--- HOME ---
1. employee structure

```
#include<stdio.h>

int i,n;

struct Employee{

int Employ_num;

int Basic_sal;

char Employ_name[20];

float All_allowances,IT,Net_sal,gross_sal;

}E[5];

void read_data(){

printf("enter the value of n\n");

scanf("%d",&n);

printf("Enter the Employee_num\t Basic_sal\t Employee_name\n");

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

scanf("%d%d%s",&E[i].Employ_num,&E[i].Basic_sal,E[i].Employ_name);}}

void claculate_data(){

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

E[i].All_allowances=1.23*E[i].Basic_sal;

E[i].gross_sal=E[i].All_allowances+E[i].Basic_sal;

E[i].IT=0.3*E[i].gross_sal;

E[i].Net_sal=E[i].gross_sal-E[i].IT;}}

void display_data(){

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

printf("\nThe All_allowances of Employee is:%f\n",E[i].All_allowances);

printf("The Gross salary of Employee is:%f\n",E[i].gross_sal);

printf("The income tax of Employee is:%f\n",E[i].IT);

printf("The Net Salary of Employee is:%f\n",E[i].Net_sal);}}

int main(){
```

```
read_data();
claculate_data();
display_data();

return 0;}
```

2.Student roll number

```
#include<stdio.h>
#include<stdlib.h>
int a[10],n,i,key,pos;
void create(){
printf("Enter the value of n\n");
scanf("%d",&n);
printf("Enter the %d elements\n",n);
for (i=0;i<n;i++){
scanf("%d",&a[i]);}}
void insert(){
printf("Enter the key element and position to be insert\n");
scanf("%d%d",&key,&pos);
for(i=n;i>=pos;i--){
a[i+1]=a[i];}
a[pos]=key;
n++;}
void display(){
printf("The elements are:\n");
for(i=0;i<n;i++){
printf("%d\t",a[i]);}}
void dele(){
```

```
printf("Enter the position to be deleted\n");
scanf("%d",&pos);
key=a[pos];
for(i=pos;i<n-1;i++){
a[i]=a[i+1];}
n--;
printf("The deleted element is %d:\n",key);}
int main(){
int ch;
while(1){
printf("\nEnter the choice\n 1.create\n 2.insert\n 3.delete\n 4.display\n ");
scanf("%d",&ch);
switch(ch){
case 1:create();
break;
case 2:insert();
break;
case 3:dele();
break;
case 4:display();
break;

default:printf("Invalid Key\n");}}}
```

3.push and pop

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define max 4
```

```
int a[max],top=-1;
void push(int m){
a[++top]=m;}
int pop()
{return a[top--];
}void display()
{int i;
if(top==-1){
printf("stack is empty");}
printf("Enter the elements");
for(i=top;i>=0;i--){
printf("%d\t",a[i]);}}
void palin()
{int i, n,num,rem;
top=-1;
printf("Enter n");
scanf("%d",&n);
num=n;
while(n!=0){
rem=n%10;
push(rem);
n=n/10;}
for(i=0;top!=-1;i++)
{n=pop()*pow(10,i)+n;}
if(num==n){
printf("\n is a palindrome");}
else{
printf("\nis not a plaindrome\n");}}
```

```
int main()
{int c,m;
while(1){
printf("\n 1.push\t 2.pop\t 3.palindrome\t 4.display\t 5.exit\n");
printf("Enter the choice\n");
scanf("%d",&c);
switch(c){
case 1:if(top==max-1){
printf("Stack is overflow\n");}
else{
printf("Enter the elements\n");
scanf("%d",&m);
push(m);}
break;
case 2:if(top==-1){
printf("Stack is underflow\n");}
else{
printf("The popped elements are %d\n",pop());}
break;
case 3:palin();
break;
case 4:display();
break;
case 5:exit(0);
break;
default:printf("INVALID KEY");}}}
```

4.a infix to postfix

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

```
#include<string.h>
char stack[20];
int top=-1;
void push(char s){
stack[++top]=s;}
char pop(){
return stack[top--];}
int precd(char s){
switch(s){
case '^':return 4;
case '*':case '/':return 3;
case '+':case '-':return 2;
case '(':case ')':case '#':return 1;}}
void convertip(char infix[],char postfix[]){
int i,j=0;
char symb;
push('#');
for(i=0;i<strlen(infix);i++){
symb=infix[i];
switch(symb){
case '(':push(symb);break;
case ')':while(stack[top]!='(')
postfix[j++]=pop();
pop();
break;
case '*':case '/':case '+':case '-':case '^':while(precd(symb)<=precd(stack[top]))
postfix[j++]=pop();
push(symb);
```

```
break;
default:postfix[j++]=symb;
break;}}
while(stack[top]!='#')
postfix[j++]=pop();}
int main(){
char infix[20],postfix[20];
printf("Enter the infix expression\n");
gets(infix);
convertip(infix,postfix);
puts(postfix);

return 0;}
```

4b. postfix(suffix)

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
float stack[20];
int top=-1;
void push(char symb){
stack[++top]=symb;}
float pop(){
return stack[top--];}
float eval_postfix(char postfix[]){
int i;
float op1,op2;
char symb;
for(i=0;i<strlen(postfix);i++){
```

```
symb=postfix[i];
if(isdigit(symb)){
push(symb-'0');}
else{
op2=pop();op1=pop();
switch(symb){
case '+':push(op1+op2);break;
case '-':push(op1-op2);break;
case '*':push(op1*op2);break;
case '/':push(op1/op2);break;}}}
return pop();}
int main(){
float val;
char postfix[20];
printf("Enter valid postfix expression\n");
gets(postfix);
val=eval_postfix(postfix);
printf("\n Result of postfix expression %s=%f",postfix,val);

return 0;}
```

5. circular queue in rainbow colors

```
#include<stdio.h>
#include<stdlib.h>
#define size 7
char cq[size];
int r=-1,f=0,cnt=0;
void cq_insert(){
```

```
if(cnt==size){
printf("Cq is full\n");
return;}
printf("Enter the Rainbow color\n");
r=(r+1)%size;
scanf("%s",&cq[r]);
cnt++;}
void del(){
if(cnt==0){
printf("Cq is empty\n");
return;}
printf("The deleted Rainbow color is %c",cq[f]);
f=(f+1)%size;
cnt--;}
void cq_display(){
int i,k=f;
if(cnt==0){
printf("Cq is empty\n");
return;}
printf("\n Cq Rainbow color are\n");
for(i=0;i<cnt;i++){
printf("%c",cq[k]);
k=(k+1)%size;}}
int main(){
int ch;
while(1){
printf("\n1.Cq_insert\n 2.cq_delete\n 3.cq_display\n 4.exit\n");
printf("Enter the choice\n");
scanf("%d",&ch);
```

```
switch(ch){
case 1:cq_insert();break;
case 2:del();break;
case 3:cq_display();break;
case 4:exit(0);break;
default:printf("Invalid input");}}

return 0;}
```

6, SLL

```
#include<stdio.h>
#include<stdlib.h>
struct SLL
{
char usn[10];
struct SLL *next;
};
typedef struct SLL node;
node *start=NULL;
node *getnode()
{
node *p;
p=(node*)malloc(sizeof(node));
printf("\n Enter USN");
scanf("%s",p->usn);
fflush(stdin);
p->next=NULL;
return p;
```

```
}
void insert_front()
{
node *n1;
n1=getnode();
n1->next=start;
start=n1;
}
void delete_front()
{
node *temp;
if(start==NULL)
{
printf("\n Empty list");
return;
}
temp=start;
printf("%s info is deleted\n",temp->usn);
start=temp->next;
free(temp);
}
void insert_end()
{
node *n1,*temp;
n1=getnode();
if(start==NULL)
{
start=n1;
return;
```

```
}
temp=start;
while(temp-> next!=NULL)
temp=temp->next;
temp-> next=n1;
}
void delete_end()
{
node *temp, *prev;
if(start==NULL)
{
printf("\n Empty list");
return;
}
if(start-> next ==NULL)
{
printf("\n%s student details is deleted",start->usn);
free(start);
start=NULL; return;
}
temp=start;
while(temp->next!=NULL)
{
prev=temp;
temp=temp->next;
}
prev->next=NULL;
printf("\nThe deleted node is \t%s\n",temp->usn);
```

```
free(temp);
}
void display()
{
int cnt=0;
node *temp;
if(start==NULL)
{
printf("\n Empty list");
return;
}
temp=start;
printf("\n The details are\n");
while(temp != NULL)
{
printf("\n%s\t", temp->usn);
cnt++;
temp=temp->next;
}
printf("\n Number of nodes is %d",cnt);
}
int main()
{
int n, m, i;
while(1)
{
printf("\n Enter 1:insert_front\n 2:insert_end\n 3:delete_front\n 4:delete_end\n 5:display\n");
scanf("%d",&m);
switch(m)
```

```
{
case 1:printf("\n Enter n");
scanf("%d",&n);
for(i=0;i<n;i++)
insert_front();
break;
case 2: insert_end();
break;
case 3: delete_front();
break;
case 4: delete_end();
break;
case 5: display();
break;
default:exit(0);
}
}
return 0;

}
```

7. DLL

```
#include<stdio.h>
#include<stdlib.h>
struct dll
{
int usn;
struct dll *lptr, *rptr;
```

```
};
typedef struct dll node;
node *start = NULL;
node *getnode()
{
node *p;
p = (node*)malloc(sizeof(node));
printf("Enter the usn\n");
scanf("%d", &p->usn);
p->lptr = p->rptr = NULL;
return p;
}
void insert_front()
{
node *new1 = getnode();
if (start == NULL)
{
start = new1;
}
else
{
new1->rptr = start;
start->lptr = new1;
start = new1;
}}
void delete_front()
{
node *temp;
if (start == NULL)
```

```
{
printf("List is Empty\n");
return;
}
temp = start;
start = start->rptr;
if (start != NULL)
{
start->lptr = NULL;
}
printf("The deleted information is: %d\n", temp->usn);
free(temp);
}
void insert_end()
{
node *new1 = getnode();
node *temp = start;
if (start == NULL)
{
start = new1;
}
else
{
while (temp->rptr != NULL)
{
temp = temp->rptr;
}
temp->rptr = new1;
```

```
new1->lptr = temp;
}
}
void delete_end()
{
node *temp;
if (start == NULL)
{
printf("The list is Empty\n");
return;
}
temp = start;
while (temp->rptr != NULL)
{
temp = temp->rptr;
}
if (temp->lptr == NULL)
{
start = NULL;
}
else
{
temp->lptr->rptr = NULL;
}
printf("The deleted information is: %d\n", temp->usn);
free(temp);
}
void display()
{
```

```
node *temp = start;
int cnt = 0;
if (start == NULL)
{
printf("The list is empty\n");
return;
}
while (temp != NULL)
{
printf("%d ", temp->usn);
cnt++;
temp = temp->rptr;
}
printf("\nThe number of nodes in the list is: %d\n", cnt);
}
int main()
{
int ch;
while (1)
{
printf("\n1.insert_front\n2.delete_front\n3.insert_end\n4.delete_end\n5.display\n");
printf("Enter the choice: ");
scanf("%d", &ch);
switch (ch)
{
case 1: insert_front(); break;
case 2: delete_front(); break;
case 3: insert_end(); break;
```

```
case 4: delete_end(); break;
case 5: display(); break;
default: exit(0);
}
}
return 0;

}
```

8. SLL AND DLL OF SORT,REVERSE,CONCATENATE

```
#include<stdio.h>
#include<stdlib.h>
struct sll {
int usn;
struct sll *rptr;};
typedef struct sll node;
node *start = NULL, *s1 = NULL, *s2 = NULL;
node *getnode()
{node *p;
p = (node*)malloc(sizeof(node));
p->rptr = NULL;
return p;}
void insert()
{node *new1 = getnode();
printf("Enter USN: ");
scanf("%d", &new1->usn);
new1->rptr = start;
start = new1;}
void display(){
```

```
node *temp;
if (start == NULL){
printf("The list is empty\n");
return;}
temp = start;
while (temp != NULL){
printf("%d ", temp->usn);
temp = temp->rptr;}
printf("\n");
}void reverse()
{node *temp = start, *prev = NULL, *next = NULL;
while (temp != NULL){
next = temp->rptr;
temp->rptr = prev;
prev = temp;
temp = next;}
start = prev;}
void concatenate(node *s1, node *s2){
if (s1 == NULL) {
start = s2;
return;}
node *temp = s1;
while (temp->rptr != NULL){
temp = temp->rptr;}
temp->rptr = s2;
start = s1;}
void sorting(){
node *current = start, *index = NULL;
```

```
int temp;
while (current != NULL){
index = current->rptr;
while (index != NULL){
if (current->usn > index->usn)
{temp = current->usn;
current->usn = index->usn;
index->usn = temp;}
index = index->rptr;}
current = current->rptr;}
display();}
int main()
{int choice, n, i;
node *n1, *n2, *temp;
while (1)
{
printf("\n1. Insert\n2. Sort\n3. Concatenate\n4. Reverse\n5. Display\n");
printf("Enter choice: ");
scanf("%d", &choice);
switch (choice)
{case 1:
printf("Enter the number of students: ");
scanf("%d", &n);
for (i = 0; i < n; i++)
{n1 = getnode();
scanf("%d", &n1->usn);
n1->rptr = start;
start = n1;}
break;
```

```
case 2:
sorting();
break;
case 3:
printf("Enter the number of students in the 1st list: ");
scanf("%d", &n);
s1 = NULL;
printf("Enter %d elements\n", n);
for (i = 0; i < n; i++)
{
n1 = getnode();
scanf("%d", &n1->usn);
n1->rptr = s1;
s1 = n1;
}
printf("Enter the number of students in the 2nd list: ");
scanf("%d", &n);
s2 = NULL;
printf("Enter %d elements\n", n);
for (i = 0; i < n; i++)
{
n2 = getnode();
scanf("%d", &n2->usn);
n2->rptr = s2;
s2 = n2;
}
concatenate(s1, s2);
temp=start;
```

```
while(temp!=NULL)
{
printf("%d",temp->usn);
temp=temp->rptr;
}
break;
case 4:
reverse();
display();
break;
case 5:
display();
break;
default:
exit(0);
}
}

}
```

9.harshing

```
#include <stdio.h>
#include<stdlib.h>
int L[100], max=100;
void display()
{
int i;
printf("Hash Table Contents Are\n");
printf("Index Value\n");
```

```
for(i=0;i<max;i++)
{
printf("%d %d\n",i,L[i]);
}
}
void Linear_probing(int key, int num)
{
int i;
if(L[key]==-1)
{
L[key]=num;
}
else
{
printf("Collision detected at index %d\n", key);
i=(key+1)%max;
while(i!=key)
{
if(L[i]==-1)
{
L[i]=num;
printf("Collision resolved through linear probing\n");
return;
}
i=(i+1)%max;
}
printf("Hash Table is Full\n");
}
```

```
}
int main()
{
int input, num, key;
for(int i=0;i<max;i++)
{
L[i]=-1;
}
do
{
printf("Enter a number: ");
scanf("%d",&num);
key=num%max;
Linear_probing(key,num);
printf("Enter 1 to continue: ");
scanf("%d",&input);
} while(input==1);
display();
return 0;
}
```

10.binary search tree

```
#include <stdio.h>
#include <stdlib.h>
struct bst {
int data;
struct bst *lptr, *rptr;
};
typedef struct bst node;
```

```
node* insert(node* root, int key) {
if (root == NULL) {
root = (node*)malloc(sizeof(node));
root->lptr = root->rptr = NULL;
root->data = key;
return root;
}
if (key < root->data)
root->lptr = insert(root->lptr, key);
else if (key > root->data)
root->rptr = insert(root->rptr, key);
else
printf("Duplicate Element!! Not Allowed!!\n");
return root;
}
void inorder(node* root) {
if (root != NULL) {
inorder(root->lptr);
printf("%d\t", root->data);
inorder(root->rptr);
}
}
void preorder(node* root) {
if (root != NULL) {
printf("%d\t", root->data);
preorder(root->lptr);
preorder(root->rptr);
}
```

```
}
void postorder(node* root) {
if (root != NULL) {
postorder(root->lptr);
postorder(root->rptr);
printf("%d\t", root->data);
}
}
int search(node* root, int key) {
if (root == NULL)
return 0;
if (key == root->data)
return 1;
if (key < root->data)
return search(root->lptr, key);
else
return search(root->rptr, key);
}
int main() {
node* root = NULL;
int key, ch, n, i;
while (1) {
printf("\n1. Insert\n2. Search\n3. Preorder\n4. Postorder\n5. Inorder\n6. Exit\n");
printf("Enter your choice: ");
scanf("%d", &ch);
switch (ch) {
case 1:
printf("Enter n value: ");
scanf("%d", &n);
```

```
printf("Enter %d values:\n", n);
for (i = 0; i < n; i++) {
scanf("%d", &key);
root = insert(root, key);
}
break;
case 2:
printf("Enter the value to search: ");
scanf("%d", &key);
if (search(root, key) == 1)
printf("%d is present in the tree.\n", key);
else
printf("%d is not present in the tree.\n", key);
break;
case 3:
printf("Preorder traversal: ");
preorder(root);
printf("\n");
break;
case 4:
printf("Postorder traversal: ");
postorder(root);
printf("\n");
break;
case 5:
printf("Inorder traversal: ");
inorder(root);
printf("\n");
```

```
break;

case 6:

exit(0);

default:

printf("Invalid choice.\n");

break;

}

}

return 0;


}
```