1)CALENDAR

```
#include <stdio.h>
#include <stdlib.h>
struct Day {
char *dayName;
int date;
char *activity;
};
void create(struct Day *day) {
day->dayName = (char *)malloc(sizeof(char) * 20);
day->activity = (char *)malloc(sizeof(char) * 100);
printf("Enter the day name:");
scanf("%s", day->dayName);
printf("Enter the date:");
scanf("%d", &day->date);
printf("Enter the activity for the day:");
scanf(" %[^\n]s", day->activity);
}
void read(struct Day *calendar, int size) {
for (int i = 0; i < size; i++) {
printf("Enter details for Day %d:\n", i + 1);
create(&calendar[i]);
}}
void display(struct Day *calendar, int size) {
printf("\nWeek's Activity Details:\n");
for (int i = 0; i < size; i++) {
printf("Day %d:\n", i + 1);
printf("Day Name: %s\n", calendar[i].dayName);
printf("Date: %d\n", calendar[i].date);
printf("Activity: %s\n", calendar[i].activity);
printf("\n");
}}
void freeMemory(struct Day *calendar, int size) {
for (int i = 0; i < size; i++) {
free(calendar[i].dayName);
free(calendar[i].activity);
}}
int main() {
int size;
printf("Enter the number of days in the week:");
scanf("%d", &size);

struct Day *calendar = (struct Day
*)malloc(sizeof(struct Day) * size);

if (calendar == NULL) {
printf("Memory allocation failed. Exiting
program.\n");
return 1;
}
read(calendar, size);
display(calendar, size);
freeMemory(calendar, size);
free(calendar);
return 0;
}
```

2)STRING

```
#include <stdio.h>
void main() {
char STR[100], PAT[100], REP[100], ans[100];
int i, j, c, m, k, flag = 0;
printf("\nEnter the MAIN string: \n");
gets(STR);
printf("\nEnter a PATTERN string: \n");
gets(PAT);
printf("\nEnter a REPLACE string: \n");
gets(REP);
i = m = c = j = 0;
while (STR[c] != '\0') {
if (STR[m] == PAT[i]) {
i++;
m++;
if (PAT[i] == '\0') {
for (k = 0; REP[k] != '\0'; k++, j++) {
ans[j] = REP[k];
flag = 1;
}
i = 0;
c = m;}
} else {
ans[j] = STR[c];
j++;
c++;
m = c;
i = 0;
}}
if (flag == 0) {
printf("Pattern doesn't found!!!");
} else {
ans[j] = '\0';
printf("\nThe RESULTANT string is:%s\n", ans);
}}
```

```
void pali()
{
int digit,j,k,len=top+1,flag=0,ind=0 ,length = 0;
int num[len],rev[len],i=0;
while(top!=-1)
 {
 digit= stack[top];
 num[i]=digit;
 top--;
 i++;
 }
```

3)STACK OPERATIONS

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#define max_size 5
int stack[max_size],top = -1;
void push();
void pop();
void display();
void pali();
int main()
{
int choice;
while(choice){
printf("\n\n--------STACK OPERATIONS----------\n");
 printf("1.Push\n");
 printf("2.Pop\n");
 printf("3.Palindrome\n");
 printf("4.Display\n");
 printf("5.Exit\n");
 printf("----------------------");
 printf("\nEnter your choice:\t");
 scanf("%d",&choice);
switch(choice){
 case 1:
 push();break;
 case 2:
 pop();break;
 case 3:
 pali(); break;
 case 4:
 display();break;
 case 5:
 exit(0);break;
 default:
 printf("\nInvalid choice:\n");break;
 }}
return 0;
}
void push()
{
int item,n;
if(top==(max_size-1))
 {
 printf("\nStack Overflow:");
 }
else
 {
 printf("Enter the element to be inserted:\t");
 scanf("%d",&item);
 top=top+1;
 stack[top]=item;
}}
void pop()
{
int item;
if(top==-1)
 {
 printf("Stack Underflow:");
 }
else
 {
 item=stack[top];
 top=top-1;
 printf("\nThe poped element: %d\t",item);
 } }
for(j=0;j<len;j++){
 printf("Numbers= %d\n",num[j]);
 }
 printf("reverse operation : \n");
for(k=len-1;k>=0;k--){
 rev[k]=num[ind];
 ind++;
 }
 printf("reverse array : ");
for(k=0;k<len;k++)
 {
 printf("%d\n",rev[k]);
 }
 printf("check for palindrome :\n");

 for(i=0;i<len;i++)
 {
 if(num[i]==rev[i])
 {
 length = length+1;
 }
 }
if(length==len)
 {
 printf("It is palindrome number\n");
 }
else
 {
 printf("It is not a palindrome number\n");
 }
 top = len-1;
}
void display()
{
int i;
if(top==-1)
 {
 printf("\nStack is Empty:");
 }
else
 {
 printf("\nThe stack elements are:\n" );
 for(i=top;i>=0;i--)
 {
 printf("%d\n",stack[i]);
 }}}
```

4)INFIX TO POSTFIX

```
#define SIZE 50 /* Size of Stack */
#include <ctype.h>
#include <stdio.h>
char s[SIZE];
int top = -1; /* Global declarations */
push(char elem) /* Function for PUSH operation */
{
 s[++top] = elem;
}
char pop() /* Function for POP operation */
{
return (s[top--]);
}
int pr(char elem) /* Function for precedence */
{
switch (elem){
 case '#':
 return 0;
 case '(':
 return 1;
 case '+':
 case '-':
 return 2;
 case '*':
 case '/':
 case '%':
 return 3;
 case '^':
 return 4;
 }}
void main() /* Main Program */
{
char infx[50], pofx[50], ch, elem;
int i = 0, k = 0;
 printf("\n\nEnter the Infix Expression ");
 scanf("%s", infx);
 push('#');
while ((ch = infx[i++]) != '\0')
 {
 if (ch == '(')
 push(ch);
 else if (isalnum(ch))
 pofx[k++] = ch;
 else if (ch == ')'){
 while (s[top] != '(')
 pofx[k++] = pop();
 elem = pop(); /* Remove ( */
 }
 else {
 while (pr(s[top]) >= pr(ch))
 pofx[k++] = pop();
 push(ch);
 }}
while (s[top] != '#') /* Pop from stack till empty */
 pofx[k++] = pop();
 pofx[k] = '\0'; /* Make pofx as valid string */
 printf("\n\nGiven Infix Expn: %s Postfix Expn:
%s\n", infx, pofx);
}
```

5)SUFIX

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<math.h>
#define MAX 50
int stack[MAX];
char post[MAX];
int top= -1;
/*fUNCTION PROTOYPE */
void pushstack(int tmp);
void calculator(char c);
void main()
{
int i;
 printf("Insert a postfix notation :: ");
 scanf("%s",post);
for(i=0;i<strlen(post);i++){
 if(post[i]>='0' && post[i]<='9'){
 pushstack(i);
 }
 if(post[i]=='+' || post[i]=='-' || post[i]=='*' ||
post[i]=='/' || post[i]=='^'){
 calculator(post[i]);
 }}
 printf("\n\nResult :: %d",stack[top]);
}
void pushstack(int tmp)
{
 top++;
 stack[top]=(int)(post[tmp]-48);
}
void calculator(char c)
{
int a,b,ans;
 a=stack[top];
 stack[top]='\0';
top--;
 b=stack[top];
 stack[top]='\0';
 top--;
switch(c)
 {
 case '+':
 ans=b+a;break;
 case '-':
 ans=b-a;break;
 case '*':
 ans=b*a;break;
 case '/':
 ans=b/a;break;
 case '^':
 ans=pow(b,a);break;
 default:
 ans=0;
 }
 top++;
 stack[top]=ans;
}
```

5)B)//4B Towers of Hanoi

```
#include <stdio.h>
void towers(int, char, char, char);
int main(){
int num;
 printf("Enter the number of disks : ");
 scanf("%d", &num);
 printf("The sequence of moves involved in the
Tower of Hanoi are :\n");
 towers(num, 'A', 'C', 'B');
return 0;
}
void towers(int num, char frompeg, char topeg,
char auxpeg){
if (num == 1) {
printf("\n Move disk 1 from peg %c to peg %c",
frompeg, topeg);
 return;
 }
 towers(num - 1, frompeg, auxpeg, topeg);
 printf("\n Move disk %d from peg %c to peg %c",
num, frompeg, topeg);
 towers(num - 1, auxpeg, topeg, frompeg);
}
```

6) Circular QUEUE

```
#include<stdio.h>
#include<stdlib.h>
#define max 10
int q[10], front = 0, rear = -1;
void insert() {
int x;
if ((front == 0 && rear == max - 1) || (front > 0 &&
rear == front - 1))
printf("Queue is overflow\n");
else {
printf("Enter element to be insert:");
scanf("%d", &x);
if (rear == max - 1 && front > 0) {
rear = 0;
q[rear] = x;
} else {
if ((front == 0 && rear == -1) || (rear != front - 1))
q[++rear] = x;
}}}
void delet() {
int a;
if ((front == 0) && (rear == -1)) {
printf("Queue is underflow\n");
exit(1);
}
if (front == rear) {
a = q[front];
rear = -1;
front = 0;
} else if (front == max - 1) {
a = q[front];
front = 0;
} else
a = q[front++];
printf("Deleted element is:%d\n", a);}
void display() {
int i, j;
if (front == 0 && rear == -1) {
printf("Queue is underflow\n");
exit(1);
}
if (front > rear) {
for (i = 0; i <= rear; i++)
printf("\t%d", q[i]);
for (j = front; j <= max - 1; j++)
printf("\t%d", q[j]);
printf("\nrear is at %d\n", q[rear]);
printf("\nfront is at %d\n", q[front]);
} else {
for (i = front; i <= rear; i++) {
printf("\t%d", q[i]);
}
printf("\nrear is at %d\n", q[rear]);
printf("\nfront is at %d\n", q[front]);
}
printf("\n");
}
int main() {
int ch;
printf("\nCircular Queue operations\n");
printf("1.insert\n2.delete\n3.display\n4.exit\n");
while (1) {
printf("Enter your choice:");
scanf("%d", &ch);
switch (ch) {
case 1:
insert();
break;
case 2:
delet();
break;
case 3:
display();
break;
case 4:
exit(1);
default:
printf("Invalid option\n");
}}
return 0;
}
```

7) Singly Linked List

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int count=0;
struct node {
int sem, phno;
char name[20], branch[10], usn[20];
struct node *next;
} *first=NULL, *last=NULL, *temp=NULL, *temp1;
void create() {
int sem, phno;
char name[20], branch[10], usn[20];
temp=(struct node*)malloc(sizeof(struct node));
printf("\nEnter usn,name, branch, sem, phno of
student: ");
scanf("%s %s %s %d %d", usn, name, branch,
&sem, &phno);
strcpy(temp->usn, usn);
strcpy(temp->name, name);
strcpy(temp->branch, branch);
temp->sem = sem;
temp->phno = phno;
temp->next = NULL;
count++;}
void insert_atfirst() {
if (first == NULL) {
create();
first = temp;
last = first;
} else {
create();
temp->next = first;
first = temp;}}
void insert_atlast() {
if (first == NULL) {
create();
first = temp;
last = first;
} else {
create();
last->next = temp;
last = temp;}}
void display() {
temp1 = first;
if (temp1 == NULL) {
printf("List empty to display\n");
return;}
printf("\nLinked list elements from begining:\n");
while (temp1 != NULL) {
printf("%s %s %s %d %d\n", temp1->usn, temp1-
>name, temp1->branch, temp1->sem, temp1-
>phno);
temp1 = temp1->next;}
printf("No of students = %d\n", count);}
int deleteend() {
struct node *temp;
temp = first;
if (temp->next == NULL) {
free(temp);
first = NULL;
} else {
while (temp->next != last)
temp = temp->next;
printf("%s %s %s %d %d\n", last->usn, last->name,
last->branch, last->sem, last->phno);
free(last);
temp->next = NULL;
last = temp;   }
count--;
return 0;}
int deletefront() {
struct node *temp;
temp = first;
if (temp->next == NULL) {
free(temp);
first = NULL;
return 0;
} else {
first = temp->next;
printf("%s %s %s %d %d\n", temp->usn, temp-
>name, temp->branch, temp->sem, temp->phno);
free(temp);}
count--;
return 0;}
void main() {
int ch, n, i;
first = NULL;
temp = temp1 = NULL;
printf("----------------MENU----------------------\n");
printf("\n1- Create a SLL of n emp");
printf("\n2 - Display from beginning");
printf("\n3 - Insert at end");
printf("\n4 - Delete at end");
printf("\n5 - Insert at beg");
printf("\n6 - Delete at beg");
printf("\n7 - Exit\n");
printf("-----------------------------------------\n");
while (1) {
printf("\nEnter choice: ");
scanf("%d", &ch);
switch (ch) {case 1:
printf("\nEnter no of students: ");
scanf("%d", &n);
for(i = 0; i < n; i++)
insert_atfirst(); break;
case 2:
display();break;
case 3:
insert_atlast();break;
case 4:
deleteend();break;
case 5:
insert_atfirst();break;
case 6:
deletefront();break;
case 7:
exit(0);
default:
printf("Wrong choice\n");}}}
```

8) Doubly Linked List

```
#include<string.h>
int count=0;
struct node {
struct node *prev;
int ssn, phno;
float sal;
char name[20], dept[10], desg[20];
struct node *next;
}*h, *temp, *temp1, *temp2, *temp4;
void create() {
int ssn, phno;
float sal;
char name[20], dept[10], desg[20];
temp = (struct node *)malloc(sizeof(struct node));
temp->prev = NULL;
temp->next = NULL;
printf("\nEnter ssn, name, department,
designation, salary and phno of employee: ");
scanf("%d %s %s %s %f %d", &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 insertbeg() {
if (h == NULL) {
create();
h = temp;
temp1 = h;
} else {
create();
temp->next = h;
h->prev = temp;
h = temp;}}
void insertend() {
if (h == NULL) {
create();
h = temp;
temp1 = h;
} else {
create();
temp1->next = temp;
temp->prev = temp1;
temp1 = temp;}}
void displaybeg() {
temp2 = h;
if (temp2 == NULL) {
printf("List empty to display\n");
return;}
printf("\nLinked list elements from begining:\n");
while (temp2 != NULL) {
printf("%d %s %s %s %f %d\n", temp2->ssn, temp2-
>name, temp2->dept, temp2->desg, temp2->sal,
temp2->phno);
temp2 = temp2->next;}
printf("No of employees = %d\n", count);}
int deleteend() {
struct node *temp; temp = h;
if (temp->next == NULL) {
free(temp); h = NULL;
return 0;
} else {
temp2 = temp1->prev;
temp2->next = NULL;
printf("%d %s %s %s %f %d\n", temp1->ssn, temp1-
>name, temp1->dept, temp1->desg, temp1->sal,
temp1->phno);
free(temp1);}
count--;
return 0;}
int deletebeg() {
struct node *temp;
temp = h;
if (temp->next == NULL) {
free(temp);
h = NULL;
} else {
h = h->next;
printf("%d %s %s %s %f %d\n", temp->ssn, temp-
>name, temp->dept, temp->desg, temp->sal, temp-
>phno);
free(temp);}count--;
return 0;}
void main() {
int ch, n, i; h = NULL;
temp = temp1 = NULL;
printf("----------------MENU----------------\n");
printf("\n1 - Create a DLL of n emp");
printf("\n2 - Display from beginning");
printf("\n3 - Insert at end");
printf("\n4 - Delete at end");
printf("\n5 - Insert at beg");
printf("\n6 - Delete at beg");
printf("\n7 - Exit\n");
printf("----------------------------------------\n");
while (1) {
printf("\nEnter choice: ");
scanf("%d", &ch);
switch (ch) { case 1:
printf("\nEnter no of employees: ");
scanf("%d", &n);
for(i = 0; i < n; i++)
insertend();break;
case 2:
displaybeg();break;
case 3:
insertend();break;
case 4:
deleteend();break;
case 5:
insertbeg();break;
case 6:
deletebeg();break;
case 7:exit(0);
default:
printf("Wrong choice\n");}}}
```

10) Binary Search Tree

```
#include <stdio.h>
#include <stdlib.h>
int flag = 0;
typedef struct BST {
int data;
struct BST *lchild, *rchild;
} node;
void insert(node *, node *);
void inorder(node *);
void preorder(node *);
void postorder(node *);
node *search(node *, int, node **);
void main() {
int choice;
int ans = 1;
int key;
node *new_node, *root, *tmp, *parent;
node *get_node();
root = NULL;
do {
printf("\n1.Create");
printf("\n2.Search");
printf("\n3.Recursive Traversals");
printf("\n4.Exit");
printf("\nEnter your choice :");
scanf("%d", &choice);
switch (choice) {
case 1:
do {
new_node = get_node();
printf("\nEnter The Element ");
scanf("%d", &new_node->data);
if (root == NULL)
root = new_node;
else
insert(root, new_node);
printf("\nWant To enter More Elements?(1/0)");
scanf("%d", &ans);
} while (ans);
break;
case 2:
printf("\nEnter Element to be searched :");
scanf("%d", &key);
tmp = search(root, key, &parent);
if (flag == 1) {
printf("\nParent of node %d is %d", tmp->data,
parent->data);
} else {
printf("\n The %d Element is not Present", key);
}
flag = 0;
break;
case 3:
if (root == NULL)
printf("Tree Is Not Created");
else {
printf("\nThe Inorder display :");
inorder(root);
printf("\nThe Preorder display : ");
preorder(root);
printf("\nThe Postorder display : ");
postorder(root);}
break;}
} while (choice != 4);}
node *get_node() {
node *temp;
temp = (node *)malloc(sizeof(node));
temp->lchild = NULL;
temp->rchild = NULL;
return temp;}
void insert(node *root, node *new_node) {
if (new_node->data < root->data) {
if (root->lchild == NULL)
root->lchild = new_node;
else
insert(root->lchild, new_node);}
if (new_node->data > root->data) {
if (root->rchild == NULL)
root->rchild = new_node;
else
insert(root->rchild, new_node);}}
node *search(node *root, int key, node **parent) {
node *temp;
temp = root;
while (temp != NULL) {
if (temp->data == key) {
printf("\nThe %d Element is Present", temp->data);
flag = 1;
return temp;}
*parent = temp;
if (temp->data > key)
temp = temp->lchild;
else
temp = temp->rchild;}
return NULL;}
void inorder(node *temp) {
if (temp != NULL) {
inorder(temp->lchild);
printf("%d\t", temp->data);
inorder(temp->rchild);}}
void preorder(node *temp) {
if (temp != NULL) {
printf("%d\t", temp->data);
preorder(temp->lchild);
preorder(temp->rchild);}}
void postorder(node *temp) {
if (temp != NULL) {
postorder(temp->lchild);
postorder(temp->rchild);
printf("%d\t", temp->data);
}
}
```

11)GRAPH

```
#include <stdio.h>
#include <stdlib.h>
int a[20][20], q[20], visited[20], reach[10], n, i, j, f =
0, r = -1, count = 0;
void bfs(int v) {
for (i = 1; i <= n; i++)
if (a[v][i] && !visited[i])
q[++r] = i;
if (f <= r) {
visited[q[f]] = 1;
bfs(q[f++]);}}
void dfs(int v) {
int i;
reach[v] = 1;
for (i = 1; i <= n; i++) {
if (a[v][i] && !reach[i]) {
printf("\n %d->%d", v, i);
count++;
dfs(i);
}}}
void main() {
int v, choice;
printf("\n Enter the number of vertices:");
scanf("%d", &n);
for (i = 1; i <= n; i++) {
q[i] = 0;
visited[i] = 0;}
for (i = 1; i <= n - 1; i++)
reach[i] = 0;
printf("\n Enter graph data in matrix form:\n");
for (i = 1; i <= n; i++)
for (j = 1; j <= n; j++)
scanf("%d", &a[i][j]);
printf("1.BFS\n 2.DFS\n 3.Exit\n");
scanf("%d", &choice);
switch (choice) {
case 1:
printf("\n Enter the starting vertex:");
scanf("%d", &v);
bfs(v);
if ((v < 1) || (v > n)) {
printf("\n Bfs is not possible");
} else {
printf("\n The nodes which are reachable from
%d:\n", v);
for (i = 1; i <= n; i++)
if (visited[i])
printf("%d\t", i);}
break;
case 2:
dfs(1);
if (count == n - 1)
printf("\n Graph is connected");
else
printf("\n Graph is not connected");
break;
case 3:
exit(0);
}}
```

12) Hash function

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
int create(int);
void linear_prob(int[], int, int);
void display(int[]);
void main() {
int a[MAX], num, key, i;
int ans = 1;
printf(" Collision handling by linear probing : \n");
for (i = 0; i < MAX; i++) {
a[i] = -1;}
do {
printf("\n Enter the data: ");
scanf("%4d", &num);
key = create(num);
linear_prob(a, key, num);
printf("\n Do you wish to continue? (1/0) ");
scanf("%d", &ans);
} while (ans);
display(a);}
int create(int num) {
int key;
key = num % 100;
return key;}
void linear_prob(int a[MAX], int key, int num) {
int flag, i, count = 0;
flag = 0;
if (a[key] == -1) {
a[key] = num;
} else {
printf("\nCollision Detected...!!!\n");
i = 0;
while (i < MAX) {
if (a[i] != -1)
count++;
i++;}
printf("Collision avoided successfully using
LINEAR PROBING\n");
if (count == MAX) {
printf("\n Hash table is full");
display(a);
exit(1);}
for (i = key + 1; i < MAX; i++)
if (a[i] == -1) {
a[i] = num;
flag = 1;
break;}
i = 0;
while ((i < key) && (flag == 0))
 {
if (a[i] == -1) {
a[i] = num;
flag = 1;
break;}
i++;}}}
void display(int a[MAX]) {
int i, choice;
printf("1.Display ALL\n 2.Filtered Display\n");
scanf("%d",
&choice);
if (choice == 1) {
printf("\n the hash
table is\n");
for (i = 0; i < MAX;
i++)
printf("\n %d %d ",
i, a[i]);
} else {
printf("\n the hash
table is\n");
for (i = 0; i < MAX;
i++)
if (a[i] != -1) {
printf("\n %d %d ",
i, a[i]);
      continue;}}}
```