1 Array

```c
#include<stdio.h>
#include<stdlib.h>
int a[10],n;
void create();
void insert();
void del();
void display();
int main()
{
    int ch;
    printf("array creation\n");
    create();
do
    {
        printf("1.insert\n2.delete\n3.display\n4.exit\n");
        printf("enter your choice:");
        scanf("%d",&ch);
            switch(ch)
                {
                    case 1 :insert();
                        break;
                    case 2 :del();
                        break;
                    case 3 :display();
                        break;
                    case 4 :exit(0);
                    default:printf("invalid choice");
                }
    }while(ch!=4);
   return 0;
}

void  create()
{
    int i;
    printf("enter the no of elements:");
    scanf("%d",&n);
    printf("enter the array elements:\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
}
void display()
{
```

```
    int i;
    printf("array elements are\n");
    for(i=0;i<n;i++)
        printf("%d",a[i]);
}
void insert()
{
    int i,e,pos;
    printf("enter the position where element is to be inserted\n");
    scanf("%d",&pos);
    if(pos=!n){

    printf("it is not done\n ");}
    else{

    printf("it is done");}
    printf("enter the element to be inserted:");
    scanf("%d",&e);
    for(i=n;i>pos;i--)
        a[i]=a[i-1];
    a[pos]=e;
    n=n+1;
}

void del()
{
    int  i,e,pos;
    printf("enter the position to be deleted\n");
    scanf("%d",&pos);
    e=a[pos];
    for(i=pos;i<n-1;i++)
        a[i]=a[i+1];
    n=n-1;
    printf("deleted element is %d",e);
}
```

2 stack

```c
#include<stdio.h>
#include<stdlib.h>
#define MAX 5
int s[MAX];
int top=-1;
void push();
int pop();
void display();
int isoverflow();
int isunderflow();

int main()
{
    int ch,e;
    do
{
            printf("\n1.push\n2.pop\n3.display\n4.exit\n");
            printf("enter your choice:");
            scanf("%d",&ch);
                switch(ch)
                    {
                        case 1:if(isoverflow())
                                printf("stack overflow");
                            else
                                push();
                            break;
                        case 2:if(isunderflow())
                                printf("stack underflow");
                            else
                                {
                                e=pop();
                                printf("deleted element is %d",e);
                                }
                            break;
                        case 3:if(isunderflow())
                                printf("stack underflow");
                            else
                                display();
                            break;

                        case 4:exit(0);
                        default:printf("invalid choice\n");
                    }
        }while(ch!=4);
```

```
    return 0;
}

int isoverflow()
{
    if(top==MAX-1)
        return 1;
    else
        return 0;
}

int isunderflow()
{
    if(top==-1)
        return 1;
    else
        return 0;
}

void push()
{
    int e;
    printf("enter the element to be pushed:");
    scanf("%d",&e);
    s[++top]=e;
}

int pop()
{
    int e;
    e=s[top--];
    return e;
}

void display()
{
    int i;
    printf("content of stack are\n");
    for(i=top;i>=0;i--)
        printf("%d\n",s[i]);
}
```

3 infix to postfix

```c
#include<stdio.h>
#include<ctype.h>
char s[20];
int top=-1;
void push(char);
char pop();
int prior(char);
main()
{
    char infix[20],postfix[20];
    int i,j=0;
printf("enter the infix expression:");
    scanf("%s",infix);
    push('#');
        for(i=0;infix[i]!='\0';i++)
            {
                if(isalnum(infix[i]))
                    postfix[j++]=infix[i];
                else if(infix[i]=='(')
                    push(infix[i]);
                else if(infix[i]==')')
                    {
                        while(s[top]!='(')
                            postfix[j++]=pop();
                        pop();
                    }
                else
                    {
                        while(prior(s[top])>=prior(infix[i]))
                            postfix[j++]=pop();
                        push(infix[i]);
                    }
            }
    while(s[top]!='#')
        postfix[j++]=pop();
    postfix[j]='\0';
    printf("postfix expression is:%s",postfix);
return 0;
}

void push(char x)
{
    s[++top]=x;
}
char pop()
{
    return(s[top--]);
```

```
}

int prior(char x)
{
    if(x=='^')
    return 3;
    if(x=='*'||x=='%'||x=='/')
    return 2;
    if(x=='+'||x=='-')
    return 1;
    if(x=='('||x=='#')
    return 0;
}
```

4 postfix exp

```
#include<stdio.h>
#include<ctype.h>
#include<math.h>
int s[20];
int top=-1;
void push(int);
int pop();
int main()
{
    char postfix[20];
    int op1,op2,res,i;
    printf("enter the postfix expression:");
    scanf("%s",postfix);
    for(i=0;postfix[i]!='\0';i++)
        {
            if(isdigit(postfix[i]))
                push(postfix[i]-'0');
            else
                {
                    op2=pop();
                    op1=pop();
                        switch(postfix[i])
                            {
                                case '+':res=op1+op2;
                                    push(res);
                                    break;
                                case '-':res=op1-op2;
                                    push(res);
                                break;
                                case '*':res=op1*op2;
                                    push(res);
                                    break;
                                case '/':res=op1/op2;
```

```
                                    push(res);
                                    break;
                                case '%':res=op1%op2;
                                    push(res);
                                    break;
                                case '^':res=pow(op1,op2);
                                    push(res);
                                    break;
                            }
                    }
            }
        printf("postfix expression is:%d",res);
return 0;
}
void push(int ch)
{
    s[++top]=ch;
}

int pop()
{
    return(s[top--]);
}
```

5 a fabanaci series

```
#include<stdio.h>
int fib(int n);
 void main()
{
    int n,i;
    printf("Enter the limit");
    scanf("%d",&n);
    for(i=0;i<=n;i++)
    {
        printf("%d\t",fib(i));
    }
    return(0);
}
int fib(int i)
{
    if(i==0||i==1)
            return(i);
    else
       return(fib(i-1)+fib(i-2));
    }
```

5 b Tower of hanoi

```c
#include<stdio.h>
void tower(int,char,char,char);
int main()
{
    int n;
    printf("enter no of disks:");
    scanf("%d",&n);
        if(n==0)
        {
            printf("no disk found\n");
            return 0;
        }
    printf("moves involed in tower of hanio\n");
    tower(n,'A','C','B');
return 0;
}

void tower(int n,char source,char dest,char temp)
{
    if(n==1)
    {
        printf("move %d disk from %c to %c\n",n,source,dest);
        return;
    }
    tower(n-1,source,temp,dest);
    printf("move %d disk from %c to %c\n",n,source,dest);
    tower(n-1,temp,dest,source);
}
```

6 linner queue

```c
#include<stdio.h>
#include<stdlib.h>
#define MAX 5
int q[MAX],f=-1,r=-1;
void insert();
void rem();
void display();
main()
{
    int ch;
    do
    {
        printf("1.Insert\n2.Remove\n3.Display\n4.Exit\n");
        printf("Enter your choice:");
        scanf("%d",&ch);
```

```
        switch(ch)
        {
            case 1:insert();
                break;
            case 2:rem();
                 break;
            case 3:display();
                 break;
            case 4:exit(0);
            default: printf("invalid choice");
        }
    }while(ch!=4);
    return(0);
}
void insert()
{
    int e;
    if(r==MAX-1)
    {
        printf("Queue is full");
        return;
    }
    printf("Enter the element to be inserted:");
    scanf("%d",&e);
    r=r+1;
    q[r]=e;
    if(f==-1)
    f=0;
}
void rem()
{
    int e;
    if(f==-1)
    {
        printf("Queue is empty");
        return;
    }
    e=q[f];
    if(f==r)
        f=r=-1;
    else
        f=f+1;
    printf("deleted element is:%d",e);
}
void display()
{
    int i;
    if(f==-1)
```

```
    {
        printf("Queue is empty");
        return;
    }
    printf("Elements of queue are:");
    for(i=f;i<=r;i++)
     printf("%d",q[i]);
}
```

Circular cq

```
#include<stdio.h>
#include<stdlib.h>
#define MAX 3
char cq[20];
int  f=-1,r=-1;
void insert();
void del();
void display();
int  main()
{
    int ch;
    do
    {
        printf("1.insert\n2.delete\n3.display\n4.exit\n");
        printf("enter your choice:");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:insert();
            break;
            case 2:del();
            break;
            case 3:display();
            break;
            case 4:exit(0);
            default:printf("invalid choice");
        }
    }while(ch!=4);
    return 0;
}

void insert()
{
    char e;
    if((r+1)%MAX==f)
    {
```

```
        printf("cq is full");
        return;
    }
    printf("enter the item:");
    scanf(" %c",&e);  /*give one space before %c*/
    r=(r+1)%MAX;
    cq[r]=e;
    if(f==-1)
    f=0;
}


void del()
{
    char e;
    if(f==-1)
    {
        printf("cq is empty");
        return;
    }
    else
    {
        e=cq[f];
        if(f==r)
           f=r=-1;
        else
           f=(f+1)%MAX;
        printf("deleted item:%c",e);
    }
}

void display()
{
    int i;
    if(f==-1)
    {
        printf("cq is empty");
        return;
    }
    else
    {
        i=f;
        while(i!=r)
        {
            printf("%c\n",cq[i]);
            i=(i+1)%MAX;
        }
        printf("%c\n",cq[r]);
}}
```

8 linked list

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

typedef struct NODE
{
    char usn[10];
    char name[10];
    char branch[10];
    int sem;
    int phno;
    struct NODE *next;
} node;

node *first = NULL;
node *read_data();
void front_insert();
void front_del();
void end_insert();
void end_del();
void display();

int main()
{
    int ch;
    do
    {
        printf("1.front_insert\n2.front_delete\n3.end_insert\n4.end_delete\n5.
display\n6.exit\n");
        printf("enter your choice:");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            front_insert();
            break;
        case 2:
            front_del();
            break;
        case 3:
            end_insert();
            break;
        case 4:
            end_del();
            break;
        case 5:
            display();
            break;
```

```
        case 6:
            exit(0);
        default:
            printf("invalid choice");
        }
    } while (ch != 6);
    return 0;
}

node *read_data()
{

    node *nn;
    nn = (node *)malloc(sizeof(node));
    printf("enter the usn:");
    scanf("%s", nn->usn);
    printf("enter the name:");
    scanf("%s", nn->name);
    printf("enter the branch:");
    scanf("%s", nn->branch);
    printf("enter the semester:");
    scanf("%d", &nn->sem);
    printf("enter the phno:");
    scanf("%d", &nn->phno);
    nn->next = NULL;
    return (nn);
}

void front_insert()
{
    node *temp;
    temp = (node *)malloc(sizeof(node));
    temp = read_data();
    if (first == NULL)
        first = temp;
    else
    {
        temp->next = first;
        first = temp;
    }
}

void end_insert()
{
    node *temp, *curr;
    temp = (node *)malloc(sizeof(node));
    temp = read_data();
    if (first == NULL)
```

```
        first = temp;
    else
    {
        curr = first;
        while (curr->next != NULL)
            curr = curr->next;
        curr->next = temp;
    }
}

void front_del()
{
    node *temp;
    if (first == NULL)
        printf("list is empty\n");
    else
    {
        temp = first;
        first = first->next;
        free(temp);
    }
}

void end_del()
{
    node *curr, *prev;
    if (first == NULL)
        printf("list is empty\n");
    if (first->next == NULL)
    {
        curr = first;
        first = NULL;
        free(curr);
    }
    else
    {
        curr = first;
        while (curr->next != NULL)
        {
            prev = curr;
            curr = curr->next;
        }
        prev->next = NULL;
        free(curr);
    }
}

void display()
```

```
{
    int count = 0;
    node *temp;
    if (first == NULL)
        printf("list is empty");
    else
    {
        temp = first;
        while (temp != NULL)
        {
            printf("\nusn:%s", temp->usn);
            printf("name:%s", temp->name);
            printf("branch:%s", temp->branch);
            printf("sem:%d", temp->sem);
            printf("phno:%d\n", temp->phno);
            count++;
            temp = temp->next;
        }
        printf("no of nodes:%d", count);
    }
}
```

9 double linked list

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node
{
    char ssn[10];
    char name[20];
    char dept[10];
    char des[10];
    int sal;
    char ph[11];
    struct node *left;
    struct node *right;
} node;

node *first = NULL;
node *readdata();
void frontinsert();
void endinsert();
void frontdel();
```

```c
void enddel();
void display();

int main()
{
    int ch;
    do
    {
        printf("1.insrt frnt\n2.end insert\n3.del frnt\n4.del
end\n5display\n6.exit\n");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            frontinsert();
            break;
        case 2:
            endinsert();
            break;
        case 3:
            frontdel();
            break;
        case 4:
            enddel();
            break;
        case 5:
            display();
            break;
        case 6:
            exit(0);
        }
    } while (ch != 6);
    return 0;
}

node *readdata()
{
    node *nn;
    nn = (node *)malloc(sizeof(node));
    printf("enter ssn of employee");
    scanf("%s", nn->ssn);
    printf("enter name");
    scanf("%s", nn->name);
    printf("enter dept");
    scanf("%s", nn->dept);
    printf("enter des");
    scanf("%s", nn->des);
    printf("enter salary");
```

```
    scanf("%d", &nn->sal);
    printf("enter ph no");
    scanf("%s", nn->ph);
    nn->left = NULL;
    nn->right = NULL;
    return (nn);
}

void frontinsert()
{
    node *temp;
    temp = readdata();
    if (first == NULL)
        first = temp;
    else
    {
        temp->right = first;
        first->left = temp;
        first = temp;
    }
}

void endinsert()
{
    node *temp, *curr;
    temp = readdata();
    if (first == NULL)
        first = temp;
    else
    {
        curr = first;
        while (curr->right != NULL)
            curr = curr->right;
        curr->right = temp;
        temp->left = curr;
    }
}

void frontdel()
{
    node *temp;
    if (first == NULL)
        printf("list is emopty");
    else if (first->right == NULL)
    {
        temp = first;
        first = NULL;
        free(temp);
```

```
    }
    else
    {
        temp = first;
        first = first->right;
        first->left = NULL;
        free(temp);
    }
}

void enddel()
{
    node *temp, *prev, *curr;
    if (first == NULL)
    {
        printf("list is empty");
    }
    else if (first->right == NULL)
    {
        temp = first;
        first = NULL;
        free(temp);
    }
    else
    {
        prev = curr = first;
        while (curr->right != NULL)
        {
            prev = curr;
            curr = curr->right;
        }
        free(curr);
        prev->right = NULL;
    }
}

void display()
{
    node *temp;
    int n = 0;
    if (first == NULL)
    {
        printf("list is empty");
    }
    else
    {
        temp = first;
        while (temp != NULL)
```

```
        {
            printf("ssn is %s", temp->ssn);
            printf("name is%s", temp->name);
            printf("dept is %s", temp->dept);
            printf("des is %s", temp->des);
            printf("salary is %d", temp->sal);
            printf("ph no is %s", temp->ph);
            n++;
            temp = temp->right;
        }
        printf("num of nodes%d", n);
    }
}
```

10 Tree

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

typedef struct NODE

{
    int info;
    struct NODE *lchild;
    struct NODE *rchild;
} node;

node *root = NULL;
node *tree;
void create();
node *insert(int);
void preorder(node *);
void inorder(node *);
void postorder(node *);
int search(node *, int);

int main()

{
    int ch, key, flag = 0;
    do
    {

        printf("\n1.create\n2.preorder\n3.inorder\n4.postorder\n5.search\n6.ex
it\n");
        printf("enter your choice:");
```

```
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            create();
            break;
        case 2:
            preorder(root);
            break;
        case 3:
            inorder(root);
            break;
        case 4:
            postorder(root);
            break;
        case 5:
            printf("enter the search key:");
            scanf("%d", &key);
            flag = search(root, key);
            if (flag == 1)
                printf("element is found");
            else
                printf("not found");
            break;
        case 6:
            exit(0);
        default:
            printf("invalid choice");
        }
    } while (ch != 6);
    return 0;
}

void create()
{
    int i, n, e;
    printf("enter the no of nodes:");
    scanf("%d", &n);
    printf("enter the element:");
    for (i = 0; i < n; i++)
    {
        scanf("%d", &e);
        root = insert(e);
    }
    printf("tree constructed\n");
}

node *insert(int e)
```

```
{
    node *nn, *prev, *temp;
    nn = (node *)malloc(sizeof(node));
    nn->rchild = NULL;
    nn->lchild = NULL;
    nn->info = e;
    if (root == NULL)
    {
        root = nn;
        return (root);
    }
    else
    {
        temp = root;
        while (temp != NULL)
        {
            prev = temp;
            if (temp->info > nn->info)
                temp = temp->lchild;
            else if (temp->info < nn->info)
                temp = temp->rchild;
            else
            {
               printf("duplicate");
                return (root);
            }
        }
        if (prev->info < nn->info)

            prev->rchild = nn;

        else

            prev->lchild = nn;

        return (root);
    }
}

void preorder(node *tree)
{
    if (tree == NULL)
        return ;
    else
    {
        printf("%d", tree->info);
        preorder(tree->lchild);
        preorder(tree->rchild);
```

```
    }
}

void postorder(node *tree)
{
    if (tree == NULL)
        return ;
    else
    {
        postorder(tree->lchild);
        postorder(tree->rchild);
        printf("%d", tree->info);
    }
}

void inorder(node *tree)
{
    if (tree == NULL)
        return ;
    else
    {
        inorder(tree->lchild);
        printf("%d", tree->info);
        inorder(tree->rchild);
    }
}

int search(node *root, int e)
{
    if (root == NULL)
    {
        return 0;
    }
    else if (root->info == e)
        return 1;
    else if (root->info > e)
        search(root->lchild, e);
    else
        search(root->rchild, e);
}
```

11 Graph

```c
#include <stdio.h>
int a[10][10], n, v[10], source;
void input();
void dfs(int);
void output();
int main()
{
    input();
    dfs(source);
    output();
    return 0;
}

void input()
{
    int i, j;
    printf("enter the no of nodes:");
    scanf("%d", &n);
    printf("enter the adjacency matrix:");
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            scanf("%d", &a[i][j]);
    printf("enter the source vertex:");
    scanf("%d", &source);
    for (i = 1; i <= n; i++)
        v[i] = 0;
}

void dfs(int s)
{
    int k;
    v[s] = 1;
    for (k = 1; k <= n; k++)
    {
        if (a[s][k] == 1 && v[k] == 0)
        {
            printf("%d->%d", s, k);
            dfs(k);
        }
    }
}

void output()
{
    int i;
    for (i = 1; i <= n; i++)
    {
```

```
        if (v[i] == 0)
            printf("%d not reacable\n", i);
        else
            printf("%d reacable\n", i);
    }
}
```

12

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

#define MAX 100

int a[MAX], num, key, i;
int create(int);
void linear_prob(int[], int, int);
void display(int[]);
int main()
{
    int j;
    printf("\ncollision handling by linear probing");
    for (i = 10; i < MAX; i++)
        a[i] = -1;
    do
    {
        printf("enter four digit number:");
        scanf("%d", &num);
        key = create(num);
        linear_prob(a, key, num);
        printf("\ndo you want to continue ? (0/1)");
        scanf("%d", &j);
    } while (j == 1);
    display(a);
    return 0;
}

int create(int num)
{

    key = num % 100;
```

```
    return key;
}
void linear_prob(int a[MAX], int key, int num)
{
    int flag, count;
    flag = count;1
        if (a[key] == -1)
            a[key] = num;
    else

    {
        i = 10;
        while (i < MAX - 10)

        {
            if (a[i] != -1)
                count++;
            i++;
        }
        if (count == MAX - 10)
        {
            printf("\nhash table is full");
            display(a);
            exit(1);
        }
        for (i = key + 1; i < MAX; i++)
        {
            if (a[i] == -1)
            {
                a[i] = num;

                flag = 1;

                break;
            }
        }
        for (i = 10; i < key && flag == 0; i++)
        {
            if (a[i] == -1)

            {
                a[i] = num;

                flag = 1;

                break;
            }
```

```
        }
    }
}

void display(int a[MAX])
{
    printf("\nthe hash table is...\n");
    for (i = 10; i < MAX; i++)
        printf("\n %d %d", i, a[i]);
}
```