**TW1**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<string.h>**

**void printUnion(char arr1[][15],char arr2[][15],int m,int n)**

**{**

**FILE \*f3;**

**f3=fopen("C:/Users/sahil/Desktop/third.txt","w");**

**if(f3==NULL){**

**printf("File opening error");**

**exit(1);**

**}**

**int i=0,j=0;**

**while (i<m && j<n)**

**{**

**if(strcmp(arr1[i],arr2[j])<0){**

**printf("\n%s",arr1[i]);**

**fprintf(f3,"\n%s",arr1[i++]);**

**}**

**else if(strcmp(arr2[j],arr1[i])<0){**

**printf("\n%s",arr2[j]);**

**fprintf(f3,"\n%s",arr1[j++]);**

**}else{**

**printf("\n%s",arr2[j]);**

**fprintf(f3,"\n%s",arr2[j++]);**

**i++;**

**}**

**}**

**while (i<m)**

**{**

**printf("\n%s",arr1[i]);**

**fprintf(f3,"\n%s",arr1[i++]);**

**}**

**while (j<n)**

**{**

**printf("\n%s",arr2[j]);**

**fprintf(f3,"\n%s",arr2[j++]);**

**}**

**}**

**void printIntersection(char arr1[][15],char arr2[][15],int m,int n){**

**for (int i = 0; i < n; i++)**

**{**

**for(int j=0;j<m;j++){**

**if(strcmp(arr1[i],arr2[j])==0)**

**{**

**printf("\n%s",arr1[i]);**

**}**

**}**

**}**

**}**

**int main()**

**{**

**FILE \*f1,\*f2;**

**char one[15][15],two[15][15];**

**int m=0,n=0;**

**f1=fopen("C:/Users/sahil/Desktop/first.txt","r");**

**f2=fopen("C:/Users/sahil/Desktop/second.txt","r");**

**printf("File 1\n");**

**while(!feof(f1)){**

**fscanf(f1,"%s",one[m++]);**

**puts(one[m-1]);**

**}**

**printf("File 2\n");**

**while(!feof(f2)){**

**fscanf(f2,"%s",two[n++]);**

**puts(two[n-1]);**

**}**

**printf("UNIQUE USN FROM BOTH FILES\n");**

**printUnion(one,two,m,n);**

**printf("\nCOMMON USN FROM BOTH FILES\n");**

**printIntersection(one,two,m,n);**

**fclose(f1);**

**fclose(f2);**

**return 0;**

**}**

**Tw2**

**#include <stdio.h>**

**#include <stdlib.h>**

**#include <ctype.h>**

**#define SIZE 20**

**int stack[SIZE], top;**

**void push(int);**

**int pop();**

**void disp();**

**void balanced(char[]);**

**int prec(char);**

**int main()**

**{**

**top=-1;**

**char exp[20];**

**int i, x;**

**printf("\nEnter the infix expression: ");**

**gets(exp);**

**balanced(exp);**

**printf("\nThe postfix expression is: ");**

**for(i=0; exp[i]!='\0'; i++)**

**{**

**if(isalnum(exp[i]))**

**printf("%c", exp[i]);**

**else**

**if(exp[i] == '(')**

**push(exp[i]);**

**else**

**if(exp[i] == ')')**

**{**

**while((x=pop()) != '(')**

**printf("%c", x);**

**}**

**else**

**{**

**while(prec(stack[top]) >= prec(exp[i]))**

**printf("%c", pop());**

**push(exp[i]);**

**}**

**}**

**while(top!=-1)**

**printf("%c", pop());**

**return 0;**

**}**

**void push(int item)**

**{**

**if(top==SIZE-1)**

**{**

**printf("\nStack Overflow");**

**return;**

**}**

**stack[++top]=item;**

**}**

**int pop()**

**{**

**if(top==-1)**

**{**

**printf("\nStack underflow");**

**return -1;**

**}**

**return stack[top--];**

**}**

**void disp()**

**{**

**if(top==-1)**

**{**

**printf("\nStack is empty");**

**return;**

**}**

**printf("\nThe stack contents are: ");**

**for(int i=0; i<=top; i++)**

**printf("%d  ", stack[i]);**

**}**

**void balanced(char exp[])**

**{**

**int i, count=0;**

**for(i=0; exp[i]!='\0'; i++)**

**{**

**if(exp[i] == '(')**

**count++;**

**else**

**if(exp[i] == ')')**

**count--;**

**}**

**if(count == 0)**

**{**

**printf("\nBalanced expression");**

**return;**

**}**

**printf("\nUnbalanced expression");**

**exit(1);**

**}**

**int prec(char x)**

**{**

**if(x=='(')**

**return 0;**

**else**

**if(x=='+' || x=='-')**

**return 1;**

**else**

**if(x=='\*' || x=='/')**

**return 2;**

**else**

**return(-1);**

**}**

**Tw3**

**#include <stdio.h>**

**#include <stdlib.h>**

**#define MAX 10**

**int stack[MAX], top;**

**void push(int item);**

**int pop();**

**int postfixeval(char []);**

**int main()**

**{**

**char exp[20];**

**top=-1;**

**printf("\nEnter a valid postfix expression: ");**

**gets(exp);**

**printf("\nEvaluated result is: %d", postfixeval(exp));**

**return 0;**

**}**

**void push(int item)**

**{**

**if(top==MAX-1)**

**{**

**printf("\nStack Overflow");**

**return;**

**}**

**stack[++top]=item;**

**}**

**int pop()**

**{**

**if(top==-1)**

**{**

**printf("\nStack Underflow");**

**return(-1);**

**}**

**return(stack[top--]);**

**}**

**int postfixeval(char exp[])**

**{**

**int i, opd1, opd2;**

**for(i=0; exp[i]!='\0'; i++)**

**{**

**if(exp[i]>='0' && exp[i]<='9')**

**push(exp[i]-'0');**

**else**

**{**

**opd2 = pop();**

**opd1 = pop();**

**switch(exp[i])**

**{**

**case '+': push(opd1+opd2); break;**

**case '-': push(opd1-opd2); break;**

**case '\*': push(opd1\*opd2); break;**

**case '/': if(opd2==0)**

**{**

**printf("\nDivide by zero error");**

**exit(1);**

**}**

**push(opd1/opd2);**

**}**

**}**

**}**

**return(pop());**

**}**

**Tw4**

**#include<stdio.h>**

**#include<string.h>**

**#include<stdlib.h>**

**#define MAX 5**

**int f=0,r=-1;**

**struct queue{**

**int id;**

**char msg[20];**

**}q[MAX];**

**void insert()**

**{**

**int x;**

**char txt[20];**

**if(r==MAX-1){**

**printf("Queue is full");**

**return;**

**}**

**printf("Enter the id and message:");**

**scanf("%d",&x);**

**gets(txt);**

**r++;**

**q[r].id=x;**

**strcpy(q[r].msg,txt);**

**printf("Message is inserted\n");**

**}**

**void delete()**

**{**

**if(f>r)**

**{**

**printf("Queue is empty\n");**

**f=0,r=MAX-1;**

**return;**

**}**

**printf("%d is deleted\n",q[f++].id);**

**}**

**void display(){**

**int i;**

**if(f>r){**

**printf("Queue is empty");**

**return;**

**}**

**for(i=f;i<=r;i++){**

**printf("%d\t %s\n",q[i].id,q[i].msg);**

**}**

**}**

**int main()**

**{**

**int c;**

**printf("Enter\n1.Insert\n2.Delete\n3.Display\n4.Exit\n");**

**while (1)**

**{**

**printf("Enter your choice:");**

**scanf("%d",&c);**

**switch (c)**

**{**

**case 1:insert();**

**break;**

**case 2:delete();**

**break;**

**case 3:display();**

**break;**

**case 4:exit(0);**

**default:printf("invalid choice\n");**

**}**

**}**

**}**

**Tw5**

**#include<stdio.h>**

**#include<stdlib.h>**

**#include<string.h>**

**#define SIZE 10**

**typedef struct**

**{**

**int custid;**

**char custname[30];**

**int custphno;**

**int empty;**

**}CUST;**

**int HashFn(int key)**

**{**

**return (key % SIZE);**

**}**

**int Search(int key, CUST custs[])**

**{**

**int count, pos;**

**pos = HashFn(key);**

**for(count = 1; count <= SIZE ; count++)**

**{**

**if(custs[pos].empty == 1)**

**return -1;**

**if(custs[pos].custid == key)**

**return pos;**

**pos = (pos + 1) % SIZE;**

**}**

**return -1;**

**}**

**void InsHT\_LP(CUST cust, CUST custs[])**

**{**

**int count, pos, flag=0;**

**int key = cust.custid;**

**pos = HashFn(key);**

**for(count = 1; count <= SIZE; count++)**

**{**

**if(custs[pos].empty == 1)**

**{**

**if(flag==1)**

**printf("\nCollision resolved, new pos = %d", pos);**

**custs[pos].custid = cust.custid;**

**strcpy(custs[pos].custname,cust.custname);**

**custs[pos].custphno = cust.custphno;**

**custs[pos].empty = -1;**

**printf("\nRecord Inserted into Hash Table\n");**

**return;**

**}**

**if(custs[pos].empty==-1)**

**{**

**printf("\nCollision Detected\n");**

**flag=1;**

**}**

**pos = (pos + 1) % SIZE;**

**}**

**printf("\nHash Table is full\n");**

**}**

**void Display(CUST custs[])**

**{**

**int count;**

**printf("\nHash Table");**

**for(count = 0; count < SIZE; count++)**

**{**

**printf("\n[%d]:\t", count);**

**if(custs[count].empty == -1)**

**printf("\nCustomer - ID: %d Name: %s Phone: %d",custs[count].custid, custs[count].custname, custs[count].custphno);**

**else**

**printf("No Hash Entry\n");**

**}**

**}**

**int main()**

**{**

**int count, key, option;**

**CUST custs[SIZE], cust;**

**for(count = 0; count < SIZE ; count++)**

**custs[count].empty=1;**

**while(1)**

**{**

**printf("\n\n1. Insert a Record\n");**

**printf("2. Search a Record\n");**

**printf("3. Display All Records\n");**

**printf("4. Exit\n");**

**printf("\n Enter Your Option:");**

**scanf("%d", &option);**

**switch(option)**

**{**

**case 1: printf("\nEnter Customer id, name, ph:");**

**scanf("%d%s%d", &cust.custid,cust.custname,&cust.custphno);**

**InsHT\_LP(cust, custs);**

**break;**

**case 2: printf("\nEnter the Key to Search:");**

**scanf("%d", &key);**

**count = Search(key, custs);**

**if(count == -1)**

**printf("\nRecord Not Found\n");**

**else**

**printf("\nRecord Found at Index pos:%d\n", count);**

**break;**

**case 3: Display(custs);**

**break;**

**case 4: exit(1);**

**}**

**}**

**return 0;**

**}**

**Tw6**

**#include <stdio.h>**

**#include <stdlib.h>**

**typedef struct node**

**{**

**int data;**

**struct node \*next;**

**}NODE;**

**NODE\* add(NODE\*, int);**

**void disp(NODE\*);**

**int search(NODE\*, int);**

**int main()**

**{**

**NODE \*head=NULL;**

**int opt, item;**

**while(1)**

**{**

**printf("\n1: Add item    2: Disp Warehouse    ");**

**printf("3: Search item   4: exit");**

**printf("\nEnter your option: ");**

**scanf("%d", &opt);**

**switch(opt)**

**{**

**case 1: printf("\nEnter item to add to warehouse: ");**

**scanf("%d", &item);**

**head=add(head, item);**

**break;**

**case 2: disp(head);   break;**

**case 3: printf("\nEnter the item to search: ");**

**scanf("%d", &item);**

**if(search(head, item))**

**printf("\nItem %d is present in the warehouse", item);**

**else**

**printf("\nItem %d is NOT present in the warehouse", item);**

**break;**

**case 4: exit(0);**

**}**

**}**

**return 0;**

**}**

**NODE\* add(NODE\* head, int item)**

**{**

**NODE \*prev, \*curr;**

**NODE \*newnode=(NODE\*)malloc(sizeof(NODE));**

**if(newnode==NULL)**

**{**

**printf("\nMalloc failure");**

**exit(1);**

**}**

**newnode->data=item;**

**newnode->next=NULL;**

**// Case i - List is empty**

**if(head==NULL)**

**head=newnode;**

**else // Case ii - adding the smallest item**

**if(item < head->data)**

**{**

**newnode->next = head;**

**head = newnode;**

**}**

**else // Case iii**

**{**

**prev=head;**

**curr=head->next;**

**while(curr && item > curr->data)**

**{**

**prev=prev->next;**

**curr=curr->next;**

**}//end of while**

**newnode->next=curr;**

**prev->next=newnode;**

**}//end of else**

**return head;**

**}**

**void disp(NODE \*head)**

**{**

**if(head==NULL)**

**{**

**printf("\nWarehouse is empty");**

**return;**

**}**

**printf("\nThe warehouse items are: ");**

**while(head)**

**{**

**printf("%d    ", head->data);**

**head=head->next;**

**}**

**}**

**int search(NODE\* head, int item)**

**{**

**if(head==NULL)**

**{**

**printf("\nWarehouse is empty");**

**return 0;**

**}**

**while(head && (item > head->data))**

**head=head->next;**

**if(head==NULL)**

**return 0;**

**if(item==head->data)**

**return 1;**

**else**

**return 0;**

**}**

**Tw7**

**#include<stdio.h>**

**#include<stdlib.h>**

**struct poly**

**{**

**int coeff;**

**int expo;**

**};**

**int readPoly(struct poly []);**

**int addPoly(struct poly [],struct poly [],int ,**

**int ,struct poly []);**

**void displayPoly( struct poly [],int terms);**

**int main()**

**{**

**int t1,t2,t3;**

**struct poly p1[10],p2[10],p3[10];**

**t1=readPoly(p1);**

**printf("\nFirst Polynomial:");**

**displayPoly(p1,t1);**

**t2=readPoly(p2);**

**printf("\nSecond Polynomial:");**

**displayPoly(p2,t2);**

**t3=addPoly(p1,p2,t1,t2,p3);**

**printf("\nResultant polynomial after addition:");**

**displayPoly(p3,t3);**

**printf("\n");**

**return 0;**

**}**

**int readPoly(struct poly p[10])**

**{**

**int t1,i;**

**printf("\nEnter the total number of terms in the polynomial\n");**

**scanf("%d",&t1);**

**printf("Enter the coefficient and exponent in descending order\n");**

**for(i=0;i<t1;i++){**

**printf("Enter the Coefficient(%d): ",i+1);**

**scanf("%d",&p[i].coeff);**

**printf("Enter the exponent(%d): ",i+1);**

**scanf("%d",&p[i].expo);**

**}**

**return(t1);**

**}**

**int addPoly(struct poly p1[10],struct poly p2[10],int t1,int t2,struct poly p3[10])**

**{**

**int i,j,k;**

**i=j=k=0;**

**while(i<t1 && j<t2)**

**{**

**if(p1[i].expo==p2[j].expo)**

**{**

**p3[k].coeff=p1[i].coeff + p2[j].coeff;**

**p3[k].expo=p1[i].expo;**

**i++;**

**j++;**

**k++;**

**}**

**else if(p1[i].expo>p2[j].expo)**

**{**

**p3[k].coeff=p1[i].coeff;**

**p3[k].expo=p1[i].expo;**

**i++;**

**k++;**

**}**

**else**

**{**

**p3[k].coeff=p2[j].coeff;**

**p3[k].expo=p2[j].expo;**

**j++;**

**k++;**

**}**

**}**

**while(i<t1)**

**{**

**p3[k].coeff=p1[i].coeff;**

**p3[k].expo=p1[i].expo;**

**i++;**

**k++;**

**}**

**while(j<t2)**

**{**

**p3[k].coeff=p2[j].coeff;**

**p3[k].expo=p2[j].expo;**

**j++;**

**k++;**

**}**

**return(k);**

**}**

**void displayPoly(struct poly p[10],int term)**

**{**

**int k;**

**for(k=0;k<term-1;k++)**

**printf("%d(x^%d)+",p[k].coeff,p[k].expo);**

**printf("%d(x^%d)",p[term-1].coeff,p[term-1].expo);**

**}**

**Tw8**

**#include <stdio.h>**

**#include <stdlib.h>**

**typedef struct node**

**{**

**int data;**

**struct node \*left, \*right;**

**}NODE;**

**NODE\* newNode(int item)**

**{**

**NODE \* tmp=(NODE \*)malloc(sizeof(NODE));**

**tmp->data=item;**

**tmp->left=tmp->right=NULL;**

**return tmp;**

**}**

**NODE\* addBST(NODE\* root, int key)**

**{**

**if(root==NULL)**

**return newNode(key);**

**if(key<root->data)**

**root->left=addBST(root->left, key);**

**else**

**root->right=addBST(root->right, key);**

**return root;**

**}**

**int leaf(NODE\* root)**

**{**

**if(root==NULL)**

**return 0;**

**if(root->left==NULL && root->right==NULL)**

**return 1;**

**return leaf(root->left)+leaf(root->right);**

**}**

**int nonleaf(NODE\* root)**

**{**

**if(root==NULL)**

**return 0;**

**if(root->left==NULL && root->right==NULL)**

**return 0;**

**return 1+nonleaf(root->left)+nonleaf(root->right);**

**}**

**int two(NODE\* root)**

**{**

**if(root==NULL)**

**return 0;**

**if(root->left!=NULL && root->right!=NULL)**

**return 1+two(root->left)+two(root->right);**

**if(root->left==NULL)**

**return two(root->right);**

**if(root->right==NULL)**

**return two(root->left);**

**return 0;**

**}**

**int total(NODE\* root)**

**{**

**if(root==NULL)**

**return 0;**

**return 1+total(root->left)+total(root->right);**

**}**

**int main()**

**{**

**int i,n,key;**

**NODE \*root=NULL;**

**printf("\nEnter n: ");**

**scanf("%d", &n);**

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

**{**

**printf("\nEnter int %d: ", i+1);**

**scanf("%d", &key);**

**root=addBST(root, key);**

**}**

**printf("\nNo of leaf nodes: %d", leaf(root));**

**printf("\nNo of non-leaf nodes: %d", nonleaf(root));**

**printf("\nNo of nodes with degree two: %d", two(root));**

**printf("\nTotal no of nodes: %d", total(root));**

**return 0;**

**}**

**Tw10**

**#include <stdio.h>**

**#include <stdlib.h>**

**typedef struct node**

**{**

**int data;**

**struct node \*next, \*prev;**

**}NODE;**

**//Prototypes**

**NODE\* addFront(NODE\*);**

**NODE\* addRear(NODE\*);**

**void disp(NODE\*);**

**void search(NODE\*);**

**//Driver**

**int main()**

**{**

**//Create an empty DLL**

**NODE \*head=NULL;**

**int opt;**

**while(1)**

**{**

**printf("\n1: FrontAdd\t2: RearAdd\t3: Disp\t");**

**printf("4: Search\t5: Exit");**

**printf("\nEnter an option: ");**

**scanf("%d", &opt);**

**switch(opt)**

**{**

**case 1: head=addFront(head);   break;**

**case 2: head=addRear(head);   break;**

**case 3: disp(head);   break;**

**case 4: search(head);   break;**

**case 5: exit(0);**

**}**

**}**

**return 0;**

**}**

**//Adding a node at the front of the DLL**

**NODE\* addFront(NODE \*head)**

**{**

**int data;**

**NODE \*tmp=(NODE \*)malloc(sizeof(NODE));**

**if(tmp==NULL)**

**{**

**printf("\nMalloc failure");**

**exit(1);**

**}**

**printf("\nEnter data to add: ");**

**scanf("%d", &data);**

**tmp->data=data;**

**tmp->prev=tmp->next=NULL;**

**if(head==NULL) //Empty List**

**head=tmp;**

**else   //List has atleast one datanode**

**{**

**tmp->next=head;**

**head->prev=tmp;**

**head=tmp;**

**}**

**return(head);**

**}**

**void disp(NODE \*head)**

**{**

**if(head==NULL)**

**{**

**printf("\nList is empty");**

**return;**

**}**

**printf("\nThe List contents are: ");**

**while(head)**

**{**

**printf("%d    ", head->data);**

**head=head->next;**

**}**

**}**

**//Adding a node at the end of the DLL**

**NODE\* addRear(NODE \*head)**

**{**

**int data;**

**NODE \*trav;**

**NODE \*tmp=(NODE \*)malloc(sizeof(NODE));**

**if(tmp==NULL)**

**{**

**printf("\nMalloc failure");**

**exit(1);**

**}**

**printf("\nEnter data to add: ");**

**scanf("%d", &data);**

**tmp->data=data;**

**tmp->prev=tmp->next=NULL;**

**if(head==NULL) //Empty List**

**head=tmp;**

**else   //List has atleast one datanode**

**{**

**trav=head;**

**while(trav->next)**

**trav=trav->next;**

**trav->next=tmp;**

**tmp->prev=trav;**

**}**

**return(head);**

**}**

**//Search data in DLL**

**void search(NODE \*head)**

**{**

**int item, count=0;**

**if(head==NULL)**

**{**

**printf("\nList is empty");**

**return;**

**}**

**printf("\nEnter data to search: ");**

**scanf("%d", &item);**

**while(head)**

**{**

**if(item==head->data)**

**{**

**printf("\nData %d present at position %d", item, count+1);**

**return;**

**}**

**else**

**{**

**count++;**

**head=head->next;**

**}**

**}**

**printf("\nData %d not present", item);**

**}**