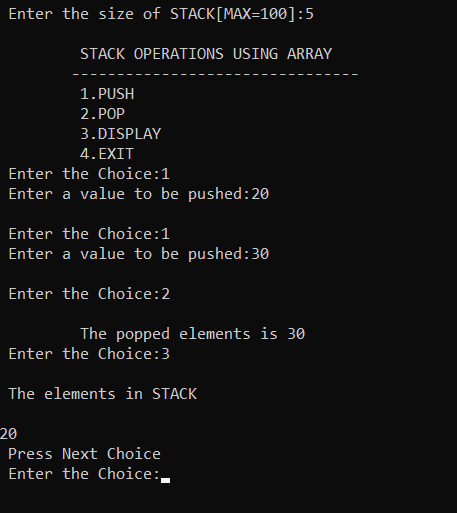
**Name- Anvitha Gowda K**

**USN- 1BM18CS018**

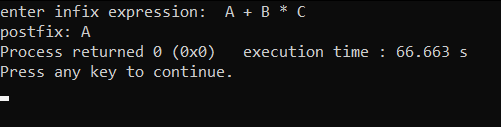
Program 1- Stack Implementation

|  |
| --- |
| #include<stdio.h> |
|  | int stack[100],choice,n,top,x,i; |
|  | void push(void); |
|  | void pop(void); |
|  | void display(void); |
|  | int main() |
|  | { |
|  | //clrscr(); |
|  | top=-1; |
|  | printf("\n Enter the size of STACK[MAX=100]:"); |
|  | scanf("%d",&n); |
|  | printf("\n\t STACK OPERATIONS USING ARRAY"); |
|  | printf("\n\t--------------------------------"); |
|  | printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT"); |
|  | do |
|  | { |
|  | printf("\n Enter the Choice:"); |
|  | scanf("%d",&choice); |
|  | switch(choice) |
|  | { |
|  | case 1: |
|  | { |
|  | push(); |
|  | break; |
|  | } |
|  | case 2: |
|  | { |
|  | pop(); |
|  | break; |
|  | } |
|  | case 3: |
|  | { |
|  | display(); |
|  | break; |
|  | } |
|  | case 4: |
|  | { |
|  | printf("\n\t EXIT POINT "); |
|  | break; |
|  | } |
|  | default: |
|  | { |
|  | printf ("\n\t Please Enter a Valid Choice(1/2/3/4)"); |
|  | } |
|  |  |
|  | } |
|  | } |
|  | while(choice!=4); |
|  | return 0; |
|  | } |
|  | void push() |
|  | { |
|  | if(top>=n-1) |
|  | { |
|  | printf("\n\tSTACK is over flow"); |
|  |  |
|  | } |
|  | else |
|  | { |
|  | printf(" Enter a value to be pushed:"); |
|  | scanf("%d",&x); |
|  | top++; |
|  | stack[top]=x; |
|  | } |
|  | } |
|  | void pop() |
|  | { |
|  | if(top<=-1) |
|  | { |
|  | printf("\n\t Stack is under flow"); |
|  | } |
|  | else |
|  | { |
|  | printf("\n\t The popped elements is %d",stack[top]); |
|  | top--; |
|  | } |
|  | } |
|  | void display() |
|  | { |
|  | if(top>=0) |
|  | { |
|  | printf("\n The elements in STACK \n"); |
|  | for(i=top; i>=0; i--) |
|  | printf("\n%d",stack[i]); |
|  | printf("\n Press Next Choice"); |
|  | } |
|  | else |
|  | { |
|  | printf("\n The STACK is empty"); |
|  | } |
|  |  |
|  | } |



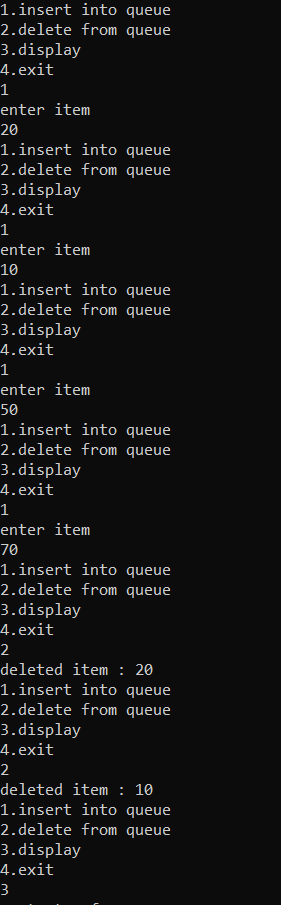
Program 2- Infix to Postfix

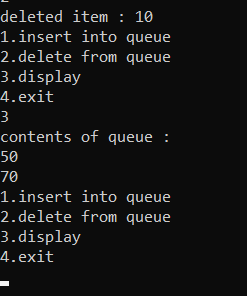
|  |
| --- |
| #include <stdio.h> |
|  | #include <string.h> |
|  | #include<ctype.h> |
|  | #define SIZE 100 |
|  | int top=-1; |
|  | char s[SIZE]; |
|  |  |
|  | void push(char val) |
|  | { |
|  | if(top==SIZE-1) |
|  | printf("stack full"); |
|  | else{ |
|  | top=top+1; |
|  | s[top]=val; |
|  | } |
|  | } |
|  |  |
|  | char pop() |
|  | { |
|  | char val=' '; |
|  | if(top==-1) |
|  | printf("stack empty"); |
|  | else{ |
|  | val=s[top]; |
|  | top=top-1; |
|  | } |
|  | return val; |
|  | } |
|  |  |
|  | int priority(char op) |
|  | { |
|  | if(op=='%' || op=='/' || op=='\*' ) |
|  | return 1; |
|  | else |
|  | return 0; |
|  | } |
|  |  |
|  | void infixToPostfix(char in[],char post[]) |
|  | { |
|  | int i=0,j=0; |
|  | while(in[i]!='\0') |
|  | { |
|  | if(in[i]=='(') |
|  | { |
|  | push(in[i]); |
|  | i++; |
|  | } |
|  | else if(in[i]==')') |
|  | { |
|  | while(top!=-1 && s[top]!='(') |
|  | { |
|  | post[j]=pop(); |
|  | j++; |
|  | } |
|  | pop(); |
|  | i++; |
|  | } |
|  | else if(isdigit(in[i]) || isalpha(in[i])) |
|  | { |
|  | post[j]=in[i]; |
|  | i++; |
|  | j++; |
|  | } |
|  | else if(in[i]=='%' || in[i]=='/' || in[i]=='-' || in[i]=='\*' || in[i]=='+' ) |
|  | { |
|  | while(top!=-1 && s[top]!='(' && (priority(s[top])>=priority(in[i]))) |
|  | { |
|  | post[j]=pop(); |
|  | j++; |
|  | } |
|  | push(in[i]); |
|  | i++; |
|  | } |
|  | else |
|  | printf("invalid expression\n"); |
|  | } |
|  | while(top!=-1 && s[top]!='(') |
|  | { |
|  | post[j]=pop(); |
|  | j++; |
|  | } |
|  | post[j]='\0'; |
|  | } |
|  |  |
|  | int main() |
|  | { |
|  | char infix[100],postfix[100]; |
|  | printf("enter infix expression: "); |
|  | scanf("%s",infix); |
|  | strcpy(postfix,""); |
|  | infixToPostfix(infix,postfix); |
|  | printf("postfix: "); |
|  | printf("%s",postfix); |
|  | return 0; |
|  | } |



Program 3-Queue Implementation

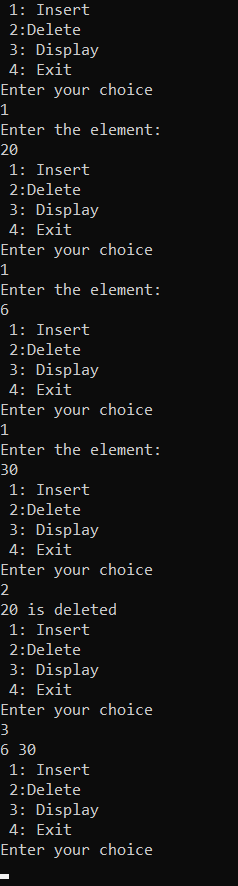
|  |
| --- |
|  |
| #include<stdio.h> |
|  | #include<stdlib.h> |
|  | #define size 5 |
|  | void enqueue(int q[], int\*r) |
|  | { |
|  | int item; |
|  | if(\*r==size-1) |
|  | printf("Queue is full\n"); |
|  | else |
|  | { |
|  | printf("Enter the value to be entered\n"); |
|  | scanf("%d", &item); |
|  | (\*r)++; |
|  | q[\*r]=item; |
|  | } |
|  | } |
|  | void dequeue(int q[], int\*r, int \*f) |
|  | { |
|  | if(\*f>\*r) |
|  | printf("Queue is empty\n"); |
|  | else |
|  | { |
|  | printf("Item deleted is: %d\n", q[\*f]); |
|  | (\*f)++; |
|  | } |
|  | } |
|  | void display(int q[], int\*r, int\*f) |
|  | { |
|  | int i; |
|  | if(\*f>\*r) |
|  | printf("Queue is empty:\n"); |
|  | for(i=\*f; i<=\*r; i++) |
|  | { |
|  | printf("%d\n", q[i]); |
|  | } |
|  | } |
|  | void main() |
|  | { |
|  | int q[20]; |
|  | int r=-1; |
|  | int f=0; |
|  | int a; |
|  | while(1) |
|  | { |
|  | printf("\n1.Enter \n"); |
|  | printf("2.Delete\n"); |
|  | printf("3.Display\n"); |
|  | printf("4.Exit\n"); |
|  | scanf("%d", &a); |
|  | switch(a) |
|  | { |
|  | case 1: enqueue(q, &r); |
|  | break; |
|  | case 2: dequeue(q,&r,&f); |
|  | break; |
|  | case 3: display(q, &r,&f); |
|  | break; |
|  | case 4: exit(0); |
|  | break; |
|  | default: printf("INVALID CHOICE\n"); |
|  | } |
|  | } |
|  | } |
|  |  |





Program 4-Circular Queue Implementation

|  |
| --- |
| #include<stdio.h> |
|  | #include<stdlib.h> |
|  | struct node |
|  | { |
|  | int data; |
|  | struct node \*next; |
|  | }; |
|  | typedef struct node \* NODE; |
|  | NODE getnode() |
|  | { |
|  | NODE p; |
|  | p=(NODE)malloc(sizeof(struct node)); |
|  | if(p!=NULL) |
|  | return p; |
|  | else |
|  | { |
|  | printf("No memory allocatipon\n"); |
|  | exit(0); |
|  | } |
|  | } |
|  | NODE enqueue(NODE head,int item) |
|  | { |
|  | NODE p,q; |
|  | q=getnode(); |
|  | q->data=item; |
|  | q->next=NULL; |
|  |  |
|  | if(head==NULL) |
|  | { |
|  |  |
|  | return q; |
|  | } |
|  | p=head; |
|  | while(p->next!=NULL) |
|  | p=p->next; |
|  | p->next=q; |
|  | return head; |
|  |  |
|  | } |
|  | NODE dequeue(NODE head) |
|  | { |
|  | NODE p=head; |
|  | if(head==NULL) |
|  | { |
|  | printf("List is empty\n"); |
|  | } |
|  | printf("Deleted element is %d \n",p->data); |
|  | head=p->next; |
|  | free(p); |
|  | return head; |
|  | } |
|  | void display(NODE head) |
|  | { |
|  | NODE p; |
|  | if(head==NULL) |
|  | { |
|  | printf("List is empty\n"); |
|  | exit(0); |
|  | } |
|  | p=head; |
|  | while(p!=NULL) |
|  | { |
|  | printf("%d\n",p->data); |
|  | p=p->next; |
|  | } |
|  | } |
|  | int main() |
|  | { |
|  | NODE p,q,head=NULL; |
|  | p=getnode(); |
|  | q=getnode(); |
|  | int option,ele,pos,value; |
|  | while(1) |
|  | { |
|  | printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n"); |
|  | printf("Enter option\n"); |
|  | scanf("%d",&option); |
|  | switch(option) |
|  | { |
|  | case 1: printf("Enter element to be inserted\n"); |
|  | scanf("%d",&ele); |
|  | head=enqueue(head,ele); |
|  | break; |
|  | case 2:head=dequeue(head); |
|  | break; |
|  | case 3:printf("Elements in list are\n"); |
|  | display(head); |
|  | break; |
|  | case 4:exit(0); |
|  | break; |
|  | default :printf("Invalid choice"); |
|  | break; |
|  |  |
|  | } |
|  | } |
|  | return 0; |
|  | } |



Program 5- Singly Linked List

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node \*next;

};

typedef struct node \*NODE;

NODE getnode()

{

NODE p;

p=(NODE)malloc (sizeof(struct node));

if(p!=NULL)

return p;

else

{

printf(" No memory allocation\n");

exit(0);

}

}

NODE insert\_front(NODE head, int item)

{

NODE p;

p=getnode();

p-> data=item;

p-> next=head;

head=p;

return head;

}

void display(NODE head)

{

NODE p;

if(head==NULL)

{

printf("List is empty\n");

}

p=head;

while(p!=NULL)

{

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

p=p->next;

}

}

NODE insert\_end(NODE head, int item)

{

NODE p,q;

q=getnode();

q-> data=item;

q-> next =NULL;

if (head==NULL)

{

return q;

}

p=head;

while(p-> next!=NULL)

p=p->next;

p-> next=q;

return head;

}

NODE insert\_pos(NODE head,int item,int pos)

{

NODE curr,prev=NULL,newn;

int count=1;

newn=getnode();

newn->data=item;

newn->next=NULL;

if(head==NULL)

{

if(pos==1)

return newn;

else

{

printf("Invalid position\n");

return 0;

}

}

if(pos==1)

{

newn->next=head;

head=newn;

return head;

}

else

{

curr=head;

while(curr!=NULL&&count!=pos)

{

prev=curr;

curr=curr->next;

count++;

}

if(count==pos)

{

prev->next=newn;

newn->next=curr;

return head;

}

else

{

printf("Invalid position\n");

return head;

}

}

}

int main()

{

NODE head=NULL;

int option,ele,pos,value;

while(1)

{

printf("1.Insert front\n2.Insert end\n3.Inset pos\n4.Display\n5.Exit\n");

printf("Enter option\n");

scanf("%d",&option);

switch(option)

{

case 1: printf("Enter element to be inserted\n");

scanf("%d",&ele);

head=insert\_front(head,ele);

break;

case 2: printf("Enter element to be inserted\n");

scanf("%d",&ele);

head=insert\_end(head,ele);

break;

case 3:printf("Enetr pos and value\n");

scanf("%d%d",&pos,&value);

head=insert\_pos(head,value,pos);

break;

case 4:printf("Elements in list are\n");

display(head);

break;

case 5:

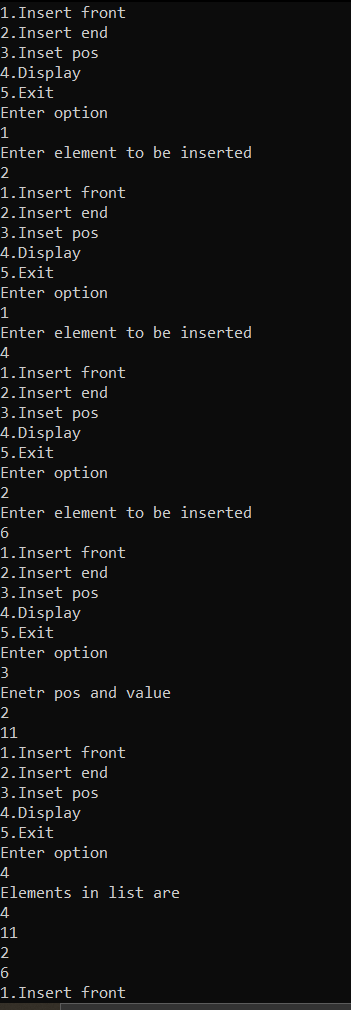
exit(0);

}

}

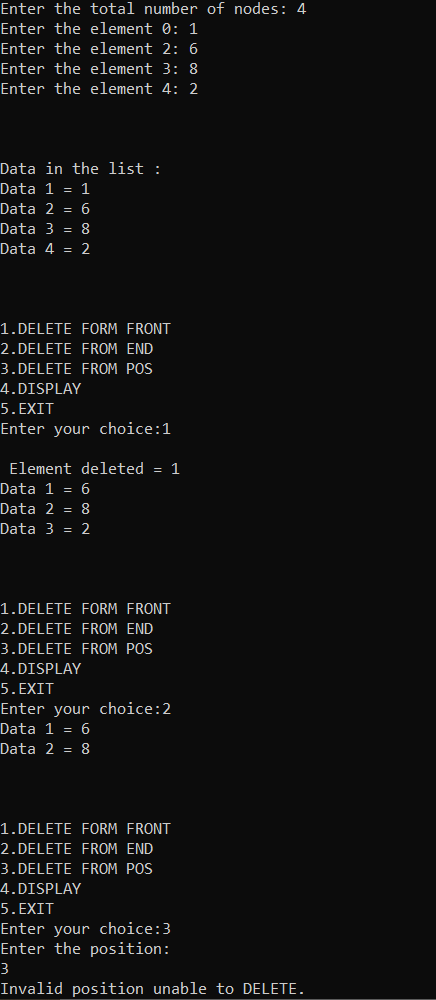
return 0;

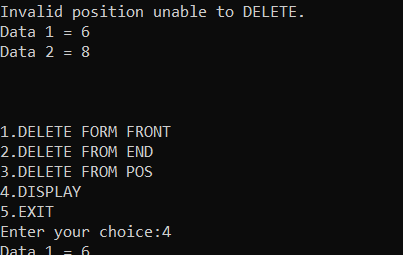
}



Program 7- Singly Linked List DELETE

|  |
| --- |
| #include <stdio.h> |
|  | #include <stdlib.h> |
|  | struct node { |
|  | int data; |
|  | struct node \*next; |
|  | }\*head; |
|  | void createList(int n) //creating n nodes |
|  | { |
|  | struct node \*newNode, \*temp; |
|  | int data, i; |
|  | head = (struct node \*)malloc(sizeof(struct node)); |
|  | if(head == NULL) |
|  | { |
|  | printf("UNABLE TO ALLOCATE MEMORY"); |
|  | } |
|  | else |
|  | { |
|  | printf("Enter the element %d: ",i); |
|  | scanf("%d", &data); |
|  |  |
|  | head->data = data; |
|  | head->next = NULL; |
|  | temp = head; |
|  | for(i=2; i<=n; i++) |
|  | { |
|  | newNode = (struct node \*)malloc(sizeof(struct node)); |
|  | if(newNode == NULL) |
|  | { |
|  | printf("Unable to allocate memory.\n"); |
|  | break; |
|  | } |
|  | else |
|  | { |
|  | printf("Enter the element %d: ", i); |
|  | scanf("%d", &data); |
|  |  |
|  | newNode->data = data; |
|  | newNode->next = NULL; |
|  |  |
|  | temp->next = newNode; |
|  | temp = temp->next; |
|  | } |
|  | } |
|  |  |
|  | printf("\n\n"); |
|  | } |
|  | } |
|  | void deleteFirstNode() //delete from first |
|  | { |
|  | struct node \*toDelete; |
|  |  |
|  | if(head == NULL) |
|  | { |
|  | printf("List is empty.\n"); |
|  | } |
|  | else |
|  | { |
|  | toDelete = head; |
|  | head = head->next; |
|  |  |
|  | printf("\n Element deleted = %d\n", toDelete->data); |
|  |  |
|  | free(toDelete); //clear memory |
|  |  |
|  |  |
|  | } |
|  | } |
|  | void deleteLastNode() //delete from end |
|  | { |
|  | struct node \*toDelete, \*secondLastNode; |
|  |  |
|  | if(head == NULL) |
|  | { |
|  | printf("List is empty.\n"); |
|  | } |
|  | else |
|  | { |
|  | toDelete = head; |
|  | secondLastNode = head; |
|  |  |
|  | while(toDelete->next != NULL) |
|  | { |
|  | secondLastNode = toDelete; |
|  | toDelete = toDelete->next; |
|  | } |
|  |  |
|  | if(toDelete == head) |
|  | { |
|  | head = NULL; |
|  | } |
|  | else |
|  | { |
|  | secondLastNode->next = NULL; |
|  | } |
|  | free(toDelete); |
|  |  |
|  |  |
|  | } |
|  | } |
|  | void deleteNode(int position) //deleting form given position |
|  | { |
|  | int i; |
|  | struct node \*toDelete, \*prevNode; |
|  |  |
|  | if(head == NULL) |
|  | { |
|  | printf("List is empty.\n"); |
|  | } |
|  | else |
|  | { |
|  | toDelete = head; |
|  | prevNode = head; |
|  |  |
|  | for(i=2; i<=position; i++) |
|  | { |
|  | prevNode = toDelete; |
|  | toDelete = toDelete->next; |
|  |  |
|  | if(toDelete == NULL) |
|  | break; |
|  | } |
|  |  |
|  | if(toDelete != NULL) |
|  | { |
|  | if(toDelete == head) |
|  | head = head->next; |
|  |  |
|  | prevNode->next = toDelete->next; |
|  | toDelete->next = NULL; |
|  | free(toDelete); |
|  |  |
|  |  |
|  | } |
|  | else |
|  | { |
|  | printf("Invalid position unable to DELETE.\n"); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void displayList() |
|  | { |
|  | struct node \*temp; |
|  | int j=1; |
|  | if(head == NULL) |
|  | { |
|  | printf("List is empty."); |
|  | } |
|  | else |
|  | { |
|  | temp = head; |
|  | while(temp != NULL) |
|  | { |
|  | printf("Data %d = %d\n",j++, temp->data); |
|  | temp = temp->next; |
|  | } |
|  | } |
|  | } |
|  | int main() |
|  | { |
|  | int n, choice,pos; |
|  | printf("Enter the total number of nodes: "); |
|  | scanf("%d", &n); |
|  | createList(n); |
|  | printf("\nData in the list :\n"); |
|  | displayList(); |
|  | while(1) |
|  | { |
|  | printf("\n\n\n"); |
|  | printf("1.DELETE FORM FRONT\n2.DELETE FROM END\n3.DELETE FROM POS\n4.DISPLAY\n5.EXIT\n"); |
|  | printf("Enter your choice:"); |
|  | scanf("%d",&choice); |
|  | switch(choice) |
|  | { |
|  | case 1:deleteFirstNode(); |
|  | displayList(); |
|  | break; |
|  | case 2:deleteLastNode(); |
|  | displayList(); |
|  | break; |
|  | case 3:printf("Enter the position:\n"); |
|  | scanf("%d",&pos); |
|  | deleteNode(pos); |
|  | displayList(); |
|  | break; |
|  |  |
|  | case 4:displayList(); |
|  | break; |
|  | case 5:exit(0); |
|  | default : printf("INVAILD CHOICE"); |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | return 0; |
|  | } |





Program 8- SLL operations

#include<stdio.h>

#include<conio.h>

#include<malloc.h>

#include<process.h>

struct node

{

int info;

struct node \*link;

};

typedef struct node \*NODE;

NODE getnode()

{

NODE x;

x=(NODE)malloc(sizeof(struct node));

if(x==NULL)

{

printf("mem full\n");

exit(0);

}

return x;

}

NODE insert\_front(NODE first,int item)

{

NODE temp;

temp=getnode();

temp->info=item;

temp->link=NULL;

if(first==NULL)

return temp;

temp->link=first;

first=temp;

return first;

}

NODE insert\_rear(NODE first,int item)

{

NODE temp,cur;

temp=getnode();

temp->info=item;

temp->link=NULL;

if(first==NULL)

return temp;

cur=first;

while(cur->link!=NULL)

cur=cur->link;

cur->link=temp;

return first;

}

NODE insert\_pos(int item,int pos,NODE first)

{

NODE temp;

NODE prev,cur;

int count;

temp=getnode();

temp->info=item;

temp->link=NULL;

if(first==NULL && pos==1)

return temp;

if(first==NULL)

{

printf("invalid pos\n");

return first;

}

if(pos==1)

{

temp->link=first;

return temp;

}

count=1;

prev=NULL;

cur=first;

while(cur!=NULL && count!=pos)

{

prev=cur;

cur=cur->link;

count++;

}

if(count==pos)

{

prev->link=temp;

temp->link=cur;

return first;

}

printf("IP\n");

return first;

}

void display(NODE first)

{

NODE temp;

if(first==NULL)

printf("list empty cannot display items\n");

for(temp=first;temp!=NULL;temp=temp->link)

{

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

}

}

void main()

{

int item,choice,pos;

NODE first=NULL;

for(;;)

{

printf("\n 1:Insert at front\n 2:Insert at rear\n 3:insert at a position\n 4:display the linked list \n 5:Exit\n");

printf("enter the choice\n");

scanf("%d",&choice);

switch(choice)

{

case 1:printf("enter the item at front-end\n");

scanf("%d",&item);

first=insert\_front(first,item);

break;

case 2:printf("enter the item at rear-end\n");

scanf("%d",&item);

first=insert\_rear(first,item);

break;

case 3:printf("enter the position\n");

scanf("%d",&pos);

printf("enter the item to be inserted \n");

scanf("%d",&item);

first=insert\_pos(item,pos,first);

break;

case 4:display(first);

break;

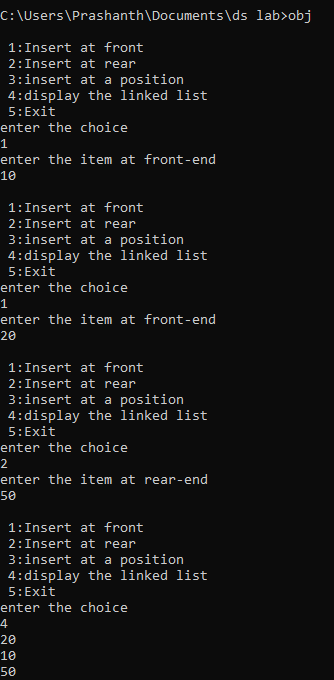
default:exit(0);

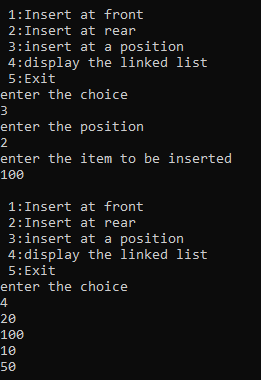
}

}

getch();

}





Program 9-Binary Tree

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node \*left;

struct node \*right;

};

typedef struct node \*NODE;

NODE getnode(int item) {

NODE p = (NODE)malloc(sizeof(struct node));

if(p!=NULL){

p->data=item;

p->left = NULL;

p->right = NULL;

return p;

}

else {

printf("Memory allocation failed!\n");

exit(0);

}

}

NODE insert(NODE root,int item)

{

if(root ==NULL)

return getnode(item);

if(item<root->data)

root->left = insert(root->left,item);

else if(item>root->data)

root->right = insert(root->right,item);

return root;

}

void inorder(NODE root){

if(root == NULL)

return;

inorder(root->left);

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

inorder(root->right);

}

void preorder(NODE root){

if(root == NULL)

return;

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

preorder(root->left);

preorder(root->right);

}

void postorder(NODE root){

if(root == NULL)

return;

postorder(root->left);

postorder(root->right);

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

}

int main(){

NODE root = NULL;

int ele,ch;

for(;;){

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

printf("1.Insert.\n2.Inorder Traversal.\n3.Preorder Traversal.\n4.Postorder Traversal.\n5.Exit:\n");

scanf("%d",&ch);

switch(ch){

case 1: printf("\nEnter the element:\n");

scanf("%d",&ele);

root = insert(root,ele);

break;

case 2: inorder(root);

break;

case 3: preorder(root);

break;

case 4: postorder(root);

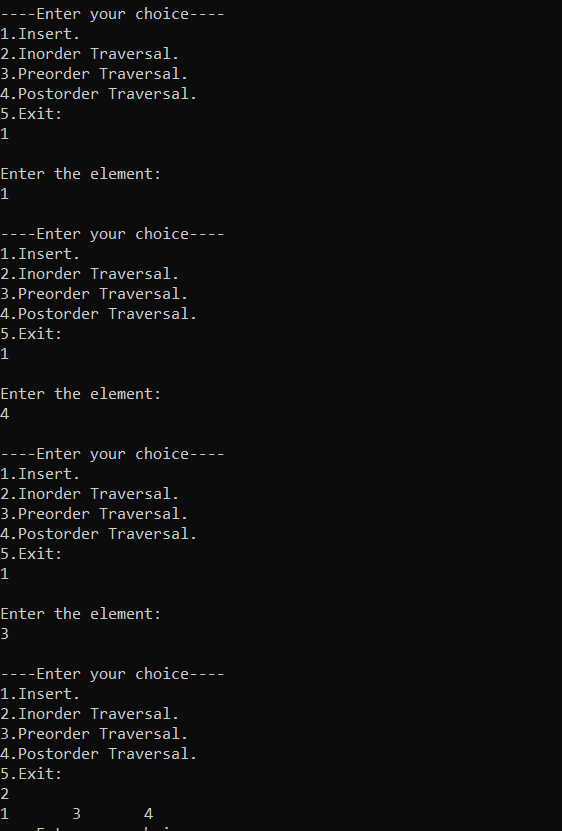
break;

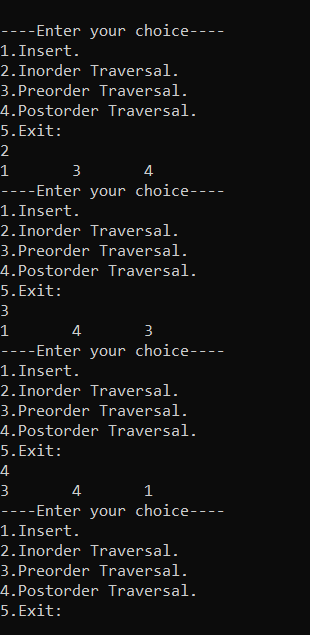
case 5: exit(1);

default :printf("Invalid Choice");

}

}

}



Program 10- Double Link List

|  |
| --- |
| #include<stdio.h> |
|  | #include<stdlib.h> |
|  |  |
|  | typedef struct node{ |
|  | int data; |
|  | struct node \*next; |
|  | struct node \*prev; |
|  | }\*Node; |
|  |  |
|  | Node create\_node(){ |
|  | Node temp; |
|  | temp=(Node)malloc(sizeof(Node)); |
|  | return temp; |
|  | } |
|  |  |
|  | Node insert\_front(Node head,int ele){ |
|  | Node p; |
|  | p=create\_node(); |
|  | p->data=ele; |
|  | p->next=head; |
|  | p->prev=NULL; |
|  | if(head!=NULL) |
|  | head->prev=p; |
|  | return p; |
|  | } |
|  |  |
|  | Node insert\_rear(Node head,int ele){ |
|  | Node p,q; |
|  | p=create\_node(); |
|  | p->data=ele; |
|  | p->next=NULL; |
|  | p->prev=NULL; |
|  | if(head==NULL) |
|  | return p; |
|  | else{ |
|  | q=head; |
|  | while(q->next!=NULL) |
|  | q=q->next; |
|  | q->next=p; |
|  | p->prev=q; |
|  | return head; |
|  | } |
|  | } |
|  |  |
|  | Node insert\_position(Node head,int ele,int pos){ |
|  | int count=1; |
|  | Node p,q,n; |
|  | p=create\_node(); |
|  | p->data=ele; |
|  | p->next=NULL; |
|  | p->prev=NULL; |
|  | if(head==NULL){ |
|  | if(pos==1) |
|  | return p; |
|  | else |
|  | printf("Invalid\n"); |
|  | } |
|  | else{ |
|  | q=head; |
|  | if(pos==1){ |
|  | p->next=head; |
|  | head->prev=p; |
|  | return p; |
|  | } |
|  | while(q!=NULL && count!=pos){ |
|  | n=q; |
|  | q=q->next; |
|  | count++; |
|  | } |
|  | if(count==pos){ |
|  | n->next=p; |
|  | p->prev=n; |
|  | p->next=q; |
|  | if(q!=NULL) |
|  | q->prev=p; |
|  | } |
|  | else |
|  | printf("Invalid\n"); |
|  | } |
|  | return head; |
|  | } |
|  |  |
|  | Node delete\_front(Node head){ |
|  | Node p; |
|  | if(head==NULL) |
|  | return head; |
|  | p=head->next; |
|  | printf("Deleted element: %d",head->data); |
|  | free(head); |
|  | if(p!=NULL) |
|  | p->prev=NULL; |
|  | return p; |
|  | } |
|  |  |
|  | Node delete\_rear(Node head){ |
|  | Node p,q; |
|  | if(head==NULL) |
|  | return head; |
|  | if(head->next==NULL){ |
|  | printf("Deleted element: %d\n",head->data); |
|  | free(head); |
|  | return NULL; |
|  | } |
|  | p=head; |
|  | while(p->next!=NULL){ |
|  | q=p; |
|  | p=p->next; |
|  | } |
|  | printf("Deleted element: %d\n",p->data); |
|  | q->next=NULL; |
|  | free(p); |
|  | return head; |
|  | } |
|  |  |
|  | Node delete\_position(Node head,int pos){ |
|  | Node p,q; |
|  | int c=1; |
|  | if(head==NULL) |
|  | return head; |
|  | if(pos==1) |
|  | return(delete\_front(head)); |
|  | p=head; |
|  | while(p->next!=NULL && c!=pos){ |
|  | q=p; |
|  | p=p->next; |
|  | c++; |
|  | } |
|  | if(c==pos){ |
|  | printf("Deleted element: %d\n",p->data); |
|  | q->next=p->next; |
|  | if(p->next!=NULL){ |
|  | (p->next)->prev=q; |
|  | } |
|  | free(p); |
|  | } |
|  | else |
|  | printf("Invalid choice\n"); |
|  | return head; |
|  | } |
|  |  |
|  | void display(Node head){ |
|  | Node p; |
|  | p=head; |
|  | if(head==NULL) |
|  | printf("List is empty\n"); |
|  | else{ |
|  | printf("Elements: "); |
|  | while(p!=NULL){ |
|  | printf("%d ",p->data); |
|  | p=p->next; |
|  | } |
|  | } |
|  | printf("\n"); |
|  | } |
|  |  |
|  | int main(){ |
|  | Node head=NULL; |
|  | int option,ele,pos; |
|  | do{ |
|  | printf("\n1)Insert Rear\n2)Insert Front\n3)Insert At Position\n4)Delete front\n5)Delete Rear\n6)Delete at position\n7)Exit\n"); |
|  | printf("Enter your option: "); |
|  | scanf("%d",&option); |
|  | switch(option){ |
|  | case 1: printf("Enter the element to be inserted: "); |
|  | scanf("%d",&ele); |
|  | head=insert\_rear(head,ele); |
|  | display(head); |
|  | break; |
|  | case 2: printf("Enter the element to be inserted: "); |
|  | scanf("%d",&ele); |
|  | head=insert\_front(head,ele); |
|  | display(head); |
|  | break; |
|  | case 3: printf("Enter the element to be inserted and position: "); |
|  | scanf("%d %d",&ele,&pos); |
|  | head=insert\_position(head,ele,pos); |
|  | display(head); |
|  | break; |
|  | case 4: head=delete\_front(head); |
|  | display(head); |
|  | break; |
|  | case 5: head=delete\_rear(head); |
|  | display(head); |
|  | break; |
|  | case 6: printf("Enter the position: "); |
|  | scanf("%d",&pos); |
|  | head=delete\_position(head,pos); |
|  | display(head); |
|  | break; |
|  | case 7: break; |
|  | default: printf("Invalid choice\n"); |
|  | } |
|  | }while(option!=7); |
|  | return 0; |
|  | } |

