1.

a) Define a structure called Student with the members: Name, Reg_no, marks in 3 tests and average_ marks. Develop a menu driven program to perform the following by writing separate function for each operation: a) read information of N students b) display student's information c) to calculate the average of best two test marks of each student. Note: Allocate memory dynamically and illustrate the use of pointer to an array of structure.

b) Define a structure called Time containing 3 integer members (Hour, Minute, Second). Develop a menu driven program to perform the following by writing separate function for each operation. a) Read (T) :To read time b) Display (T):To display time c) update(T):To Update time d) Add (T1, T2) : Add two time variables. Update function increments the time by one second and returns the new time (if the increment results in 60 seconds, then the second member is set to zero and minute member is incremented by one. If the result is 60 minutes, the minute member is set to zero and the hour member is incremented by one. Finally, when the hour becomes 24, Time should be reset to zero. While adding two time variables, normalize the resultant time value as in the case of update function. Note: Illustrate the use of pointer to pass time variable to different functions

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
typedef struct student{
char name[50];
int rn;
float m[3];
float avgm;
}STUDENT;
int n;

STUDENT *read(STUDENT *s){
    printf("enter number of students required: ");
    scanf("%d",&n);
    s=(STUDENT*)malloc(n*sizeof(STUDENT));
    for(int i=0;i<n;i++){
        printf("enter name of student-%d: ",i+1);
        scanf("%s",s[i].name);
        printf("enter roll number of student-%d: ",i+1);
        scanf("%d",&s[i].rn);
        for(int j=0;j<3;j++){
            printf("enter test %d marks of student -%d: ",(j+1),(i+1));
            scanf("%f",&s[i].m[j]);
        }
    }
    return s;
}

void display(STUDENT *s){
    printf("NAME\tROLL NO\tTEST-1\tTEST-2\tTEST-3\t\n");
    for(int i=0;i<n;i++){

printf("%s\t%d\t%f\t%f\t%f\t\n",s[i].name,s[i].rn,s[i].m[0],s[i].m[1],s[i].m[2]);
    }
}

STUDENT *calcavg(STUDENT *s){
    printf("NAME\tAVERAGE MARKS\t\n");
    for(int i=0;i<n;i++){
        if(s[i].m[0]<=s[i].m[1]&&s[i].m[0]<=s[i].m[2])
        s[i].avgm=(s[i].m[2]+s[i].m[1])/2;
        if(s[i].m[1]<=s[i].m[0]&&s[i].m[1]<=s[i].m[2])
        s[i].avgm=(s[i].m[2]+s[i].m[0])/2;
        if(s[i].m[2]<=s[i].m[1]&&s[i].m[2]<=s[i].m[1])
        s[i].avgm=(s[i].m[1]+s[i].m[0])/2;
        printf("%s\t%f\t\n",s[i].name,s[i].avgm);
    }
    return s;
}

void main(){
    STUDENT *s;
    int choice;
    printf("1.read the student's information\n2.display\ncalculate
average\n0.exit\n");
    do{
        printf("enter choice: ");scanf("%d",&choice);
        switch(choice){
        case 1:s=read(s);break;
        case 2:display(s);break;
        case 3:s=calcavg(s);break;
        case 0:printf("--------------------------------------------------
THANKYOU---------------------------------------------------------\n");break;
        default:printf("wrong choice\n");break;
        }
    }while(choice!=0);
}

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
typedef struct time{
    int h,m,s;
}TIME;

TIME *read(TIME *p);
TIME *update(TIME *p);
TIME *correct(TIME *p);
void display(TIME *p);
TIME *add();

TIME *read(TIME *p){
    printf("enter time in hour:minute:second format:  " );
    scanf("%d %d %d",&p->h,&p->m,&p->s);
   p=correct(p);
    return p;
}

TIME *update(TIME *p){
    p->s=p->s+1;
   p=correct(p);
    return p;
}

TIME *correct(TIME *p){
     if(p->s>=60){
        p->m=p->m+p->s/60;
        p->s%=60;
    }
    if(p->m>=60){
        p->h=p->h+p->m/60;
        p->m%=60;
    }
     if(p->h>=24)
    {p->h%=24;}
    return p;
   }

void display(TIME *p){
    printf("TIME :\n%d:%d:%d\n",p->h,p->m,p->s);
}

TIME *add(TIME *a,TIME *q){
    TIME *r;
    r->h=a->h+q->h;
    r->m=a->m+q->m;
    r->s=a->s+q->s;
    r=correct(r);
    return r;
}
```

```
void main(){
    TIME *p,sr,*a,sp,*q,sq;
    p=&sr;
    a=&sp;
    q=&sq;
    int choice;
    printf("1.read time\n2.display time\n3.update time\n4.add two time\n0.exit\n");
    do{
        printf("enter choice: ");scanf("%d",&choice);
        switch(choice){
            case 1:p=read(p);break;
            case 2:display(p);break;
            case 3:p=update(p);break;
            case 4:q=read(q);a=read(a);p=add(a,q);break;
            case 0:printf("---------THANKYOU--------\n");break;
            default:printf("wrong choice\n");break;

        }
    }while(choice!=0);
}
```

2. Develop a menu driven program to implement the following operations on an array of integers with dynamic memory allocation. i) Insert by position ii) Delete by position iii) Insert by key iv) Delete by key v) Insert by order vi) Search by key vii) Search by position viii) Reverse the contents.

```
#include<stdio.h>
#include<stdlib.h>
int n;
int searchk(int *a,int key);

void read(int *a){
    printf("enter values: ");
    for(int i=0;i<n;i++){
    scanf("%d",&a[i]);
    }
}

void display(int *a){
    printf("elements are: \n");
    for(int i=0;i<n;i++){
        printf("%d",a[i]);
        printf("\n");
}
}

void insertp(int *a,int pos,int val){
if(pos<=n&&pos>=1){
  n++;a=(int *)realloc(a,n*sizeof(int));
for(int i=n;i>=pos-1;i--){
    a[i+1]=a[i];
}a[pos-1]=val;}
else
printf("invalid position!!\n");
}

void deletep(int *a,int pos){
if(pos<=n&&pos>=1){
for(int i=pos-1;i<n;i++){
    a[i]=a[i+1];
}n--;a=(int *)realloc(a,n*sizeof(int));}
else
printf("invalid position!!\n");
}

void insertk(int a[],int val){
   int pos=searchk(a,val);
   if(pos==0)
   printf("not found!!\n");
   else
   insertp(a,pos,val);
}
```

```
void deletek(int *a,int val){
   int pos=searchk(a,val);
   if(pos==0)
   printf("not found!!\n");
   else
   deletep(a,pos);
}

int searchk(int *a,int key){
    for(int i=0;i<n;i++)
        if(a[i]==key)
            return (i+1);
    return 0;
}

void searchp(int *a,int pos){
    if(pos<=n&&pos>=1)
        printf("value at %d position is %d\n",pos,a[pos-1]);
    else
    printf("invalid position\n");
}

void inserto(int *a,int val){
    int t;n++;a=(int *)realloc(a,n*sizeof(int));
    a[n-1]=val;
    for(int i=0;i<n-1;i++){
        for(int j=0;j<n-i-1;j++){
            if(a[j]>a[j+1]){
                t=a[j];
                a[j]=a[j+1];
                a[j+1]=t;
            }
        }
    }
}

void reverse(int *a){
     int temp;
    for(int i = 0; i<n/2; i++){
        temp = a[i];
        a[i] = a[n-i-1];
        a[n-i-1] = temp;
    }
}
void main(){
    int *a;
    int choice,pos,val,key;
    printf("1.read\n2.display\n3.insert by position\n4.delete by position\n5.insert
by key\n6.delete by key\n7.search value\n8.search position\n9.reverse\n10.insert by
order\n0.exit\n");
    for(;;){
      printf("enter choice: ");scanf("%d",&choice);
                switch(choice){
                    case 1:
                    printf("enter number of values required: ");
                    scanf("%d",&n);
                    a=(int*)malloc(n*sizeof(int));
                    read(a);
                    break;
                    case 2:
                    display(a);
                    break;
                    case 3:
                    printf("enter position and value to insert:");
                    scanf("%d %d",&pos,&val);
                    insertp(a,pos,val);
                    break;
                    case 4:
                    printf("enter position to delete:");
                    scanf("%d",&pos);
                    deletep(a,pos);
                    break;
                    case 5:
                    printf("enter key after which insertion should be done:");
                    scanf("%d",&key);
                    printf("enter value to insert");
                    scanf("%d",&val);
                    insertk(a,val);
                    break;
                    case 6:
                    printf("enter value to delete:");
                    scanf("%d",&val);
                    deletek(a,val);
                    break;
                    case 7:
                    printf("enter value to search:");
                    scanf("%d",&val);
                    pos=searchk(a,val);
                    if(pos==0)
                    printf("not found\n");
                    else
                    printf("%d found at position %d\n",val,pos);
                    break;
                    case 8:
                    printf("enter position to search:");
                    scanf("%d",&pos);
                    searchp(a,pos);
                    break;
                    case 9:
                    reverse(a);
                    break;
                    case 10:
                    printf("enter value to insert:");
                    scanf("%d",&val);
                    inserto(a,val);
                    break;
                    case 0:
                    printf("---------THANKYOU---------\n");
                    exit(0);
                    default:
                    printf("wrong choice\n");
                    break;}}}
```

3. Implement circular single linked list to perform the following operations i) Insert by order ii) Delete by position iii) Search for an item by key Display the list contents after each operation

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node{
  int data;
  struct node *link;
}NODE;
int n;

NODE*getnode(int val){
  NODE *nn=(NODE *)malloc(sizeof(NODE));
  nn->link=nn;
  nn->data=val;
  return nn;
}

NODE *inserto(NODE *h,int val){
  NODE *tm=h;
  NODE *nn=getnode(val);
  if(h==NULL)
  h=nn;
  else if(val<h->data){
    nn->link=h;
    while(tm->link!=h)
    tm=tm->link;
    tm->link=nn;
    h=nn;
  }
  else{
    while(tm->link!=h&&val>tm->link->data)
    tm=tm->link;
    nn->link=tm->link;
    tm->link=nn;
  }
  n++;
  return h;
}

NODE *deletep(NODE *h,int pos){
  NODE *tm=h,*pn=h;
  if(h==NULL)
    printf("empty!!!!\n");
  else if(pos>n||pos<1)
  printf("invalid position!!\n");
  else if(pos==1){
    if(h->link==h)
    h=NULL;
    else{
    while(tm->link!=h)
    tm=tm->link;
    tm->link=h->link;
    h=h->link;}
    n--;
    free(pn);
  }
  else{
    for(int i=1;i<pos-1;i++)
    tm=tm->link;
    pn=tm->link;
    tm->link=pn->link;
    n--;
    free(pn);
  }
  return h;
}

void display(NODE *h){
  if(h==NULL)
  printf("empty!!!\n");
  else{
    NODE *tm=h;
    do{
      printf("%d ",tm->data);
      tm=tm->link;
    }while(tm!=h);
    printf("\n");
  }
}

void searchk(NODE *h,int val){
  if(h==NULL)
  printf("empty!!!\n");
  else{
    NODE *tm=h;
    for(int i=0;i<n;i++){
    if(tm->data==val){
      printf("%d found at position %d\n",val,i+1);
      return;
    }
  }
  printf("not found!!\n");
  }
}

void main(){
  NODE *h=NULL;
  int val,pos,ch;
  printf("1.create ordered list\n2.delete by key\n3.search by key\n4.display\n0.exit\n");
  for(;;){
    printf("enter choice: ");
    scanf("%d",&ch);
    switch(ch){
    case 1:
    printf("enter value: ");
    scanf("%d",&val);
    h=inserto(h,val);
    break;
    case 2:
    printf("enter position to delete: ");
    scanf("%d",&pos);
    h=deletep(h,pos);
    break;
    case 3:
    printf("enter value to search: ");
    scanf("%d",&val);
    searchk(h,val);
    break;
    case 4:
    display(h);
    break;
    case 0:
    printf("THANKYOU\n");
    exit(0);
    default:
    printf("wrong choice\n");
    break;
    }
  }
}
```

4. Implement circular double linked list to perform the following operations i) Insert by order ii) Delete by key iii) Search by position Display the list contents after each operation

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node{
  int data;
  struct node *l,*r;
}NODE;

NODE*getnode(int val){
  NODE *nn=(NODE *)malloc(sizeof(NODE));
  nn->l=nn->r=nn;
  nn->data=val;
  return nn;
}

void display(NODE *h){
  if(h->r==h){
  printf("empty!!!\n");
  return;}
  NODE *tm=h->r;
  while(tm!=h){
    printf("%d ",tm->data);
    tm=tm->r;
  }
  printf("\n");
}

NODE *inserto(NODE *h,int val){
  NODE *nn=getnode(val);
  NODE *tm=h;
  if(h->r==h){
    nn->r=h->r;
    nn->l=h;
    h->r=nn;
    h->l=nn;
    h->data++;
    return h;
  }
  while(tm->r!=h&&val>tm->r->data)
  tm=tm->r;
  nn->r=tm->r;
  nn->l=tm;
  tm->r->l=nn;
  tm->r=nn;
  h->data++;
  return h;
}

NODE *deletek(NODE *h,int val){
  if(h->r==h){
  printf("empty!!!\n");
  return h;}
    NODE *tm=h->r;
    while(tm!=h&&tm->data!=val)
    tm=tm->r;
    if(tm==h)
    printf("not found!!\n");
    else{
    tm->l->r=tm->r;
    tm->r->l=tm->l;
    h->data--;
    free(tm);
  }
  return h;
}

NODE *searchp(NODE *h,int pos){
  if(h->r==h){
  printf("empty!!!\n");
  return h;}
  if(pos>h->data||pos<1){
    printf("invalid position!!\n");
    return h;
  }
  NODE *tm=h->r;
  for(int i=1;i<pos;i++)
  tm=tm->r;
  printf("value at position %d is %d\n",pos,tm->data);
}

void main(){
  NODE *h=(NODE*)malloc(sizeof(NODE));
  h->data=0;
  h->r=h->l=h;
  int val,pos,ch;
  printf("1.insert by order\n2.delete by key\n3.search by
position\n4.display\n0.exit\n");
  for(;;){
    printf("enter choice: ");
    scanf("%d",&ch);
    switch(ch){
    case 1:
    printf("enter value: ");
    scanf("%d",&val);
    h=inserto(h,val);
    break;
    case 2:
    printf("enter value: ");
    scanf("%d",&val);
    h=deletek(h,val);
    break;
    case 3:
    printf("enter position to search: ");
    scanf("%d",&pos);
    searchp(h,pos);
    break;
    case 4:
```

```
    display(h);
    break;
    case 0:
    printf("THANKYOU\n");
    exit(0);
    default:
    printf("wrong choice\n");
    break;
    }
  }
}
```

5. Implement circular single linked list to perform the following operations i) Insert front ii) Insert rear iii) Delete a node with the given key iv) Search for an item by position Display the list contents after each operation

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node{
  int data;
  struct node *link;
}NODE;
int n;

NODE*getnode(int val){
  NODE *nn=(NODE *)malloc(sizeof(NODE));
  nn->link=nn;
  nn->data=val;
  return nn;
}

NODE *insertf(NODE *h,int val){
  NODE *nn=getnode(val);
  n++;
  if(h==NULL){
    h=nn;
    return h;
  }
  nn->link=h;
  NODE *tm=h;
  while(tm->link!=h)
  tm=tm->link;
  tm->link=nn;
  h=nn;
  return h;
}

NODE *insertr(NODE *h,int val){
  NODE *nn=getnode(val);
  n++;
  if(h==NULL){
    h=nn;
    return h;
  }
  NODE *tm=h;
  while(tm->link!=h)
  tm=tm->link;
  nn->link=tm->link;
  tm->link=nn;
  return h;
}

NODE *deletek(NODE *h,int val){
  int i;
  if(h==NULL){
  printf("empty!!!!\n");
  return h;}
    NODE *tm=h,*pn=NULL;
    for(i=0;i<n&&val!=tm->data;i++){
      pn=tm;
      tm=tm->link;}
    if(i==n)
    printf("key not found!!\n");
    else if(i==0){
       if(h->link==h)
    h=NULL;
        else{
    tm=h;pn=h;
    while(tm->link!=h)
    tm=tm->link;
    tm->link=h->link;
    h=h->link;}
    n--;
    free(pn);
    }
    else{
      pn->link=tm->link;
      free(tm);
      n--;
    }
  return h;
}

void display(NODE *h){
  if(h==NULL)
  printf("empty!!!\n");
  else{
    NODE *tm=h;
    do{
      printf("%d ",tm->data);
      tm=tm->link;
    }while(tm!=h);
    printf("\n");
  }
}

void searchp(NODE *h,int pos){
  if(h==NULL)
  printf("empty!!!\n");
  else if(pos>n||pos<1)
  printf("invalid position!!\n");
  else{
    NODE *tm=h;
    for(int i=1;i<pos;i++)
    tm=tm->link;
    printf("value at position %d is %d\n",pos,tm->data);
  }
 }

void main(){
  NODE *h=NULL;
  int val,pos,ch;
  printf("1.insert front\n2.insert rear\n3.delete by key\n4.search by
position\n5.display\n0.exit\n");
  for(;;){
    printf("enter choice: ");
    scanf("%d",&ch);
    switch(ch){
    case 1:
    printf("enter value: ");
    scanf("%d",&val);
    h=insertf(h,val);
    break;
    case 2:
    printf("enter value: ");
    scanf("%d",&val);
    h=insertr(h,val);
    break;
    case 3:
    printf("enter value: ");
    scanf("%d",&val);
    h=deletek(h,val);
    break;
    case 4:
```

```
    printf("enter position to search: ");
    scanf("%d",&pos);
    searchp(h,pos);
    break;
    case 5:
    display(h);
    break;
    case 0:
    printf("THANKYOU\n");
    exit(0);
    default:
    printf("wrong choice\n");
    break;
    }
  }
}
```

6. Implement circular double linked list to perform the following operations i) Insert front ii) Insert rear iii) Delete kth node iv) Search for an item by value. Display the list contents after each operation

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node{
  int data;
  struct node *l,*r;
}NODE;

NODE*getnode(int val){
  NODE *nn=(NODE *)malloc(sizeof(NODE));
  nn->l=nn->r=nn;
  nn->data=val;
  return nn;
}

void display(NODE *h){
  if(h->r==h){
  printf("empty!!!\n");
  return;}
  NODE *tm=h->r;
  while(tm!=h){
    printf("%d ",tm->data);
    tm=tm->r;
  }
  printf("\n");
}

NODE *insertf(NODE *h,int val){
  NODE *nn=getnode(val);
  nn->r=h->r;
  nn->l=h;
  h->r->l=nn;
  h->r=nn;
  h->data++;
  return h;
}

NODE *insertr(NODE *h,int val){
  NODE *nn=getnode(val);
  nn->r=h;
  nn->l=h->l;
  h->l->r=nn;
  h->l=nn;
  h->data++;
  return h;
}

NODE *deletep(NODE *h,int pos){
  if(h->r==h){
  printf("empty!!!\n");
  return h;}
  if(pos>h->data||pos<1){
    printf("invalid position!!\n");
    return h;
  }
  NODE *tm=h->r;
  for(int i=1;i<pos;i++)
  tm=tm->r;
  tm->l->r=tm->r;
  tm->r->l=tm->l;
  h->data--;
  free(tm);
  return h;
}

void searchk(NODE *h,int val){
  if(h->r==h){
  printf("empty!!!\n");
  return;}
  NODE *tm=h->r;
  int i=1;
  while(tm!=h&&tm->data!=val){
  tm=tm->r;
  i++;}
  if(tm==h)
  printf("not found!!\n");
  else
  printf("%d found at position %d\n",val,i);
}

void main(){
  NODE *h=(NODE*)malloc(sizeof(NODE));
  h->data=0;
  h->r=h->l=h;
  int val,pos,ch;
  printf("1.insert at front\n2.insert at rear\n3.delete by position\n4.search by
position\n5.display\n0.exit\n");
  for(;;){
    printf("enter choice: ");
    scanf("%d",&ch);
    switch(ch){
    case 1:
    printf("enter value: ");
    scanf("%d",&val);
    h=insertf(h,val);
    break;
    case 2:
    printf("enter value: ");
    scanf("%d",&val);
    h=insertr(h,val);
    break;
    case 3:
    printf("enter position to delete: ");
    scanf("%d",&pos);
    h=deletep(h,pos);
    break;
    case 4:
    printf("enter value to search: ");
    scanf("%d",&val);
    searchk(h,val);
    break;
    case 5:
    display(h);
    break;
    case 0:
    printf("THANKYOU\n");
    exit(0);
    default:
    printf("wrong choice\n");
    break;
    }
  }
}
```

7. Implement circular single linked list to perform the following operations i) Insert by position ii) Delete rear iii) Delete Front iv) Search for an item by value Display the list contents after each operation

```c
#include<stdio.h>
#include<stdlib.h>
typedef struct node{
  int data;
  struct node *link;
}NODE;
int n;

NODE*getnode(int val){
  NODE *nn=(NODE *)malloc(sizeof(NODE));
  nn->link=nn;
  nn->data=val;
  return nn;
}

NODE *deletef(NODE *h){
  if(h==NULL){
  printf("empty!!!\n");
  return h;}
  NODE *tm=h;
  while(tm->link!=h)
  tm=tm->link;
  tm->link=h->link;
  tm=h;
  h=h->link;n--;
  free(tm);
  return h;
}

NODE *deleter(NODE *h){
  if(h==NULL){
  printf("empty!!!\n");
  return h;}
  NODE *tm=h,*pn=NULL;
  while(tm->link!=h){
  pn=tm;
  tm=tm->link;}
  pn->link=tm->link;
  free(tm);n--;
  return h;
}

NODE *insertp(NODE *h,int val,int pos){
  NODE *nn=getnode(val);
  NODE *tm=h,*pn=h;
  if(h==NULL){
    h=nn;n++;
    return h;}
  else if(pos>n||pos<1)
  printf("invalid position!!\n");
  else if(pos==1){
    nn->link=h;
    while(tm->link!=h)
    tm=tm->link;
    tm->link=nn;
    h=nn;n++;
  }
  else{
    for(int i=1;i<pos-1;i++)
    tm=tm->link;
    nn->link=tm->link;
    tm->link=nn;n++;
  }
  return h;
}

void display(NODE *h){
  if(h==NULL)
  printf("empty!!!\n");
  else{
    NODE *tm=h;
    do{
      printf("%d ",tm->data);
      tm=tm->link;
    }while(tm!=h);
    printf("\n");
  }
}

void searchp(NODE *h,int pos){
  if(h==NULL)
  printf("empty!!!\n");
  else if(pos>n||pos<1)
  printf("invalid position!!\n");
  else{
    NODE *tm=h;
    for(int i=1;i<pos;i++)
    tm=tm->link;
    printf("value at position %d is %d\n",pos,tm->data);
  }
 }

void main(){
  NODE *h=NULL;
  int val,pos,ch;
  printf("1.delete front\n2.delete rear\n3.insert by position\n4.search by
position\n5.display\n0.exit\n");
  for(;;){
    printf("enter choice: ");
    scanf("%d",&ch);
    switch(ch){
    case 1:
    h=deletef(h);
    break;
    case 2:
    h=deleter(h);
    break;
    case 3:
    printf("enter value and position to insert: ");
    scanf("%d %d",&val,&pos);
    h=insertp(h,val,pos);
    break;
    case 4:
    printf("enter position to search: ");
    scanf("%d",&pos);
    searchp(h,pos);
    break;
    case 5:
    display(h);
    break;
    case 0:
    printf("THANKYOU\n");
    exit(0);
    default:
    printf("wrong choice\n");
    break;
    }
  }
}
```

8. Implement circular double linked list to perform the following operations i) Insert by order ii) Delete rear iii) Delete Front iv) Search for an item by position Display the list contents after each operation

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node{
  int data;
  struct node *l,*r;
}NODE;
```

```
NODE*getnode(int val){
  NODE *nn=(NODE *)malloc(sizeof(NODE));
  nn->l=nn->r=nn;
  nn->data=val;
  return nn;
}

void display(NODE *h){
  if(h->r==h){
  printf("empty!!!\n");
  return;}
  NODE *tm=h->r;
  while(tm!=h){
    printf("%d ",tm->data);
    tm=tm->r;
  }
  printf("\n");
}

NODE *deletef(NODE *h){
  if(h->r==h){
  printf("empty!!!\n");
  return h;}
  NODE *tm=h->r;
  tm->l->r=tm->r;
  tm->r->l=tm->l;
  h->data--;
  free(tm);
  return h;
}

NODE *deleter(NODE *h){
  if(h->r==h){
  printf("empty!!!\n");
  return h;}
  NODE *tm=h->l;
  tm->l->r=tm->r;
  tm->r->l=tm->l;
  h->data--;
  free(tm);
  return h;
}

NODE *inserto(NODE *h,int val){
  NODE *nn=getnode(val);
  NODE *tm=h;
  if(h->r==h){
    nn->r=h->r;
    nn->l=h;
    h->r=nn;
    h->l=nn;
    h->data++;
    return h;
  }
  while(tm->r!=h&&val>tm->r->data)
  tm=tm->r;
  nn->r=tm->r;
  nn->l=tm;
  tm->r->l=nn;
  tm->r=nn;
  h->data++;
  return h;
}

NODE *deletep(NODE *h,int pos){
  if(h->r==h){
  printf("empty!!!\n");
  return h;}
  if(pos>h->data||pos<1){
    printf("invalid position!!\n");
    return h;
  }
  NODE *tm=h->r;
  for(int i=1;i<pos;i++)
  tm=tm->r;
  tm->l->r=tm->r;
  tm->r->l=tm->l;
  h->data--;
  free(tm);
  return h;
}

NODE *searchp(NODE *h,int pos){
  if(h->r==h){
  printf("empty!!!\n");
  return h;}
  if(pos>h->data||pos<1){
    printf("invalid position!!\n");
    return h;
  }
  NODE *tm=h->r;
  for(int i=1;i<pos;i++)
  tm=tm->r;
  printf("value at position %d is %d\n",pos,tm->data);
}

void main(){
  NODE *h=(NODE*)malloc(sizeof(NODE));
  h->data=0;
  h->r=h->l=h;
  int val,pos,ch;
  printf("1.insert by order\n2.delete at rear\n3.delete at front\n4.search by
position\n5.display\n0.exit\n");
  for(;;){
    printf("enter choice: ");
    scanf("%d",&ch);
    switch(ch){
    case 1:
    printf("enter value: ");
    scanf("%d",&val);
    h=inserto(h,val);
    break;
    case 2:
    h=deleter(h);
    break;
    case 3:
    h=deletef(h);
    break;
    case 4:
    printf("enter position to search: ");
    scanf("%d",&pos);
    searchp(h,pos);
    break;
    case 5:
    display(h);
    break;
    case 0:
    printf("THANKYOU\n");
    exit(0);
    default:
    printf("wrong choice\n");
    break;
    }
  }
}
```

9. Develop a menu driven program to convert infix expression to postfix expression using stack (Test for nested parenthesized expressions)

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

```
int stackprec(char symbol){
switch(symbol){
case '+' :
case '-' : return 2;
case '*' :
case '/' :
case '%' : return 4;
case '^' :
case '$' : return 5;
case '(' : return 0;
case '#' : return -1;
default : return 8;
}
}

int inprec(char symbol){
switch(symbol){
case '+' :
case '-' : return 1;
case '*' :
case '/' :
case '%' : return 3;
case '^' :
case '$' : return 6;
case '(' : return 9;
case ')' : return 0;
default : return 7;
}
}

void infixtopostfix(char infix[], char postfix[]){
char S[30];
int i=0;
int j=0;
int top=-1;
S[++top] = '#';
char symbol;
for(i=0;i<strlen(infix);i++){
symbol = infix[i];
while(stackprec(S[top])>inprec(symbol))
postfix[j++]=S[top--];
if(stackprec(S[top])!=inprec(symbol))
S[++top] = symbol;
else
top--;
}
while(S[top]!='#'){
postfix[j]=S[top--];
j++;
}
postfix[j]='\0';
}
void main(){
char infix[30];
char postfix[30];
int choice;
printf("1.Convert to Postfix\n0.Exit\n");
for(;;){
printf("Enter Your Choice: ");
scanf("%d",&choice);
switch (choice){
case 1 : printf("Enter the Infix Expression : ");
scanf("%s",infix);
 infixtopostfix(infix,postfix);
printf("The Postfix Expression is : ");
printf("%s\n",postfix);
break;
case 0 : printf("!! THANK YOU !!\n");
exit(0);
default: printf("Invalid Choice\n");
}
}
}
```

10. Develop a menu driven program to convert infix expression to prefix expression using stack (Test for nested parenthesized expressions)

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

int stackprec(char symbol)
{
switch(symbol)
{
case '+' :
case '-' : return 1;
case '*' :
case '/' :
case '%' : return 3;
case '^' :
case '$' : return 6;
case ')' : return 0;
case '#' : return -1;
default : return 8;
}
}

int inprec(char symbol)
{
switch(symbol)
{
case '+' :
case '-' : return 2;
case '*' :
case '/' :
case '%' : return 4;
case '^' :
case '$' : return 5;
case '(' : return 0;
case ')' : return 9;
default : return 7;
}
}

void infixtoprefix(char infix[], char prefix[])
{
char S[30];
int i=0;
int j=0;
int top=-1;
S[++top] = '#';
char symbol;
char temp;
for(i=0;i<strlen(infix)/2;i++){
    temp=infix[i];
    infix[i]=infix[strlen(infix)-i-1];
    infix[strlen(infix)-i-1]=temp;
}
for(i=0;i<strlen(infix);i++)
{
symbol = infix[i];
while(stackprec(S[top])>inprec(symbol))
{
prefix[j++]=S[top--];
}
if(stackprec(S[top])!=inprec(symbol))
{
S[++top] = symbol;
}
else
{
top--;
}
}
while(S[top]!='#')
{
prefix[j]=S[top--];
j++;
}
prefix[j]='\0';
for(i=0;i<strlen(prefix)/2;i++){
    temp=prefix[i];
    prefix[i]=prefix[strlen(prefix)-i-1];
    prefix[strlen(prefix)-i-1]=temp;
}
}
void main()
{
char infix[30];
char prefix[30];
int choice;
printf("1.Convert to Prefix\n0.Exit\n");
for(;;)
{
printf("Enter Your Choice: ");
scanf("%d",&choice);
switch (choice)
{
case 1 : printf("Enter the Infix Expression : ");
scanf("%s",infix);
printf("The Prefix Expression is : ");
infixtoprefix(infix,prefix);
printf("%s\n",prefix);
break;
case 0: printf("!! THANK YOU !!\n");
exit(0);
default: printf("Invalid Choice\n");
}
}
}
```

11. Develop a menu driven program to evaluate postfix and prefix expressions using stack.

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

void prefixevaluate(char prefix[])
{
int i,op1,op2,res;
strrev(prefix);
int Stack[10];
int top=-1;
char ch;
for(i=0;i<strlen(prefix);i++)
{
ch = prefix[i];
if(isdigit(ch))
{
Stack[++top]=ch-'0';
}
else
{
if(top<1)
printf("Not enough operands\n");
else
{
op1=Stack[top--];
op2=Stack[top--];
switch (ch)
{
case '+' : res = op1 + op2;
break;
case '-' : res = op1 - op2;
break;
case '/' : res = op1 / op2;
break;
case '*' : res = op1 * op2;
break;
case '%' : res = op1 % op2;
break;
default: printf("Invalid Operator\n");
}
Stack[++top] = res;
}
}
}
printf("Final Result = %d\n",Stack[top]);
}

void postfixevaluate(char postfix[])
{
int i,op1,op2,res;
int Stack[10];
int top=-1;
char ch;
for(i=0;i<strlen(postfix);i++)
{
ch = postfix[i];
if(isdigit(ch))
{
Stack[++top]=ch-'0';
}
else
{
if(top<1)
printf("Not enough operands\n");
else
{
op2=Stack[top--];
op1=Stack[top--];
switch (ch)

{
case '+' : res = op1 + op2;
break;
case '-' : res = op1 - op2;
break;
case '/' : res = op1 / op2;
break;
case '*' : res = op1 * op2;
break;
case '%' : res = op1 % op2;
break;
default: printf("Invalid Operator\n");
}
Stack[++top] = res; //Pushing Result into Stack
}
}
}
printf("Final Result = %d\n",Stack[top]);
}

void main()
{
char prefix[30];
char postfix[30];
int choice;
printf("Menu\n1.evaluate prefix\n2.Evaluate Postfix\n0.Exit\n");
for(;;)
{
printf("Enter Your Choice: ");
scanf("%d",&choice);
switch (choice)
```

```
{
case 1 : printf("Enter the prefix Expression : ");
scanf("%s",prefix);
prefixevaluate(prefix);
break;
case 2:printf("Enter the postfix Expression : ");
scanf("%s",postfix);
postfixevaluate(postfix);
break;
case 0 : printf("!! THANK YOU !!\n");
exit(0);
default: printf("Invalid Choice\n");
}
}
}
```

12. Develop a menu driven program to implement the following types of Queues by allocating memory dynamically. i) Circular Queue ii) Double ended Queue

```
#include <stdio.h>
#include <stdlib.h>
int n = 0,max_size=3;

typedef struct queue
{
  int *queue;
  int front, rear;
}QUEUE;

QUEUE insert(QUEUE Q, int item)
{
  if(n==max_size)
  {
    printf("Queue overflow\nreallocating memory\n");
    max_size++;
    Q.queue=(int*)realloc(Q.queue,max_size*sizeof(int));
  }
    Q.rear = (Q.rear + 1) % max_size;
    Q.queue[Q.rear] = item;
    if (Q.front == -1)
      Q.front = 0;
    n = n + 1;
  return Q;
}

QUEUE delete (QUEUE Q)
{
  int item;
  if (n==0)
  {
    printf("Queue is empty\n");
  }
  else
  {
    item = Q.queue[Q.front];
    Q.front = (Q.front + 1) % max_size;
    n--;
    printf("Deleted item is %d\n", item);
  }
  return Q;
}

void display(QUEUE Q)
{
  int i, j;
  if (n==0)
  {
    printf("Queue empty\n");
    return;
  }
  printf("Contents are :\n");
  j = Q.front;
  for (i = 1; i <= n; i++)
  {
    printf("Q[%d]=%d\t", j, Q.queue[j]);
    j = (j + 1) % max_size;
  }printf("\n");
}


QUEUE insert_rear(QUEUE Q, int item)
{
  if (Q.rear == max_size - 1){
    printf("Queue overflow\nreallocating memory\n");
    max_size++;
    Q.queue=(int*)realloc(Q.queue,max_size*sizeof(int));
  }
    Q.rear = Q.rear + 1;
    Q.queue[Q.rear] = item;
    if (Q.front == -1)
      Q.front = 0;
  return Q;
}

QUEUE insert_front(QUEUE Q, int item)
{
  if (Q.front == 0)
  {
    printf("cannot insert from front end\n");
   return Q;
  }
  if (Q.front == -1){
    Q.front++;
    Q.rear = 0;}
  else
    Q.front--;
  Q.queue[Q.front] = item;
  return Q;
}

QUEUE delete_front(QUEUE Q)
{
  int item;
  if (Q.front == -1 || Q.front > Q.rear)
  {
    printf("Queue is empty\n");
    return Q;
  }
    item = Q.queue[Q.front];
    Q.front = Q.front + 1;
    printf("Deleted item is %d\n", item);
    return Q;
}

QUEUE delete_rear(QUEUE Q)
{
  int item;
  if (Q.rear == -1 || Q.front > Q.rear)
  {
    printf("Queue is empty\n");
    return Q;
  }
    item = Q.queue[Q.rear];
    Q.rear = Q.rear - 1;
    printf("Deleted item is %d\n", item);
    return Q;
}

void Ddisplay(QUEUE Q)
{
  int i;
  if (Q.rear != -1 && (Q.front > Q.rear))
    printf("\nQueue empty\n");
    else{
```

```
  printf("\nContents are :\n");
  for (i = Q.front; i <= Q.rear; i++)
    printf("%d\t", Q.queue[i]);
    printf("\n");
    }
}
void main()
{
  QUEUE Q;
  Q.queue=(int*)malloc(max_size*sizeof(int));
  Q.front = -1;
  Q.rear = -1;
  int val,ele, choice;
    printf("1.circular queue\n2.double ended queue\n");
    printf("Enter your choice\n");
    scanf("%d", &choice);
    switch (choice){
      case 1:
      printf("1: Insert\n2: Delete\n3.Display\n4: Exit\n");
     for (;;)
     {
    printf("Enter your choice\n");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
      printf("Enter the element\n");
      scanf("%d", &val);
      Q=insert(Q, val);
      break;
    case 2:
      Q=delete (Q);
      break;
    case 3:
      display(Q);
      break;
    default:
      exit(0);
    }
  }
  case 2:
  printf("1: Insert Front\n2: Insert Rear\n3: Delete Front\n4: Delete Rear\n5:
Display\n0: Exit\n");
  for(;;){
    printf("Enter your choice\n");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
      printf("Enter the element\n");
      scanf("%d", &ele);
      Q=insert_front(Q, ele);
      break;
    case 2:
      printf("Enter the element\n");
      scanf("%d", &ele);
      Q=insert_rear(Q, ele);
      break;
    case 3:
      Q=delete_front(Q);
      break;
    case 4:
      Q=delete_rear(Q);
      break;
    case 5:
      Ddisplay(Q);
      break;
    default:
      exit(0);}
    }
  }
  }
```

13. Develop a menu driven program to implement the following types of Queues by allocating memory dynamically. i) Circular Queue ii) Priority Queue

```
#include <stdio.h>
#include <stdlib.h>
int n = 0,max_size=3;

typedef struct queue
{
  int *queue;
  int front, rear;
}QUEUE;

QUEUE insert(QUEUE Q, int item)
{
  if(n==max_size)
  {
    printf("Queue overflow\nreallocating memory\n");
    max_size++;
    Q.queue=(int*)realloc(Q.queue,max_size*sizeof(int));
  }
    Q.rear = (Q.rear + 1) % max_size;
    Q.queue[Q.rear] = item;
    if (Q.front == -1)
      Q.front = 0;
    n = n + 1;
  return Q;
}

QUEUE delete (QUEUE Q)
{
  int item;
  if (n==0)
  {
    printf("Queue is empty\n");
  }
  else
  {
    item = Q.queue[Q.front];
    Q.front = (Q.front + 1) % max_size;
    n--;
    printf("Deleted item is %d\n", item);
  }
  return Q;
}

void display(QUEUE Q)
{
  int i, j;
  if (n==0)
  {
    printf("Queue empty\n");
    return;
  }
  printf("Contents are :\n");
  j = Q.front;
  for (i = 1; i <= n; i++)
  {
    printf("Q[%d]=%d\t", j, Q.queue[j]);
    j = (j + 1) % max_size;
  }printf("\n");
}


QUEUE ins(QUEUE Q,int ele)
{
  int j;
   if (Q.rear == max_size - 1){
    printf("Queue overflow\nreallocating memory\n");
    max_size++;
    Q.queue=(int*)realloc(Q.queue,max_size*sizeof(int));}
    j = Q.rear;
   while (j >= 0 && ele <Q.queue[j]) {
    Q.queue[j + 1] =Q.queue[j];
     j--;  }
    Q.queue[j + 1] = ele;
    Q.rear++;
  return Q;
}

QUEUE del(QUEUE Q)
{
 if (Q.front > Q.rear)
   printf("Queue empty\n");
 else
  printf("The element deleted is %d\n",Q.queue[Q.front++]);
  return Q;
}

void disp(QUEUE Q)
{
   int i;
 if (Q.front > Q.rear){
   printf("Queue empty\n");
   return; }
       for (i = Q.front; i <= Q.rear; i++)
       printf("%d\t",Q.queue[i]);
       printf("\n");
}

void main()
{
  QUEUE Q;
  Q.queue=(int*)malloc(max_size*sizeof(int));
  Q.front = -1;
  Q.rear = -1;
  int val,ele, choice;
    printf("1.circular queue\n2.priority queue\n");
    printf("Enter your choice\n");
    scanf("%d", &choice);
    switch (choice){
      case 1:
      printf("1: Insert\n2: Delete\n3.Display\n4: Exit\n");
     for (;;)
     {
    printf("Enter your choice\n");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
      printf("Enter the element\n");
      scanf("%d", &val);
      Q=insert(Q, val);
      break;
    case 2:
      Q=delete (Q);
      break;
    case 3:
      display(Q);
      break;
    default:
      exit(0);
    }
  }
  case 2:Q.front=0;
  printf("1: Insert \n2: Delete Rear\n3: Display\n0: Exit\n");
  for(;;){
    printf("Enter your choice\n");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
      printf("Enter the element\n");
      scanf("%d", &ele);
      Q=ins(Q, ele);
      break;
    case 2:
      Q=del(Q);
      break;
    case 3:
      disp(Q);
      break;
    default:
      exit(0);}
    }
  }
  }
```

14. 14. Develop a menu driven program to implement Binary Search tree with the following operations. i) Construction ii) Traversals (Pre, In and Post Order) iii) Searching a node by key and displaying its information along with its parent is exists, otherwise a suitable message. iv) Counting all types of nodes. v) Finding height

```
#include<stdio.h>
#include<stdlib.h>
typedef struct tree{
    int data;
    struct tree *llink,*rlink;
}TREE;
TREE *create(int key);

TREE *insert(TREE *root,int key)//to insert a node
{
    if(root==NULL)
    return create(key);
    if(key<=root->data)
    root->llink= insert(root->llink,key);
    else
    root->rlink= insert(root->rlink,key);
    return root;
}

TREE *create(int key){     //this function is for insert operation
    TREE *node=(TREE*)malloc(sizeof(TREE));
    node->data=key;
    node->llink=node->rlink=NULL;
    return node;
}

//traversals
void inorder(TREE *root){
    if(root!=NULL){
        inorder(root->llink);
        printf("%d ",root->data);
        inorder(root->rlink);
    }
}

void preorder(TREE *root){
    if(root!=NULL){
        printf("%d ",root->data);
        preorder(root->llink);
        preorder(root->rlink);
    }
}

void postorder(TREE *root){
    if(root!=NULL){
        postorder(root->llink);
        postorder(root->rlink);
        printf("%d ",root->data);
    }
}
```

```
//count non leaf nodes
int countnl(TREE *root){
    if(root==NULL||(root->llink==NULL&&root->rlink==NULL))
    return 0;
    return 1+countnl(root->llink)+countnl(root->rlink);
}

//count leaf nodes
int countl(TREE *root){
    if(root==NULL)
    return 0;
    if(root->llink==NULL&&root->rlink==NULL)
    return 1;
    else
    return countl(root->llink)+countl(root->rlink);
}

//search a node and print its info along with parent
int search(TREE *root,int key){
    if(root==NULL){
    printf("node not found\n");
    return 0;}
    if((root->llink!=NULL&&root->llink->data==key)||(root->rlink!=NULL&&root->rlink-
>data==key)){
    printf("value %d node found ,its parent node is %d\n",key,root->data);
    return 1;}
    if(key<root->data)
    search(root->llink,key);
    else
    search(root->rlink,key);
}

//find height of node
int height(TREE *root){
    if(root==NULL||(root->llink==NULL&&root->rlink==NULL))
    return 0;
    else{
        int l=height(root->llink);
        int r=height(root->rlink);
        if(l>r)
        return (l+1);
        else
        return (r+1);
    }
}

void main(){
    TREE *root=NULL;
    int ch,n,key,val;
    printf("1.create tree\n2.preorder traversal\n3.inorder traversal\n4.postorder
traversal\n5.count all types of nodes\n6.find height\n7.search value\n0.exit\n");
    for(;;){
        printf("enter choice: ");
        scanf("%d",&ch);
        switch(ch){
            case 1:
            printf("enter number of nodes required: ");
            scanf("%d",&n);
            for(int i=0;i<n;i++){
                printf("enter key value: ");
                scanf("%d",&key);
                root=insert(root,key);
            }
            break;
            case 2:
            printf("PREORDER: ");
            preorder(root);
            printf("\n");
            break;
            case 3:
            printf("INORDER: ");
            inorder(root);
            printf("\n");
            break;
            case 4:
            printf("POSTORDER: ");
            postorder(root);
            printf("\n");
            break;
            case 5:
            printf("number of leaf nodes: %d\n",countl(root));
            printf("number of non-leaf nodes: %d\n",countnl(root));
            break;
            case 6:
            printf("height of the tree is: %d\n",height(root));
            break;
            case 7:
            printf("enter a value to search: ");
            scanf("%d",&val);
            key=search(root,val);
            break;
            case 0:
            printf("THANKYOU\n");
            exit(0);
            default:
            printf("wrong choice\n");
            break;
        }
    }
}
```

15. Develop a menu driven program to implement Binary Search tree with the following operations. i) Construction ii) Traversals(Pre, In and Post Order) iii) Searching a node by key and deleting if exists (node to be deleted may be leaf or non- leaf with one child or two children

```
#include<stdio.h>
#include<stdlib.h>
typedef struct tree{
    int data;
    struct tree *llink,*rlink;
}TREE;
TREE *create(int );
TREE *min(TREE *root);


TREE *insert(TREE *root,int key)//to insert a node
{
    if(root==NULL)
    return create(key);
    if(key<=root->data)
    root->llink= insert(root->llink,key);
    else
    root->rlink= insert(root->rlink,key);
    return root;
}

TREE *create(int key){     //this function is for insert operation
    TREE *node=(TREE*)malloc(sizeof(TREE));
    node->data=key;
    node->llink=node->rlink=NULL;
    return node;
}

//traversals
void inorder(TREE *root){
    if(root!=NULL){
        inorder(root->llink);
        printf("%d ",root->data);
        inorder(root->rlink);
    }
}

void preorder(TREE *root){
    if(root!=NULL){
        printf("%d ",root->data);
        preorder(root->llink);
        preorder(root->rlink);
    }
}

void postorder(TREE *root){
    if(root!=NULL){
        postorder(root->llink);
        postorder(root->rlink);
        printf("%d ",root->data);
    }
}

TREE *delete(TREE *root,int key){   //to delete a node
    if(root==NULL)
    return root;
    if(key<root->data)
    root->llink=delete(root->llink,key);
    else if(key>root->data)
    root->rlink=delete(root->rlink,key);
    else{
        if(root->llink==NULL){
            TREE *temp=root->rlink;
            free(root);
            return temp;
        }
        else if(root->rlink==NULL){
            TREE *temp=root->llink;
            free(root);
            return temp;
        }
        TREE *temp=min(root->rlink);
        root->data=temp->data;
        root->rlink=delete(root->rlink,temp->data);
    }
    return root;
}

TREE *min(TREE *root){     //this function is for delete
    TREE *temp=root;
    while(temp&&temp->llink!=NULL)
    temp=temp->llink;
    return temp;
}

void main(){
    TREE *root=NULL;
    int ch,n,key,val;
    printf("1.create tree\n2.preorder traversal\n3.inorder traversal\n4.postorder
traversal\n5.delete a node\n0.exit\n");
    for(;;){
        printf("enter choice: ");
        scanf("%d",&ch);
        switch(ch){
            case 1:
            printf("enter number of nodes required: ");
            scanf("%d",&n);
            for(int i=0;i<n;i++){
                printf("enter key value: ");
                scanf("%d",&key);
                root=insert(root,key);
            }
            break;
            case 2:
            printf("PREORDER: ");
            preorder(root);
            printf("\n");
            break;
            case 3:
```

```
            printf("INORDER: ");
            inorder(root);
            printf("\n");
            break;
            case 4:
            printf("POSTORDER: ");
            postorder(root);
            printf("\n");
            break;
            case 5:
            printf("enter a value to delete a node: ");
            scanf("%d",&val);
            root=delete(root,val);
            break;
            case 0:
            printf("THANKYOU\n");
            exit(0);
            default:
            printf("wrong choice\n");
            break;
        }
    }
}
```