```c
//quick sort with pivot pointing to the last index
#include<stdio.h>
void quicksort(int [],int,int);
int partition(int [],int,int);
main()
{
    int arr[40],size,i,low,high;
    printf("Enter the size of the array : ");
    scanf("%d",&size);
    low = 0;
    high = size-1;
    for(i=0;i<size;i++)
    {
        printf("Enter element : ");
        scanf("%d",&arr[i]);
    }
    printf("Array before sorting is : \n");
    for(i=0;i<size;i++)
    {
        printf("%d ",arr[i]);
    }
    printf("\n");
    quicksort(arr,low,high);
    printf("Array after sorting is : \n");
    for(i=0;i<size;i++)
    {
        printf("%d ",arr[i]);
    }
    printf("\n");
}
void quicksort(int arr[],int low,int high)
{
    int position;
    if(low<high)
    {
        position = partition(arr,low,high);
        quicksort(arr,low,position-1);
        quicksort(arr,position+1,high);
    }
}
int partition(int arr[],int low,int high)
{
    int pivot,i,j,temp;
    pivot = arr[high];
    j = low;
    i = j-1;
    while(j<high)
    {
        if(arr[j]<=pivot)
        {
            i++;
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
```

```
        }
        j++;
    }
    temp = arr[i+1];
    arr[i+1] = pivot;
    arr[high] = temp;
    return (i+1);
}



//quick sort with pivot pointing to the starting index
#include<stdio.h>
void quicksort(int [],int,int);
int partition(int [],int,int);
main()
{
    int arr[40],size,i,low,high;
    printf("Enter the size of the array : ");
    scanf("%d",&size);
    low = 0;
    high = size-1;
    for(i=0;i<size;i++)
    {
        printf("Enter element : ");
        scanf("%d",&arr[i]);
    }
    printf("Array before sorting is : \n");
    for(i=0;i<size;i++)
    {
        printf("%d ",arr[i]);
    }
    printf("\n");
    quicksort(arr,low,high);
    printf("Array after sorting is : \n");
    for(i=0;i<size;i++)
    {
        printf("%d ",arr[i]);
    }
    printf("\n");
}
void quicksort(int arr[],int low,int high)
{
    int position;
    if(low<=high)
    {
        position = partition(arr,low,high);
        quicksort(arr,low,position-1);
        quicksort(arr,position+1,high);
    }
}
int partition(int arr[],int low,int high)
{
    int pivot,i,j,temp;
```

```
    pivot = arr[low];
    j = low+1;
    i = j-1;
    while(j<=high)
    {
        if(arr[j]<=pivot)
        {
            i++;
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
        j++;
    }
    temp = arr[i];
    arr[i] = pivot;
    arr[low] = temp;
    return (i);
}


//merge sort
#include<stdio.h>
void merge(int arr[],int low,int high,int mid)
{
    int i,j,k,temp[40];
    i = low;
    j = mid+1;
    k = low;
    while((i<=mid)&&(j<=high))
    {
        if(arr[i]<=arr[j])
        {
            temp[k] = arr[i];
            i++;
            k++;
        }
        else
        {
            temp[k] = arr[j];
            j++;
            k++;
        }
    }
    while(i<=mid)
    {
        temp[k] = arr[i];
        k++;
        i++;
    }
    while(j<=high)
    {
        temp[k] = arr[j];
        k++;
```

```
        j++;
    }
    k = 0;
    for(i=0;i<=high;i++)
    {
        arr[k] = temp[i];
        k++;
    }
}
void mergesort(int arr[],int low,int high)
{
    int mid;
    if(low<high)
    {
        mid = (low+high)/2;
        mergesort(arr,low,mid);
        mergesort(arr,mid+1,high);
        merge(arr,low,high,mid);
    }
}
main()
{
    int arr[40],i,size;
    printf("Enter the size of the array : ");
    scanf("%d",&size);
    for(i=0;i<size;i++)
    {
        printf("Enter the element : ");
        scanf("%d",&arr[i]);
    }
    printf("The array before sorting is : \n");
    for(i=0;i<size;i++)
    {
        printf("%d ",arr[i]);
    }
    mergesort(arr,0,size-1);
    printf("\nThe array after sorting is : \n");
    for(i=0;i<size;i++)
    {
        printf("%d ",arr[i]);
    }
    printf("\n");
}
```

```
//heap sort
#include<stdio.h>
void maxheapify(int [],int,int);
void buildmaxheap(int [],int);
void heapsort(int [],int);
main()
{
    int arr[40],size,i;
```

```
    printf("Enter the size of the array : ");
    scanf("%d",&size);
    for(i=0;i<size;i++)
    {
        printf("Enter the element : ");
        scanf("%d",&arr[i]);
    }
    printf("The array before sorting is : \n");
    for(i=0;i<size;i++)
    {
        printf("%d ",arr[i]);
    }
    heapsort(arr,size-1);
    printf("\nThe array after sorting is : \n");
    for(i=0;i<size;i++)
    {
        printf("%d ",arr[i]);
    }
    printf("\n");
}
void maxheapify(int arr[],int index,int size)
{
    int large,left,right,temp;
    large = index;
    left = (2*index)+1;
    right = (2*index)+2;
    if((left<=size)&&(arr[left]>arr[index]))
    {
        large = left;
    }
    else
    {
        large = index;
    }
    if((right<=size)&&(arr[right]>arr[large]))
    {
        large = right;
    }
    if(large != index)
    {
        temp = arr[index];
        arr[index] = arr[large];
        arr[large] = temp;
        maxheapify(arr,large,size);
    }
}
void buildmaxheap(int arr[],int size)
{
    int index;
    for(index=(size-1)/2;index>=0;index--)
    {
        maxheapify(arr,index,size);
    }
}
```

```
void heapsort(int arr[],int size)
{
    int value,temp;
    buildmaxheap(arr,size);
    for(value = size;value>=1;value--)
    {
        temp = arr[0];
        arr[0] = arr[size];
        arr[size] = temp;
        size = size-1;
        maxheapify(arr,0,size);
    }
}
```

//single linked list

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int info;
    struct Node *next;
};
struct Node *start = NULL;
struct Node *start1 = NULL;
void traverse();
void create();
void insertbegin();
void insertend();
void insertafternode();
void insertlocation();
void deletebegin();
void deleteend();
void deleteafternode();
void deletelocation();
void reverse();
void bubblesort();
void merge();
int main()
{
    int choice , ch=1;
    while(ch)
    {
        printf("\nWELCOME TO LINKED LIST FUNCTIONS : \n");
        printf("1.CREATE\n2.TRAVERSE\n3.INSERT AT BEGINNING\n4.INSERT AT END\n5.INSERT
AFTER A GIVEN NODE\n6.INSERT AT SPECIFIC INDEX\n7.DELETE FROM BEGINNING\n8.DELETE FROM
END\n9.DELETE AFTER A GIVEN NODE\n10.DELETE FROM SPECIFIC INDEX\n11.REVERSE THE
LIST\n12.SORT THE LIST\n13.MERGE THE SINGLE LINKED LIST\n14.EXIT\n");
        printf("\nEnter your choice : ");
        scanf("%d",&choice);
        switch (choice)
        {
            case 1 : create();
                     break;
```

```
            case 2 : traverse();
                     break;
            case 3 : insertbegin();
                     break;
            case 4 : insertend();
                     break;
            case 5 : insertafternode();
                     break;
            case 6 : insertlocation();
                     break;
            case 7 : deletebegin();
                     break;
            case 8 : deleteend();
                     break;
            case 9 : deleteafternode();
                     break;
            case 10 : deletelocation();
                      break;
            case 11 : reverse();
                      break;
            case 12 : bubblesort();
                      printf("AFTER SORTING ");
                      traverse();
                      break;
            case 13 : merge();
                      break;
            case 14 : ch = 0;
                      break;
            default : printf("Wrong choice \n");
        }
    }
    return 0;
}
void create()
{
    int item,i,n;
    struct Node *newnode,*temp;
    printf("\nEnter the number of elements of the linked list : ");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        newnode=(struct Node *)malloc(sizeof(struct Node));
        printf("Enter the item you want to insert : ");
        scanf(" %d",&item);
        newnode->info = item;
        newnode->next = NULL;
        if(start == NULL)
        {
            start = newnode;
        }
        else
        {
            temp = start;
            while(temp->next != NULL)
```

```
            {
                temp = temp->next;
            }
            temp->next = newnode;
        }
    }
}
void traverse()
{
    struct Node *temp;
    if (start == NULL)
    {
        printf("\nLIST IS EMPTY\n");
    }
    else
    {
        temp = start;
        printf("\nTHE LIST IS : \n");
        while(temp!=NULL)
        {
            printf("%d ",temp->info);
            temp = temp->next;
        }
        printf("\n");
    }
}
void insertbegin()
{
    struct Node *newnode;
    int item;
    newnode = (struct Node *)malloc(sizeof(struct Node));
    printf("\nEnter the item to be inserted at the beginning : ");
    scanf("%d",&item);
    newnode->info = item;
    newnode->next = NULL;
    if(start == NULL)
    {
        start = newnode;
    }
    else
    {
        newnode->next = start;
        start = newnode;
    }
}
void insertend()
{
    struct Node *newnode,*temp;
    int item;
    newnode = (struct Node*)malloc(sizeof(struct Node));
    printf("\nEnter the item to be inserted at the end : ");
    scanf("%d",&item);
    newnode->info = item;
    newnode->next = NULL;
```

```
    if(start == NULL)
    {
        start = newnode;
    }
    else
    {
        temp = start;
        while(temp->next != NULL)
        {
            temp = temp->next;
        }
        temp->next = newnode;
    }
}
void deletebegin()
{
    struct Node *temp;
    if(start == NULL)
    {
        printf("\nLIST IS EMPTY\n");
    }
    else
    {
        temp = start;
        printf("\nDeleted item is %d \n",temp->info);
        start = start->next;
        free(temp);
    }
}
void deleteend()
{
    struct Node *temp,*temp1;
    if(start == NULL)
    {
        printf("\nLIST IS EMPTY\n");
    }
    else if(start->next == NULL)
    {
        temp = start;
        printf("\nDeleted item is %d \n",start->info);
        start = NULL;
        free(temp);
    }
    else
    {
        temp = start;
        while(temp->next != NULL)
        {
            temp1 = temp;
            temp = temp->next;
        }
        printf("\nDeleted item is %d \n",temp->info);
        free(temp);
        temp1->next = NULL;
```

```
    }
}
void insertafternode()
{
    struct Node *newnode,*temp;
    int item,value;
    newnode = (struct Node *) malloc (sizeof(struct Node));
    printf("\nEnter the item you want to insert : ");
    scanf("%d",&item);
    newnode->info = item;
    newnode->next = NULL;
    if(start == NULL)
    {
        start = newnode;
    }
    else
    {
            printf("\nEnter the value after which you want to insert : ");
            scanf("%d",&value);
        temp = start;
        while((temp!=NULL)&&(temp->info!=value))
        {
            temp = temp->next;
        }
        if(temp == NULL)
        {
                    printf("\nThe value after which you want to insert doesn't exist\n");
        }
        else
        {
            newnode->next = temp->next;
            temp->next = newnode;
        }
    }
}
void deleteafternode()
{
    struct Node *temp,*temp1;
    int value;
    if(start == NULL)
    {
        printf("\nLIST IS EMPTY\n");
    }
    else
    {
        printf("\nEnter the value after which you want to delete : ");
        scanf("%d",&value);
        temp = start;
        while((temp!=NULL)&&(temp->info!=value))
        {
            temp = temp->next;
        }
        if(temp==NULL)
        {
```

```
            printf("\nThe value after which you want to delete doesn't exist\n");
        }
        else if(temp->next == NULL)
        {
            printf("\nThere is no value after %d to be deleted \n",value);
        }
        else
        {
            temp1 = temp;
            temp = temp->next;
            printf("\nDeleted item is %d \n",temp->info);
            temp1->next = temp->next;
            free(temp);
        }
    }
}
void bubblesort()
{
    struct Node *temp,*temp1;
    int temp2;
    if(start == NULL)
    {
        printf("\nLIST IS EMPTY\n");
    }
    else
    {
        for(temp = start;temp!=NULL;temp = temp->next)
        {
            for(temp1 = temp->next ; temp1!=NULL ; temp1 = temp1->next)
            {
                if(temp->info > temp1->info)
                {
                    temp2 = temp1->info;
                    temp1->info = temp->info;
                    temp->info = temp2;
                }
            }
        }
    }
}
void insertlocation()
{
    struct Node *newnode,*temp,*temp1;
    int item,index,i;
    newnode = (struct Node*)malloc(sizeof(struct Node));
    printf("\nEnter the item to be inserted  : ");
    scanf("%d",&item);
    newnode->info = item;
    newnode->next = NULL;
    if(start == NULL)
    {
        start = newnode;
    }
    else
```

```
    {
        printf("\nEnter the index at which you want to insert : ");
        scanf("%d",&index);
        temp = start;
        for(i = 1;i<index;i++)
        {
            temp1 = temp;
            temp = temp->next;
        }
        newnode->next = temp;
        temp1->next = newnode;
    }
}
void deletelocation()
{
    struct Node *temp,*temp1;
    int index,i=1;
    if(start == NULL)
    {
        printf("\nLIST IS EMPTY\n");
    }
    else if(start->next == NULL)
    {
        temp = start;
        start = NULL;
        printf("\nDeleted item is %d \n",temp->info);
        free(temp);
    }
    else
    {
        printf("\nEnter the index from which you want to delete : ");
        scanf("%d",&index);
        if(index == 1)
        {
            temp = start;
            printf("Deleted item is %d\n",temp->info);
            start = temp->next;
            free(temp);
        }
        else
        {
            temp = start;
            while(temp != NULL && i!=index)
            {
                temp1 = temp;
                temp = temp->next;
                i++;
            }
            if(temp == NULL)
            {
                printf("\nThere is no value at index %d\n",index);
            }
            else
            {
```

```
                printf("\nDeleted item is %d \n",temp->info);
                temp1->next = temp->next;
                free(temp);
            }
        }
    }
}
void reverse()
{
    struct Node *temp,*temp1,*ptr;
    int i=1,j=1,temp2;
    if(start == NULL)
    {
        printf("\nLIST IS EMPTY\n");
    }
    else
    {
        temp = start;
        temp1 = start;
        while(temp->next != NULL)
        {
            temp = temp->next;
            j++;
        }
        while(i<=j)
        {
            temp2 = temp1->info;
            temp1->info = temp->info;
            temp->info = temp2;
            temp1 = temp1->next;
            ptr = start;
            while(ptr->next!=temp)
            {
                ptr= ptr->next;
            }
            temp = ptr;
            i++;
            j--;
        }
    }
}
void merge()
{
    struct Node *temp,*newnode;
    int item,i,n;
    printf("\nEnter the number of elements of the other linked list : ");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        newnode=(struct Node *)malloc(sizeof(struct Node));
        printf("Enter the item you want to insert : ");
        scanf(" %d",&item);
        newnode->info = item;
        newnode->next = NULL;
```

```
        if(start1 == NULL)
        {
            start1 = newnode;
        }
        else
        {
            temp = start1;
            while(temp->next != NULL)
            {
                temp = temp->next;
            }
            temp->next = newnode;
        }
    }
    if((start == NULL)&&(start1 == NULL))
    {
        printf("\nLISTS ARE EMPTY\n");
    }
    else if(start == NULL)
    {
        printf("\nMERGED LIST IS : \n");
        for(temp = start1;temp!=NULL;temp = temp->next)
        {
            printf("%d ",temp->info);
        }
        printf("\n");
    }
    else if(start1 == NULL)
    {
        printf("\nMERGED LIST IS : \n");
        for(temp = start;temp!=NULL;temp = temp->next)
        {
            printf("%d ",temp->info);
        }
        printf("\n");
    }
    else
    {
        temp = start;
        while(temp->next!=NULL)
        {
            temp = temp->next;
        }
        temp->next = start1;
        printf("\nMERGED LIST IS : \n");
        for(temp = start;temp!=NULL;temp = temp->next)
        {
            printf("%d ",temp->info);
        }
        printf("\n");
    }
}
```

```c
//double linked list
#include<stdio.h>
#include<stdlib.h>
struct node
{
    struct node *prev;
    int info;
    struct node *next;
};
struct node *start = NULL;
void create();
void traverse();
void insertbegin();
void insertend();
void insertnode();
void insertloc();
void deletebegin();
void deleteend();
void deletenode();
void deleteloc();
main()
{
    int choice;
    while(1)
    {
        printf("WELCOME TO LINKED LIST FUNCTIONS :\n");
        printf("1.CREATE\n2.TRAVERSE\n3.INSERT AT BEGINNING\n4.INSERT AT END\n5.INSERT
AFTER NODE\n6.INSERT AT SPECIFIC LOCATION\n7.DELETE FROM BEGINNING\n8.DELETE FROM
END\n9.DELETE AFTER NODE\n10.DELETE FROM SPECIFIC LOCATION\n11.EXIT\n");
        printf("Enter your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : create();
                 break;
            case 2 : traverse();
                 break;
            case 3 : insertbegin();
                 break;
            case 4 : insertend();
                 break;
            case 5 : insertnode();
                 break;
            case 6 : insertloc();
                 break;
            case 7 : deletebegin();
                 break;
            case 8 : deleteend();
                 break;
            case 9 : deletenode();
                 break;
            case 10 : deleteloc();
                  break;
            case 11 : exit(0);
```

```
                break;
            default : printf("Wrong Choice\n");
        }
    }
}
void create()
{
    int size,i,item;
    struct node *newnode,*temp;
    printf("Enter the no. of elements of the list : ");
    scanf("%d",&size);
    for(i=0;i<size;i++)
    {
        newnode = (struct node*)malloc(sizeof(struct node));
        printf("Enter item : ");
        scanf("%d",&item);
        newnode->info = item;
        newnode->prev = NULL;
        newnode->next = NULL;
        if(start == NULL)
        {
            start = newnode;
        }
        else
        {
            temp = start;
            while(temp->next!=NULL)
            {
                temp = temp->next;
            }
            newnode->prev = temp;
            temp->next = newnode;
        }
    }
}
void traverse()
{
    struct node *temp,*temp1;
    if(start == NULL)
    {
        printf("LIST IS EMPTY\n");
    }
    else
    {
        temp = start;
        printf("The list in forward direction is  : \n");
        while(temp!=NULL)
        {
            printf("%d ",temp->info);
            temp1 = temp;
            temp = temp->next;
        }
        printf("\nThe list in backward direction is :\n");
        while(temp1!=NULL)
```

```
        {
            printf("%d ",temp1->info);
            temp1 = temp1->prev;
        }
        printf("\n");
    }
}
void insertbegin()
{
    struct node *newnode;
    int item;
    newnode = (struct node *)malloc(sizeof(struct node));
    printf("Enter the item to be inserted at the beginning : ");
    scanf("%d",&item);
    newnode->info = item;
    newnode->next = NULL;
    newnode->prev = NULL;
    if(start == NULL)
    {
        start = newnode;
    }
    else
    {
        newnode->next = start;
        start->prev = newnode;
        start = newnode;
    }
}
void insertend()
{
    struct node *newnode,*temp;
    int item;
    newnode = (struct node *)malloc(sizeof(struct node));
    printf("Enter the item you want to insert at the end : ");
    scanf("%d",&item);
    newnode->info = item;
    newnode->prev = NULL;
    newnode->next = NULL;
    if(start == NULL)
    {
        start = newnode;
    }
    else
    {
        temp = start;
        while(temp->next != NULL)
        {
            temp = temp->next;
        }
        newnode->prev = temp;
        temp->next = newnode;
    }
}
void insertnode()
```

```
{
    int item,value;
    struct node *newnode,*temp;
    newnode = (struct node*)malloc(sizeof(struct node));
    printf("Enter the item you want to insert : ");
    scanf("%d",&item);
    newnode->info = item;
    newnode->prev = NULL;
    newnode->next = NULL;
    if(start == NULL)
    {
        start = newnode;
    }
    else
    {
        printf("Enter the value after which you want to insert : ");
        scanf("%d",&value);
        temp = start;
        while((temp!=NULL)&&(temp->info!=value))
        {
            temp = temp->next;
        }
        if(temp == NULL)
        {
            printf("Value is not present \n");
        }
        else if(temp->next == NULL)
        {
            newnode->prev = temp;
            temp->next = newnode;
        }
        else
        {
            newnode->prev = temp;
            newnode->next = temp->next;
            (temp->next)->prev = newnode;
            temp->next = newnode;
        }
    }
}
void insertloc()
{
    struct node *newnode,*temp,*temp1;
    int item, loc,count = 1;
    newnode = (struct node*)malloc(sizeof(struct node));
    printf("Enter the item to be inserted : ");
    scanf("%d",&item);
    newnode->info = item;
    newnode->prev = NULL;
    newnode->next = NULL;
    if(start == NULL)
    {
        start = newnode;
    }
```

```
    else
    {
        temp = start;
        printf("Enter the location where you want to insert : ");
        scanf("%d",&loc);
        if(loc == 1)
        {
            newnode->next = start;
            start->prev = newnode;
            start = newnode;
        }
        else
        {
            while((temp!=NULL)&&(count!=loc))
            {
                temp1 = temp;
                temp = temp->next;
                count++;
            }
            if(temp == NULL)
            {
                printf("Node is not present \n");
            }
            else
            {
                newnode->prev = temp1;
                newnode->next = temp1->next;
                (temp1->next)->prev = newnode;
                temp1->next = newnode;
            }
        }
    }
}
void deletebegin()
{
    struct node *temp;
    if(start == NULL)
    {
        printf("LIST IS EMPTY\n");
    }
    else if(start->next == NULL)
    {
        temp = start;
        printf("Deleted item is %d\n",temp->info);
        start = NULL;
        free(temp);
    }
    else
    {
        temp=start;
        (temp->next)->prev = NULL;
        start = temp->next;
        printf("Deleted item is %d\n",temp->info);
        free(temp);
```

```
    }
}
void deleteend()
{
    struct node *temp;
    if(start == NULL)
    {
        printf("LIST IS EMPTY\n");
    }
    else if(start->next == NULL)
    {
        temp = start;
        printf("The deleted item is %d\n",start->info);
        start = NULL;
        free(temp);
    }
    else
    {
        temp = start;
        while(temp->next != NULL)
        {
            temp = temp->next;
        }
        printf("The deleted item is %d\n",temp->info);
        (temp->prev)->next = NULL;
        free(temp);
    }
}
void deletenode()
{
    struct node *temp,*temp1;
    int value;
    if(start == NULL)
    {
        printf("LIST IS EMPTY\n");
    }
    else if(start->next == NULL)
    {
        printf("No node is present to be deleted\n");
    }
    else
    {
        temp = start;
        printf("Enter the value of the node : ");
        scanf("%d",&value);
        while((temp!=NULL)&&(temp->info!=value))
        {
            temp = temp->next;
        }
        if(temp == NULL)
        {
            printf("Node is not present\n");
        }
        else if(temp->next == NULL)
```

```
        {
            printf("No node is present to be deleted\n");
        }
        else if((temp->next)->next == NULL)
        {
            temp1 = temp->next;
            printf("The deleted item is %d\n",temp1->info);
            temp->next = NULL;
            free(temp1);
        }
        else
        {
            temp1 = temp->next;
            printf("The deleted item is %d\n",(temp->next)->info);
            temp->next = (temp->next)->next;
            (temp1->next)->prev = temp1->prev;
            free(temp1);
        }
    }
}
void deleteloc()
{
    struct node *temp;
    int value,count = 1;
    if(start == NULL)
    {
        printf("LIST IS EMPTY\n");
    }
    else if(start->next == NULL)
    {
        printf("Enter the location of the element you want to delete : ");
        scanf("%d",&value);
        if(value == 1)
        {
            temp = start;
            printf("Deleted item is %d\n",start->info);
            start = NULL;
            free(temp);
        }
        else
        {
            printf("No value is present to be deleted\n");
        }
    }
    else
    {
        temp = start;
        printf("Enter the location of the element you want to delete : ");
        scanf("%d",&value);
        if(value == 1)
        {
            start = start->next;
            start->prev = NULL;
            printf("The deleted item is %d\n",temp->info);
```

```
            free(temp);
        }
        else
        {
            while((temp!=NULL)&&(count!=value))
            {
                temp = temp->next;
                count++;
            }
            if(temp == NULL)
            {
                printf("No value is present to be deleted\n");
            }
            else if(temp->next == NULL)
            {
                (temp->prev)->next = NULL;
                printf("The deleted item is %d\n",temp->info);
                free(temp);
            }
            else
            {
                printf("The deleted item is %d\n",temp->info);
                (temp->next)->prev = temp->prev;
                (temp->prev)->next = temp->next;
                free(temp);
            }
        }
    }
}
```

```
//circular linked list
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int info;
    struct node *next;
};
struct node *start = NULL;
void create()
{
    struct node *newnode,*temp;
    int size,i,item;
    printf("Enter the size of the list : ");
    scanf("%d",&size);
    for(i=0;i<size;i++)
    {
        newnode = (struct node*)malloc(sizeof(struct node));
        printf("Enter the item : ");
        scanf("%d",&item);
        newnode->info = item;
        newnode->next = newnode;
```

```
        if(start == NULL)
        {
            start = newnode;
        }
        else
        {
            temp = start;
            do
            {
                temp = temp->next;
            } while (temp->next!= start);
            newnode->next = temp->next;
            temp->next = newnode;
        }
    }
}
void insertbegin()
{
    struct node *temp,*newnode;
    int item;
    printf("Enter the item to be inserted at the beginning : ");
    scanf("%d",&item);
    newnode = (struct node*)malloc(sizeof(struct node));
    newnode->info = item;
    newnode->next = newnode;
    if(start == NULL)
    {
        start = newnode;
    }
    else
    {
        temp = start;
        while(temp->next!=start)
        {
            temp = temp->next;
        }
        newnode->next = start;
        start = newnode;
        temp->next = newnode;
    }
}
void traverse()
{
    struct node *temp;
    if(start == NULL)
    {
        printf("List is empty\n");
    }
    else
    {
        temp = start;
        printf("The list is : \n");
        do
        {
```

```
            printf("%d ",temp->info);
            temp = temp->next;
        }
        while (temp!=start);
        printf("\n");
    }
}
void insertend()
{
    struct node *newnode,*temp;
    int item;
    printf("Enter the item to be inserted at the end : ");
    scanf("%d",&item);
    newnode = (struct node *)malloc(sizeof(struct node));
    newnode->info = item;
    newnode->next = newnode;
    if(start == NULL)
    {
        start = newnode;
    }
    else
    {
        temp = start;
        do
        {
            temp = temp->next;
        } while (temp->next!= start);
        newnode->next = temp->next;
        temp->next = newnode;
    }
}
void insertnode()
{
    struct node *temp,*newnode;
    int item,value;
    printf("Enter the item to be inserted : ");
    scanf("%d",&item);
    newnode = (struct node *)malloc(sizeof(struct node));
    newnode->info = item;
    newnode->next = newnode;
    if(start == NULL)
    {
        start = newnode;
    }
    else
    {
        printf("Enter the value of the node after which you want to insert : ");
        scanf("%d",&value);
        if(start->info == value)
        {
            newnode->next = start->next;
            start->next = newnode;
        }
        else
```

```
        {
            temp = start;
            do
            {
                temp = temp->next;
            } while ((temp!=start)&&(temp->info!=value));
            if(temp == start)
            {
                printf("Node is not present\n");
            }
            else
            {
                newnode->next = temp->next;
                temp->next = newnode;
            }
        }
    }
}
void insertloc()
{
    struct node *newnode,*temp,*temp1;
    int item,loc,count=1;
    newnode = (struct node *)malloc(sizeof(struct node));
    printf("Enter the item to be inserted : ");
    scanf("%d",&item);
    newnode->info = item;
    newnode->next = newnode;
    if(start == NULL)
    {
        start = newnode;
    }
    else
    {
        printf("Enter the location at which you want to insert : ");
        scanf("%d",&loc);
        if(loc == 1)
        {
            temp = start;
            while(temp->next!=start)
            {
                temp = temp->next;
            }
            temp->next = newnode;
            newnode->next = start;
            start = newnode;
        }
        else
        {
            temp = start;
            do
            {
                temp1 = temp;
                temp = temp->next;
                count++;
```

```
            }
            while ((temp!=start)&&(count!=loc));
            if(temp == start)
            {
                printf("Item can not be inserted\n");
            }
            else
            {
                newnode->next = temp1->next;
                temp1->next = newnode;
            }
        }
    }
}
void deletebegin()
{
    struct node *temp,*temp1;
    if(start == NULL)
    {
        printf("List is empty\n");
    }
    else if(start == start->next)
    {
        temp = start;
        printf("Deleted item is %d \n",temp->info);
        start = NULL;
        free(temp);
    }
    else
    {
        temp = start;
        while(temp->next!=start)
        {
            temp = temp->next;
        }
        temp1 = start;
        temp->next = start->next;
        start = start->next;
        printf("Deleted item is %d\n",temp1->info);
        free(temp1);
    }
}
void deleteend()
{
    struct node *temp,*temp1;
    if(start == NULL)
    {
        printf("List is empty\n");
    }
    else if(start == start->next)
    {
        temp = start;
        printf("Deleted item is %d\n",temp->info);
        start = NULL;
```

```
        free(temp);
    }
    else
    {
        temp = start;
        while(temp->next!=start)
        {
            temp1 = temp;
            temp = temp->next;
        }
        temp1->next = temp->next;
        printf("Deleted item is %d\n",temp->info);
        free(temp);
    }
}
void deletenode()
{
    struct node *temp,*temp1;
    int value;
    if(start == NULL)
    {
        printf("List is empty\n");
    }
    else if(start == start->next)
    {
        printf("No node is present to be deleted\n");
    }
    else
    {
        temp = start;
        printf("Enter the node after which you want to delete : ");
        scanf("%d",&value);
        if(start->info == value)
        {
            temp = start->next;
            start->next = temp->next;
            printf("Deleted item is %d\n",temp->info);
            free(temp);
        }
        else
        {
            do
            {
                temp = temp->next;
            }
            while ((temp!=start)&&(temp->info!=value));
            if(temp == start)
            {
                printf("Node is not present\n");
            }
            else if(temp->next == start)
            {
                printf("No node is present to delete\n");
            }
```

```
            else
            {
                temp1 = temp->next;
                temp->next = temp1->next;
                printf("Deleted item is %d\n",temp1->info);
                free(temp1);
            }
        }
    }
}
void deleteloc()
{
    struct node *temp,*temp1;
    int loc,count=1;
    if(start == NULL)
    {
        printf("List is empty\n");
    }
    else if(start == start->next)
    {
        printf("Enter the location of the element you want to delete : ");
        scanf("%d",&loc);
        if(loc!=1)
        {
            printf("Node is not present\n");
        }
        else
        {
            temp = start;
            printf("Deleted item is %d\n",temp->info);
            start = NULL;
            free(temp);
        }
    }
    else
    {
        printf("Enter the location of the element you want to delete : ");
        scanf("%d",&loc);
        if(loc == 1)
        {
            temp = start;
            temp1 = start;
            while(temp->next!= start)
            {
                temp = temp->next;
            }
            temp->next = start->next;
            start = start->next;
            printf("Deleted item is %d\n",temp1->info);
            free(temp1);
        }
        else
        {
            temp = start;
```

```
            do
            {
                temp1 = temp;
                temp = temp->next;
                count++;
            }
            while ((temp!=start)&&(count!=loc));
            if(temp == start)
            {
                printf("Node is not present\n");
            }
            else
            {
                printf("Deleted item is %d\n",temp->info);
                temp1->next = temp->next;
                free(temp);
            }
        }
    }
}
main()
{
    int choice;
    while(1)
    {
        printf("WELCOME TO CIRCULAR LINKED LIST \n");
        printf("1.CREATE\n2.TRAVERSE\n3.INSERT AT THE BEGINNING\n4.INSERT AT END\n5.INSERT
AFTER NODE\n6.INSERT AT LOCATION\n7.DELETE FROM BEGINNING\n8.DELETE FROM END\n9.DELETE
AFTER NODE\n10.DELETE FROM LOCATION\n11.EXIT\n");
        printf("Enter your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : create();
                     break;
            case 2 : traverse();
                      break;
            case 3 : insertbegin();
                     break;
            case 4 : insertend();
                     break;
            case 5 : insertnode();
                     break;
            case 6 : insertloc();
                     break;
            case 7 : deletebegin();
                     break;
            case 8 : deleteend();
                     break;
            case 9 : deletenode();
                     break;
            case 10 : deleteloc();
                     break;
            case 11 : exit(0);
```

```
                    break;
            default : printf("Wrong choice\n");
        }
    }
}


//binary search tree
#include<stdio.h>
#include<stdlib.h>
struct node
{
    struct node *lchild;
    int info;
    struct node *rchild;
};
struct node *root = NULL;
void create()
{
    struct node *temp,*ptr,*newnode;
    int item,size,i;
    printf("Enter the number of elements of the tree : ");
    scanf("%d",&size);
    for(i=0;i<size;i++)
    {
        newnode = (struct node *)malloc(sizeof(struct node));
        printf("Enter the item : ");
        scanf("%d",&item);
        newnode->info = item;
        newnode->lchild = NULL;
        newnode->rchild = NULL;
        if(root == NULL)
        {
            root = newnode;
        }
        else
        {
            temp = root;
            while(temp!=NULL)
            {
                ptr = temp;
                if(item > temp->info)
                {
                    temp = temp->rchild;
                }
                else if(item < temp->info)
                {
                    temp = temp->lchild;
                }
                else
                {
                    printf("Duplicate value is not allowed \n");
                    i--;
                    break;
```

```
                }
            }
            if(ptr->info < item)
            {
                ptr->rchild = newnode;
            }
            if(ptr->info > item)
            {
                ptr->lchild = newnode;
            }
        }
    }
}
void preorder(struct node *temp)
{
    if(temp != NULL)
    {
        printf("%d ",temp->info);
        preorder(temp->lchild);
        preorder(temp->rchild);
    }
}
void inorder(struct node *temp)
{
    if(temp != NULL)
    {
        inorder(temp->lchild);
        printf("%d ",temp->info);
        inorder(temp->rchild);
    }
}
void postorder(struct node *temp)
{
    if(temp != NULL)
    {
        postorder(temp->lchild);
        postorder(temp->rchild);
        printf("%d ",temp->info);
    }
}
void insert()
{
    struct node *temp,*newnode,*ptr;
    int item;
    printf("Enter item to be inserted : ");
    scanf("%d",&item);
    newnode = (struct node *)malloc(sizeof(struct node));
    newnode->lchild = NULL;
    newnode->info = item;
    newnode->rchild = NULL;
    if(root == NULL)
    {
        root = newnode;
    }
```

```
    else
    {
        temp = root;
        while(temp!=NULL)
        {
            ptr = temp;
            if(temp->info == item)
            {
                printf("Duplicate item can't be inserted\n");
                return;
            }
            else if(temp->info > item)
            {
                temp = temp->lchild;
            }
            else
            {
                temp = temp->rchild;
            }
        }
        if(ptr->info > item)
        {
            ptr->lchild = newnode;
        }
        else
        {
            ptr->rchild = newnode;
        }
    }
}
void casezero(struct node *ptr,struct node *temp)
{
    if(ptr == NULL)
    {
        root = NULL;
    }
    else if(temp == ptr->rchild)
    {
        ptr->rchild = NULL;
    }
    else
    {
        ptr->lchild = NULL;
    }
    free(temp);
}
void caseone(struct node *ptr,struct node *temp)
{
    struct node *child;
    if(temp->lchild != NULL)
    {
        child = temp->lchild;
    }
    else
```

```
    {
        child = temp->rchild;
    }
    if(ptr == NULL)
    {
        root = child;
    }
    else if(temp == ptr->lchild)
    {
        ptr->lchild = child;
    }
    else
    {
        ptr->rchild = child;
    }
    free(temp);
}
void casetwo(struct node *ptr,struct node *temp)
{
    struct node *temp1,*ptr1;
    int temp2;
    temp1 = temp->rchild;
    ptr1 = temp;
    while(temp1->lchild != NULL)
    {
        ptr1 = temp1;
        temp1 = temp1->lchild;
    }
    temp2 = temp->info;
    temp->info = temp1->info;
    if(temp1->rchild == NULL)
    {
        printf("The node taking the place of %d is %d\n",temp2,temp1->info);
        casezero(ptr1,temp1);
    }
    else
    {
        printf("The node taking the place of %d is %d\n",temp2,temp1->info);
        caseone(ptr1,temp1);
    }
}
void delete()
{
    struct node *temp,*ptr = NULL;
    int item;
    if(root == NULL)
    {
        printf("\nTREE IS EMPTY\n");
    }
    else
    {
        temp = root;
        printf("Enter node to delete : ");
        scanf("%d",&item);
```

```
        while(temp != NULL)
        {
            if(temp->info == item)
            {
                break;
            }
            ptr = temp;
            if(temp->info > item)
            {
                temp = temp->lchild;
            }
            else
            {
                temp = temp->rchild;
            }
        }
        if(temp == NULL)
        {
            printf("Node is not present\n");
        }
        else if((temp->lchild == NULL)&&(temp->rchild == NULL))
        {
            printf("Deleted item is %d\n",temp->info);
            casezero(ptr,temp);
        }
        else if((temp->lchild != NULL)&&(temp->rchild != NULL))
        {
            printf("Deleted item is %d\n",temp->info);
            casetwo(ptr,temp);
        }
        else
        {
            printf("Deleted item is %d\n",temp->info);
            caseone(ptr,temp);
        }
    }
}
void search()
{
    struct node *temp;
    int item;
    if(root == NULL)
    {
        printf("TREE IS EMPTY \n");
    }
    else
    {
        temp = root;
        printf("Enter the item to be searched : ");
        scanf("%d",&item);
        while(temp != NULL)
        {
            if(temp->info == item)
            {
```

```
                printf("%d is present\n",temp->info);
                break;
            }
            else if(temp->info > item)
            {
                temp = temp->lchild;
            }
            else
            {
                temp = temp->rchild;
            }
        }
        if(temp == NULL)
        {
            printf("%d is not present\n",item);
        }
    }
}
void smallest()
{
    struct node *temp,*temp1;
    if(root == NULL)
    {
        printf("TREE IS EMPTY\n");
    }
    else
    {
        temp = root;
        while(temp != NULL)
        {
            temp1 = temp;
            temp = temp->lchild;
        }
        printf("The smallest element in the binary tree is %d\n",temp1->info);
    }
}
void largest()
{
    struct node *temp,*temp1;
    if(root == NULL)
    {
        printf("TREE IS EMPTY\n");
    }
    else
    {
        temp = root;
        while(temp != NULL)
        {
            temp1 = temp;
            temp = temp->rchild;
        }
        printf("The largest element in the binary tree is %d\n",temp1->info);
    }
}
```

```
main()
{
    int choice;
    while(1)
    {
        printf("\nWELCOME TO BINARY SEARCH TREE : \n");
        printf("1.CREATE\n2.PREORDER TRAVERSAL\n3.INORDER TRAVERSAL\n4.POSTORDER
TRAVERSAL\n5.INSERT\n6.SEARCHING AN ELEMENT\n7.SMALLEST ELEMENT\n8.LARGEST
ELEMENT\n9.DELETING AN ELEMENT\n10.EXIT\n");
        printf("Enter your choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1 : create();
                 break;
            case 2 : if(root == NULL)
                 {
                     printf("\nTREE IS EMPTY");
                 }
                 else
                 {
                     printf("\nTHE TREE IN PREORDER TRAVERSAL IS :\n");
                     preorder(root);
                 }
                 printf("\n");
                 break;
            case 3 : if(root == NULL)
                 {
                     printf("\nTREE IS EMPTY");
                 }
                 else
                 {
                     printf("\nTHE TREE IN INORDER TRAVERSAL IS : \n");
                     inorder(root);
                 }
                 printf("\n");
                 break;
            case 4 : if(root == NULL)
                 {
                     printf("\nTREE IS EMPTY");
                 }
                 else
                 {
                     printf("\nTHE TREE IN POSTORDER TRAVERSAL IS : \n");
                     postorder(root);
                 }
                 printf("\n");
                 break;
            case 5 : insert();
                 break;
            case 6 : search();
                 break;
            case 7 : smallest();
                 break;
```

```
            case 8 : largest();
                 break;
            case 9 : delete();
                 break;
            case 10 : exit(0);
                  break;
            default : printf("Wrong Choice\n");
        }
    }
}
```

//polynomial addition

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int coef;
    int pow;
    struct node *next;
};
struct node *p1 = NULL;
struct node *p2 = NULL;
struct node *p3 = NULL;
void insert();
void compare();
void display();
int size1,size2;
main()
{
    insert();
    compare();
    display();
}
void insert()
{
    struct node *newnode,*temp;
    int i,coeff,pow;
    printf("Enter the number of elements of the 1st polynomial : ");
    scanf("%d",&size1);
    for(i=1;i<=size1;i++)
    {
        newnode = (struct node *)malloc(sizeof(struct node));
        printf("Enter the coefficient : ");
        scanf("%d",&coeff);
        printf("Enter the power of x : ");
        scanf("%d",&pow);
        newnode->coef = coeff;
        newnode->pow = pow;
        newnode->next = NULL;
        if(p1 == NULL)
        {
            p1 = newnode;
```

```
        }
        else
        {
            temp = p1;
            while(temp->next != NULL)
            {
                temp = temp->next;
            }
            temp->next = newnode;
        }
    }
    printf("Enter the number of elements of the 2nd polyomial : ");
    scanf("%d",&size2);
    for(i=0;i<size2;i++)
    {
        newnode = (struct node *)malloc(sizeof(struct node));
        printf("Enter the coefficient : ");
        scanf("%d",&coeff);
        printf("Enter the power of x : ");
        scanf("%d",&pow);
        newnode->coef = coeff;
        newnode->pow = pow;
        newnode->next = NULL;
        if(p2 == NULL)
        {
            p2 = newnode;
        }
        else
        {
            temp = p2;
            while(temp->next!=NULL)
            {
                temp = temp->next;
            }
            temp->next = newnode;
        }
    }
}
void compare()
{
    int i,coeff,pow;
    struct node *temp,*temp1,*temp2,*newnode;
    temp = p1;
    temp1 = p2;
    if((p1 == NULL)&&(p2 == NULL))
    {
        printf("List is empty\n");
    }
    else
    {
        while((temp!=NULL)||(temp1!=NULL))
        {
            if(temp->pow == temp1->pow)
            {
```

```
        coeff = temp->coef + temp1->coef;
        pow = temp->pow;
        newnode = (struct node *)malloc(sizeof(struct node));
        newnode->coef = coeff;
        newnode->pow = pow;
        newnode->next = NULL;
        if(p3 == NULL)
        {
            p3 = newnode;
        }
        else
        {
            temp2 = p3;
            while(temp2->next != NULL)
            {
                temp2 = temp2->next;
            }
            temp2->next = newnode;
        }
        temp = temp->next;
        temp1 = temp1->next;
    }
    else if((temp->pow)>(temp1->pow))
    {
        coeff = temp->coef;
        pow = temp->pow;
        newnode = (struct node *)malloc(sizeof(struct node));
        newnode->coef = coeff;
        newnode->pow = pow;
        newnode->next = NULL;
        if(p3 == NULL)
        {
            p3 = newnode;
        }
        else
        {
            temp2 = p3;
            while(temp2->next != NULL)
            {
                temp2 = temp2->next;
            }
            temp2->next = newnode;
        }
        temp = temp->next;
    }
    else
    {
        coeff = temp1->coef;
        pow = temp1->pow;
        newnode = (struct node*)malloc(sizeof(struct node));
        newnode->coef = coeff;
        newnode->pow = pow;
        newnode->next = NULL;
        if(p3 == NULL)
```

```
                {
                    p3 = newnode;
                }
                else
                {
                    temp2 = p3;
                    while(temp2->next!= NULL)
                    {
                        temp2 = temp2->next;
                    }
                    temp2->next = newnode;
                }
                temp1 = temp1->next;
            }
        }
    }
}
void display()
{
    struct node *temp;
    if(p3 == NULL)
    {
        printf("List is empty\n");
    }
    else
    {
        temp = p3;
        printf("%dx^%d",temp->coef,temp->pow);
        temp = temp->next;
        while(temp!=NULL)
        {
            if(temp->coef < 0)
            {
                printf("%dx^%d",temp->coef,temp->pow);
            }
            else
            {
                printf("+%dx^%d",temp->coef,temp->pow);
            }
            temp = temp->next;
        }
        printf("\n");
    }
}
```

```
//postfix
#include<stdio.h>
int priority(char);
void push(char);
char pop();
void display();
void write(char);
```

```c
char postfix[60],stack[10];
int top=-1,index=-1;
main()
{
    int i;
    char str[40],item,x;
    printf("Enter the infix expression : ");
    scanf(" %[^\n]s",str);
    printf("The infix expression is : \n%s \n",str);
    for(i=0;str[i]!='\0';i++)
    {
        item = str[i];
        if (((item >= 65)&&(item <= 90))||((item >= 97)&&(item <= 122)))
        {
            write(item);
        }
        else if (item == '(')
        {
            push(item);
        }
        else if(item == ')')
        {
            x = pop();
            while(x!='(')
            {
                write(x);
                x = pop();
            }
        }
        else
        {
            x = pop();
            printf("%c \n",x);
            while (priority(x)>=priority(item))
            {
                write(x);
                x = pop();
            }
            push(x);
            push(item);
        }
    }
    while(top != -1)
    {
        x = pop();
        write(x);
    }
    display();
}
void push(char item)
{
    top++;
    stack[top]=item;
}
```

```
char pop()
{
    char item = stack[top];
    top--;
    return item;
}
int priority(char item)
{
    if(item == '^')
    {
        return 3;
    }
    else if((item == '+')||(item == '-'))
    {
        return 1;
    }
    else if((item == '*')||(item == '/')||(item == '%'))
    {
        return 2;
    }
    else
    {
        return 0;
    }
}
void write(char item)
{
    index++;
    postfix[index] = item;
}
void display()
{
    printf("The postfix expression is : \n");
    printf("%s \n",postfix);
}
```