top +1;

stack[top] = val;

}

}

void pop ()

{

if(top == -1)

printf("Underflow");

else

top = top -1;

}

void show()

{

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

{

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

}

if(top == -1)

{

printf("Stack is empty");

}

}