**----------------------------------PRG:-01 ----------------------------------**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct Day {

char\* dayName;

int date;

char\* activity;

};

struct Day\* createCalendar()

{

struct Day\* calendar = (struct Day\*)malloc(7 \* sizeof(struct Day));

for (int i = 0; i < 7; i++)

{

calendar[i].dayName = (char\*)malloc(20 \* sizeof(char));

calendar[i].activity = (char\*)malloc(100 \* sizeof(char));

}

return calendar;

}

void readCalendarData(struct Day\* calendar)

{

for (int i = 0; i < 7; i++)

{

printf("Enter the day name for Day %d: ", i + 1);

scanf("%s", calendar[i].dayName);

printf("Enter the date for Day %d: ", i + 1);

scanf("%d", &calendar[i].date);

printf("Enter the activity for Day %d: ", i + 1);

scanf(" %s", calendar[i].activity);

}

}

void displayCalendar(struct Day\* calendar)

{

printf("Weekly Activity Report:\n\n");

for (int i = 0; i < 7; i++)

{

printf("Day %d: %s\n", i + 1, calendar[i].dayName);

printf("Date: %d\n", calendar[i].date);

printf("Activity: %s\n", calendar[i].activity);

printf("\n");

}

}

int main()

{

struct Day\* calendar = createCalendar();

readCalendarData(calendar);

displayCalendar(calendar);

for (int i = 0; i < 7; i++)

{

free(calendar[i].dayName);

free(calendar[i].activity);

}

free(calendar);

return 0;

}

**---------------------------------- PRG:-02 ----------------------------------**

#include<stdio.h>

void read();

void match();

char STR[100],PAT[100],REP[100],ANS[100];

int c,i,j,k,m,flag=0;

void main()

{

read();

match();

}

void read()

{

printf("enter the main string STR:");

gets(STR);

printf("enter pattern string PAT:");

gets(PAT);

printf("enter replace string REP:");

gets(REP);

}

void match()

{

c=i=j=k=m=0;

while(STR[c]!='\0')

{

if(STR[m]==PAT[i])

{

i++;

m++;

flag=1;

if(PAT[i]=='\0')

{

for(k=0;REP[k]!='\0';k++,j++)

ANS[j]=REP[k];

i=0;

c=m;

}

}

else

{

ANS[j]=STR[c];

j++;

c++;

m=c;

i=0;

}

}

if(flag==0)

printf("pattern not found");

else

{

ANS[j]='\0';

printf("resultant string is %s",ANS);

}

}

**----------------------------------PRG:-03 ----------------------------------**

#include<stdio.h>

#include<stdlib.h>

#define MAX 4

Int stack[MAX],top=-1,item;

Void push();

Void pop();

Void palindrome();

Void display();

Void main()

{

Int choice;

While(1)

{

Printf(“\n―------- STACK OPERATIONS ------\n”);

Printf(“1.push\n 2.pop\n 3.palindrome\n 4.display\n 5.exit\n”);

Printf(“enter choice\t”);

Scanf(“%d”,&choice);

Switch(choice)

{

Case 1:push();

Break;

Case 2:pop();

Break;

Case 3:palindrome();

Break;

Case 4:display();

Break;

Case 5:exit(0);

Break;

Default:printf(“invalid choice\n”);

Break;

}

}

}

Void push()

{

If(top==MAX-1)

Printf(“stack overflow”);

Else

{

Printf(“enter the item to be pushed\n”);

Scanf(“%d”,&item);

Top=top+1;

Stack[top]=item;

}

}

Void pop()

{

If(top==-1)

Printf(“stack underflow”);

Else

{

Item=stack[top];

Top=top-1;

Printf(“deleted item is %d/t”,item);

}

}

Void display()

{

Int I;

If(top= =-1)

Printf(“stack is empty”);

Else

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

Printf(“%d\t”,stack[i]);

}

Void palindrome()

{

Int num[10],i=0,k,flag=1; k=top;

While(k!=-1)

Num[i++]=stack[k--];

For(i=0;i<=top;i++)

{

If(num[i]==stack[i])

Continue;

Else

Flag=0;

}

If(top==-1)

Printf(“stack is empty”);

Else

{

If(flag)

Printf(“palindrome”);

Else

Printf(“not a palindrome”);

}

}

**----------------------------------PRG:- 0 4---------------------------------**

#include<stdio.h>

#include<ctype.h>

#define SIZE 50

char s[SIZE];

int top=-1;

void push(char elem)

{

s[++top]=elem;

}

char pop()

{

return s[top--];

}

int pr(char elem)

{

switch(elem)

{

case '#':return 0;

case '(':return 1;

case '+':

case '-':return 2;

case '\*':

case '/':

case '%':return 3;

case '^':return 4;

}

}

void main()

{

char infix[50],postfix[50],ch,elem;

int i=0,k=0;

printf("enter the infix expression\n");

gets(infix);

push('#');

while((ch=infix[i++])!='\0')

{

if(ch=='(')

push(ch);

else if(isalnum(ch))

postfix[k++]=ch;

else if(ch==')')

{

while(s[top]!='(')

postfix[k++]=pop();

elem=pop();

}

else

{

while(pr(s[top])>=pr(ch))

postfix[k++]=pop();

push(ch);

}

}

while(s[top]!='#')

postfix[k++]=pop();

postfix[k]='\0';

printf("infix expression is %s\n postfix expression is %s\n",infix, postfix);

}

**---------------------------------- PRG:-05 ----------------------------------**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<math.h>

#define MAX 50

char post[MAX];

int stack[MAX],top=-1,i;

void pushstack(int);

void calculator(char);

main()

{

printf("entersuffix expression\n");

gets(post);

for(i=0; i<strlen(post); i++)

{

if(post[i]>'0'&&post[i]<='9')

pushstack(i);

else

calculator(post[i]);

}

printf("result=%d\n",stack[top]);

}

void pushstack(int i)

{

top=top+1;

stack[top]=(int)

(post[i]-48);

}

void calculator(char c)

{

int a,b,ans;

b=stack[top--];

a=stack[top--];

switch©

{

Case ‘+’:ans=a+b;break;

Case ‘-‘:ans=a-b;break;

Case ‘\*’:ans=a\*b;break;

Case ‘/’:ans=a/b;break;

Case’%’:ans=a%b;break;

Case ‘^’:ans=pow(a,b);break;

Default :printf(“wrong input\n”);

Exit(0);

}

Top++;

Stack[top]=ans;

}

**---------------------------------- PRG:-06 ----------------------------------**

#include<stdio.h>

#include<stdlib.h

> #define MAX 5

Char

q[MAX],item;

int f=0,r=-1,count=0;

void insert();

void delete();

void display();

main()

{

int ch;

while(1)

{

printf("1.insert 2.delete 3.display 4.exit \n");

printf("enter choice\n");

scanf("%d",&ch);

switch(ch)

{

case 1:getchar();

insert();

break;

case 2:delete();

break;

case3:display();

break

case 4:exit(0);

default :

printf("Invalid choice\n");

break;

}

}

}

void insert()

{

if(count==MAX)

printf("queue overflow\n");

else

{

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

scanf("%c", &item );

r=(r+1)%MAX;

q[r]=item;

count++;

}

}

void delete()

{

if(count==0)

printf("queue underflow\n");

else

{

printf("deleted item is%c\n",q[f]);

f=(f+1)%MAX;

count--;

}

}

void display()

{

intj=f,i;

if(count==0) printf("queue is empty\n");

else

{

printf("contents of circular queue\n");

for(i=1;i<=count;i++)

{

printf("%c\t",q[j]);

j=(j+1)%MAX;

}

printf("total number of items=%d\n",count);

}

}

**---------------------------------- PRG:-07 ----------------------------------**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

void create();

void insert\_front();

void insert\_rear();

void display();

void delete\_front();

void delete\_rear();

int count=0; struct

node{

char usn[20],name[50],branch[10];

int sem;

unsigned long long int

phno; structnode\*link;

};

Struct node\*first=NULL,\*last=NULL,\*temp=NULL,\*p;

void main()

{

int ch,n,i;

while(1)

{

printf("1.create SLL 2.insert at front 3.insert at rear 4.display 5.delete at front 6.delete at

rear 7.exit\n");

printf("enter choice\n");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("enter the no. of students\n");

scanf("%d",&n);

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

insert\_front();

break;

case2: insert\_front();

break;

case3:insert\_rear();

break;

case 4:display();

break;

case 5:delete\_front();

break;

case 6:delete\_rear();

break;

case 7:exit(0);

default: printf("invalid choice\n");

break;

}

}

}

void create()

{

char

usn[20],name[50],branch[10];

int sem;

unsigned long long int phno;

temp=(struct node\*)malloc(sizeof(struct node)); printf("enter

usn,name,branch,sem,phno\n");

scanf("%s%s%s%d%llu",usn,name,branch,&sem,&phno);

strcpy(temp->usn,usn);

strcpy(temp>name,nam;

strcpy(temp>branch,brh;

temp->sem=sem;

temp>phno=phno;

count++;

}

void insert\_front()

{

if(first==NULL)

{

create();

temp>link=NULL;

first=temp;

last=temp;

}

else

{

create();

temp>link=first;

first=temp;

}

}

void insert\_rear()

{

if(first==NULL)

{

create();

temp>link=NULL;

first=temp;

last=temp;

}

else

{

create();

temp>link=NULL;

last->link=temp;

last=temp;

}

}

void display()

{

if(first==NULL)

{

printf("list is empty\n");

}

else

{

p=first;

printf("content of list is\n");

while(p!=NULL)

{

printf("%s\t%s\t%s\t%d\t%llu\n",p->usn,p->name,p->branch,p->sem,p->phno);

p=p->link;

}

printf("total no.of students %d\n",count);

}

}

void delete\_front()

{

p=first;

if(first==NULL)

{

printf("list is empty\n");

}

else if(p->link==NULL)

{

printf("deleted node is %s\t%s\t%s\t%d\t%llu\n",p->usn,p->name,p-

>branch,p->sem,p->phno); free(p);

first=NULL;

count--;

}

else

{

first=p->link;

printf("deleted node is %s\t%s\t%s\t%d\t%llu\n",p->usn,p->name,p->branch,

p->sem,p->phno); free(p);

count--;

}

}

void delete\_rear()

{

p=first;

if(first==NULL)

{

printf("list is empty\n");

}

else if(p->link==NULL)

{

printf("deleted node is %s\t%s\t%s\t%d\t%llu\n",p->usn,p->name,p->branch,p->sem,

p->phno); free(p);

first=NULL;

count--;

}

else

{

while(p->link!=last)

p=p->link;

printf("deleted node is %s\t%s\t%s\t%d\t%llu\n",last->usn,last->name,last->branch,

last->sem,last->phno);

free(last);

p->link=NULL;

last=p;

count--;

}

}

**---------------------------------- PRG:-08 ----------------------------------**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void create();

void insert\_front();

void insert\_rear();

void display();

void delete\_front();

void delete\_rear();

int count = 0;

struct node {

int ssn;

char name[50];

char dept[20];

char desg[20];

float sal;

unsigned long long int phno;

struct node \*llink, \*rlink;

};

struct node \*first = NULL, \*last = NULL, \*temp;

int main() {

int ch, n, i;

while (1) {

printf("1. Create\n2. Insert at Front\n3. Insert at Rear\n4. Display\n5. Delete from Front\n6. Delete from Rear\n7. Exit\n");

printf("Enter choice: ");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter the number of employees: ");

scanf("%d", &n);

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

insert\_rear();

break;

case 2:

insert\_front();

break;

case 3:

insert\_rear();

break;

case 4:

display();

break;

case 5:

delete\_front();

break;

case 6:

delete\_rear();

break;

case 7:

exit(0);

default:

printf("Invalid choice\n");

break;

}

}

return 0;

}

void create() {

int ssn;

char name[50], dept[20], desg[20];

float sal;

unsigned long long int phno;

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

temp->llink = temp->rlink = NULL;

printf("Enter SSN, Name, Department, Designation, Salary, and Phone Number:\n");

scanf("%d %s %s %s %f %llu", &ssn, name, dept, desg, &sal, &phno);

temp->ssn = ssn;

strcpy(temp->name, name);

strcpy(temp->dept, dept);

strcpy(temp->desg, desg);

temp->sal = sal;

temp->phno = phno;

count++;

}

void insert\_front() {

create();

if (first == NULL) {

first = temp;

last = temp;

} else {

temp->rlink = first;

first->llink = temp;

first = temp;

}

printf("Employee added at the front.\n");

}

void insert\_rear() {

create();

if (first == NULL) { // List is empty

first = temp;

last = temp;

} else {

last->rlink = temp;

temp->llink = last;

last = temp;

}

printf("Employee added at the rear.\n");

}

void display() {

struct node \*p;

if (first == NULL) {

printf("List is empty\n");

return;

}

p = first;

printf("Contents of the list:\n");

while (p != NULL) {

printf("%d\t%s\t%s\t%s\t%.2f\t%llu\n", p->ssn, p->name, p->dept, p->desg, p->sal, p->phno);

p = p->rlink;

}

printf("Total number of employees: %d\n", count);

}

void delete\_front() {

if (first == NULL) {

printf("List is empty. No nodes to delete.\n");

return;

}

temp = first;

if (first == last) {

first = last = NULL;

} else {

first = first->rlink;

first->llink = NULL;

}

free(temp);

count--;

printf("Node deleted from the front.\n");

}

void delete\_rear() {

if (first == NULL) {

printf("List is empty. No nodes to delete.\n");

return;

}

temp = last;

if (first == last) { // Only one node

first = last = NULL;

} else {

last = last->llink;

last->rlink = NULL;

}

free(temp);

count--;

printf("Node deleted from the rear.\n");

}

**---------------------------------- PRG:-09 ----------------------------------**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int coeff; // Coefficient

int x\_exp; // Exponent of x

int y\_exp; // Exponent of y

int z\_exp; // Exponent of z

struct Node \*next;

} Node;

Node\* createNode(int coeff, int x\_exp, int y\_exp, int z\_exp) {

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

newNode->coeff = coeff;

newNode->x\_exp = x\_exp;

newNode->y\_exp = y\_exp;

newNode->z\_exp = z\_exp;

newNode->next = NULL;

return newNode;

}

void insertTerm(Node \*\*head, int coeff, int x\_exp, int y\_exp, int z\_exp) {

Node \*newNode = createNode(coeff, x\_exp, y\_exp, z\_exp);

if (\*head == NULL) { // Create circular linked list if empty

\*head = newNode;

newNode->next = \*head;

} else {

Node \*temp = \*head;

while (temp->next != \*head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = \*head;

}

}

void displayPoly(Node \*head) {

if (head == NULL) {

printf("Polynomial is empty.\n");

return;

}

Node \*temp = head;

do {

printf("%+dx^%dy^%dz^%d ", temp->coeff, temp->x\_exp, temp->y\_exp, temp->z\_exp);

temp = temp->next;

} while (temp != head);

printf("\n");

}

Node\* addPolynomials(Node \*poly1, Node \*poly2) {

Node \*polySum = NULL;

if (poly1 == NULL && poly2 == NULL) return NULL;

Node \*p1 = poly1, \*p2 = poly2;

if (p1 != NULL) {

do {

insertTerm(&polySum, p1->coeff, p1->x\_exp, p1->y\_exp, p1->z\_exp);

p1 = p1->next;

} while (p1 != poly1);

}if (p2 != NULL) {

do {

insertTerm(&polySum, p2->coeff, p2->x\_exp, p2->y\_exp, p2->z\_exp);

p2 = p2->next;

} while (p2 != poly2);

}

return polySum;

}

void inputPolynomial(Node \*\*head) {

int n, coeff, x\_exp, y\_exp, z\_exp;

printf("Enter the number of terms in the polynomial: ");

scanf("%d", &n);

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

printf("Enter term %d (coefficient, x\_exp, y\_exp, z\_exp): ", i + 1);

scanf("%d %d %d %d", &coeff, &x\_exp, &y\_exp, &z\_exp);

insertTerm(head, coeff, x\_exp, y\_exp, z\_exp);

}

}

int main() {

Node \*poly1 = NULL, \*poly2 = NULL, \*polySum = NULL;

printf("Input POLY1:\n");

inputPolynomial(&poly1);

printf("POLY1: ");

displayPoly(poly1);

printf("\nInput POLY2:\n");

inputPolynomial(&poly2);

printf("POLY2: ");

displayPoly(poly2);

printf("\nAdding POLY1 and POLY2 to get POLYSUM:\n");

polySum = addPolynomials(poly1, poly2);

printf("POLYSUM: ");

displayPoly(polySum);

return 0;

}

**---------------------------------- PRG:-10 ----------------------------------**

#include<stdio.h>

#include<stdlib.h>

typedef struct bst

{

int data;

struct bst \*rchild, \*lchild;

} node;

node\* getnode();

void insert(node\*, node\*);

void inorder(node\*);

void preorder(node\*);

void postorder(node\*);

int search(node\*, int);

int n;

node\* getnode()

{

node \*temp;

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

temp->lchild = NULL;

temp->rchild = NULL;

return temp;

}

void insert(node \*root, node \*newnode)

{

if (root->data < newnode->data)

{

if (root->rchild == NULL)

root->rchild = newnode;

else

insert(root->rchild, newnode);

}

else

{

if (root->lchild == NULL)

root->lchild = newnode;

else

insert(root->lchild, newnode);

}

}

void inorder(node \*ptr)

{

if (ptr != NULL)

{

inorder(ptr->lchild);

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

inorder(ptr->rchild);

}

}

void preorder(node \*ptr)

{

if (ptr != NULL)

{

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

preorder(ptr->lchild);

preorder(ptr->rchild);

}

}

void postorder(node \*ptr)

{

if (ptr != NULL)

{

postorder(ptr->lchild);

postorder(ptr->rchild);

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

}

}

int search(node \*root, int key)

{

node \*temp;

temp = root;

if (root != NULL)

{

if (temp->data == key)

n = 1;

else if (key > temp->data)

search(temp->rchild, key);

else

search(temp->lchild, key);

}

else

n = 0;

return n;

}

int main()

{

int ch;

int key, ans = 1;

node \*newnode, \*root = NULL, \*temp, \*parent;

while(1)

{

printf("1.create 2.search 3.travers 4.exit\n");

printf("enter choice\n");

scanf("%d", &ch);

switch(ch)

{

case 1:

while(ans == 1)

{

newnode = getnode();

printf("enter the element\n");

scanf("%d", &newnode->data);

if(root == NULL)

root = newnode;

else

insert(root, newnode);

printf("want to continue[0/1]\n");

scanf("%d", &ans);

}

break;

case 2:

printf("enter the element to search\n");

scanf("%d", &key);

search(root, key);

if(n == 1)

printf("search successful\n");

else

printf("search unsuccessful\n");

break;

case 3:

if(root == NULL)

printf("tree is empty\n");

else

{

printf("\n inorder traversal\n");

inorder(root);

printf("\n postorder traversal\n");

postorder(root);

printf("\n preorder traversal\n");

preorder(root);

}

break;

case 4:

exit(0);

break;

default:

printf("wrong choice\n");

break;

}

}

return 0;

}

**---------------------------------- PRG:-11 ----------------------------------**

#include <stdio.h>

#include <stdlib.h>

int n, a[10][10], i, j, source, s[10], choice, count;

void bfs(int n, int a[10][10], int source, int s[]) {

int q[10], u;

int front = 0, rear = 0;

s[source] = 1;

q[rear] = source;

rear++;

while (front < rear)

{

u = q[front];

front++;

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

if (a[u][i] == 1 && s[i] == 0) {

rear++;

q[rear] = i;

s[i] = 1;

}

}

}

}

void dfs(int n, int a[10][10], int source, int s[]) {

s[source] = 1;

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

if (a[source][i] == 1 && s[i] == 0) {

dfs(n, a, i, s);

}

}

}

int main() {

printf("Enter the number of nodes: \n");

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]);

}

}

while (1)

{

printf("\n1. BFS\n2. DFS\n3. Exit\n");

printf("\nEnter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("\nEnter the source node: ");

scanf("%d", &source);

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

s[i] = 0;

}

bfs(n, a, source, s);

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

if (s[i] == 0) {

printf("\nThe node %d is not reachable", i);

} else {

printf("\nThe node %d is reachable", i);

}

}

break;

case 2:

printf("\nEnter the source node: ");

scanf("%d", &source);

count = 0;

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

s[i] = 0;

}

dfs(n, a, source, s);

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

{

if (s[i])

{

count++;

}

}

if (count == n) {

printf("\nThe graph is connected.");

} else {

printf("\nThe graph is not connected.");

}

break;

case 3:

exit(0);

default:

printf("\nInvalid choice. Please try again.");

}

}

return 0;

}

**---------------------------------- PRG:-12 ----------------------------------**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

void display(int a[MAX]);

int create(int num);

void linearprob(int a[MAX], int key, int num);

void main() {

int a[MAX], i, num, key, ans = 1;

printf("Collision handling by linear probing\n");

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

a[i] = -1;

do {

printf("Enter the data:\n");

scanf("%d", &num);

key = create(num);

linearprob(a, key, num);

printf("Do you want to continue [1/0]?\n");

scanf("%d", &ans);

} while (ans);

display(a);

}

int create(int num) {

int key;

key = num % 100;

return key;

}

void linearprob(int a[MAX], int key, int num) {

int flag = 0, i;

if (a[key] == -1)

a[key] = num;

else {

printf("\nCollision detected\n");

i = key;

while (flag == 0) {

i = (i + 1) % MAX;

if (a[i] == -1) {

a[i] = num;

flag = 1;

}

}

}

}

void display(int a[MAX]) {

int ch, i;

printf("\n1. Display all\n2. Filtered display\n");

printf("Enter choice:\n");

scanf("%d", &ch);

if (ch == 1) {

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

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

}

} else {

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

if (a[i] != -1) {

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

}

}

}

}