****

**Department of Artificial Intelligence and Data Science/ Artificial Intelligence and Machine Learning**

**QUESTION BANK FOR III SEMESTER (Term: November 2022 – January 2023)**

**Data Structures Laboratory (AD33/AI33)**

**I.A. Marks: 50 Exam Hours: 03**

**Credits: 0:0:1 Exam Marks: 50**

1. Write a C program to implement iterative and recursive binary search algorithms. Define and use a macro to compare two integers in your program.

->#include<stdio.h>

#define compare(a,b) ((a)>(b) ? (1) : ((a)==(b) ? (0) : (-1)))

int bin\_search(int a[],int key,int n)

{

    int mid;

    int low = 0;

    int high = n-1;

    while(low<=high)

    {

        mid = (low+high)/2;

        switch(compare(a[mid],key))

        {

                case -1 : low = mid+1;

                  break;

            case 1:

                        high = mid-1;

                   break;

                case 0 :

                        return mid;

        }

    }

    return -1;

}

int main()

{

        int n, a[20], i, key, result;

    printf("enter the number of elements of array\n");

        scanf("%d",&n);

        printf("enter the array elements\n");

        for(i=0;i<n;i++)

            scanf("%d",&a[i]);

    printf("enter the key to be searched\n");

    scanf("%d", &key);

    result = bin\_search(a,key,n);

    if(result == -1)

        printf("%d not found in the array\n", key);

    else

        printf("%d found at location %d\n",key,result+1);

}

RECURSIVE

->#include<stdio.h>

#define compare(a,b) ((a)>(b) ? (1) : ((a)==(b) ? (0) : (-1)))

int bin\_search(int a[],int key,int left,int right)

{

    int middle;

    if(left<=right)

    {

        middle = (left+right)/2;

        switch(compare(a[middle],key))

        {

                case -1 :

                        return bin\_search(a,key,middle+1,right);

                case 0 :

                        return middle;

                case 1:

                        return bin\_search(a,key,left,middle-1);

        }

    }

    else

        return -1;

}

int main()

{

        int n, a[20], i, key, result;

    printf("enter the number of elements of array\n");

        scanf("%d",&n);

        printf("enter the array elements\n");

        for(i=0;i<n;i++)

            scanf("%d",&a[i]);

    printf("enter the key to be searched\n");

    scanf("%d", &key);

    result = bin\_search(a,key,0,n-1);

    if(result == -1)

        printf("%d not found in the array\n", key);

    else

        printf("%d found at location %d\n",key,result+1);

}

2.Write a C program to find the fast transpose of a sparse matrix.

#include<stdio.h>

typedef struct

{

        int r,c,v;

} term;

void transpose(term a[],term t[])

{

    int rt[10],sp[10];

    int i,j,numcols=a[0].c,numterms=a[0].v;

    t[0].r=numcols;

    t[0].v=numterms;

    t[0].c=a[0].r;

    if(numterms>0)

    {

        for(i=0;i<numcols;i++)

            rt[i]=0;

        for(i=1;i<=numterms;i++)

            rt[a[i].c]++;

        sp[0]=1;

        for(i=1;i<numcols;i++)

            sp[i]=sp[i-1]+rt[i-1];

        for(i=1;i<=numterms;i++)

        {

                j=sp[a[i].c]++;

                t[j].r=a[i].c;

                t[j].c=a[i].r;

                t[j].v=a[i].v;

        }

    }

}

int main()

{

    term a[10],t[10];

    int i;

    printf("\nEnter the number of rows and columns\n");

    scanf("%d%d",&a[0].r,&a[0].c);

    printf("\nEnter the number of values\n");

    scanf("%d",&a[0].v);

    for(i=1;i<=a[0].v;i++)

    {

        printf("\nEnter %dth row, column and element values\n",i);

        scanf("%d%d%d",&a[i].r,&a[i].c,&a[i].v);

    }

    printf("\nOriginal Matrix\n");

    for(i=1;i<=a[0].v;i++)

        printf("%d\t%d\t%d\n",a[i].r,a[i].c,a[i].v);

    transpose(a,t);

    printf("\nTranspose Matrix\n");

    for(i=1;i<=t[0].v;i++)

        printf("%d\t%d\t%d\n",t[i].r,t[i].c,t[i].v);

    return 0;

}

3.Write a C program to perform pattern matching using KMP Algorithm. (Print the failure function of a pattern and display whether match is found or not).

    #include<stdio.h>

    #include<string.h>

    int failure[20];

    void fail(char \*pat)

    {

        int i,j;

        int n=strlen(pat);

        failure[0]=-1;

        for(j=1;j<n;j++)

        {

            i=failure[j-1];

            while((pat[j]!=pat[i+1])&&(i>0))

                i=failure[i];

            if(pat[j]==pat[i+1])

                failure[j]=i+1;

            else

                failure[j]=-1;

        }

    }

    int match(char \*string, char \*pat)

    {

        int i=0,j=0;

        int lens=strlen(string);

        int lenp=strlen(pat);

        while(i<lens&&j<lenp)

        {

            if(string[i]==pat[j])

            {

                i++;

                j++;

            }

            else if(j==0)

                i++;

            else

                j=failure[j-1]+1;

        }

        return((j==lenp)?(i-lenp):-1);

    }

    int main()

    {

        int i;

        char str[30],pat[20];

        printf("\nEnter a string\n");

        scanf("%s",str);

        printf("\nEnter a substring\n");

        scanf("%s",pat);

        fail(str);

        i=match(str,pat);

        if(i==-1)

            printf("\nPattern %s Not found", pat);

        else

            printf("\nPattern %sFound at position %d",pat,i+1);

        return 0;

    }

4. Write a C program to implement a circular queue using dynamically allocated array and perform the following operations on it.

* 1. Insert an item (ii) Delete an item (iii) Display a circular queue

#include<stdio.h>

#include<stdlib.h>

int \*queue;

int front, rear, capacity;

int pop()

{

    front = (front+1)%capacity;

    return queue[front];

}

int push(int data)

{

    int \*newQueue, i;

    if( (rear+1)%capacity ==front%capacity )

    {

        printf("Extending Queue (new capacity=%d)\n", 2\*capacity);

        newQueue = (int\*)malloc(capacity \* 2 \* sizeof(int));

        for(i=1;front!=rear;i++)

        {

            newQueue[i] = pop();

        }

        rear = i-1;

        front = 0;

        capacity \*= 2;

        free(queue);

        queue = newQueue;

    }

    rear = (rear+1)%capacity;

    queue[rear] = data;

}

void display()

{

    printf("Elements:\n");

    if(front == rear)

    {

        printf("Queue Empty\n");

    }

    for(int i=(front+1)%capacity ; i!= (rear+1)%capacity ; i = (i+1)%capacity)

    {

        printf("%d\n", queue[i]);

    }

}

int main()

{

    front = 0;

    rear = 0;

    capacity = 2;

    queue = (int\*)malloc(capacity\* sizeof(int));

        int choice =4;

        int num;

        do{

        printf("1.push\n2.pop.\n3.display\n4.exit\n");

            scanf("%d",&choice);

            switch(choice)

            {

                case 1:

                scanf("%d",&num);

                push(num);

                break;

                case 2:

                if(front==rear)

                    printf("Queue Empty\n");

                else

                    printf("element removed is %d\n",pop());

                break;

                case 3:

                display();

                break;

                case 4:

                break;

                default : printf("Invalid Entry\n");

            }

            printf("\n");

        }while(choice!=4);

    return 0;

}

5.Write a C program to evaluate a given postfix expression using a stack.

->#include<stdio.h>

#include<string.h>

#include<ctype.h>

#define STACKSIZE 100

int stack[STACKSIZE];

int top=-1;

int pop()

{

    return stack[top--];

}

void push(int n)

{

    stack[++top] = n;

}

int result(int op1, int op2, char operator)

{

    switch(operator)

    {

        case '+':return op1+op2;

        case '-':return op1-op2;

        case '\*':return op1\*op2;

        case '/':return op1/op2;

        case '%':return op1%op2;

    }

}

int postfixEval(char \*str)

{

    int i;

    int op1, op2;

    for(i=0;i<strlen(str);i++)

    {

        if(isdigit(str[i]))

        {

            push(str[i]-'0');

        }

        else

        {

            op2=pop();

            op1=pop();

            push(result(op1, op2, str[i]));

        }

    }

    return pop();//since top of the stack has the answer

}

int main()

{

    char str[100];

    printf("Enter the Postfix Expression :\n");

    scanf("%s", str);

    printf("Result = %d\n", postfixEval(str));

    return 0;

}

6.Write a C program to implement multiple linked stacks (at least 5) and perform the following operations on them

(i) Push an item in i th stack (ii) Pop an item from i th stack (iii) Display i th stack

#include<stdio.h>

#include<stdlib.h>

#define MAX\_STACKS 3

typedef struct

{

    int key;

} ele;

typedef struct stack \*stackPtr;

typedef struct stack{

    ele data;

    stackPtr link;

} stack;

stackPtr top[MAX\_STACKS];

void push(int i, int item)

{

    stackPtr temp;

    temp=(stackPtr) malloc(sizeof(stack));

    temp->data.key = item;

    temp->link = top[i];

    top[i] = temp;

}

int pop(int i)

{

    stackPtr temp = top[i];

    int item;

    item = temp->data.key;

    top[i] = temp->link;

    free(temp);

    printf("Popped %d from stack %d\n", item, i);

}

void display()

{

    int i;

    stackPtr j;

    for(i=0;i<MAX\_STACKS;i++)

    {

        printf("Stack no.%d :\n",i+1);

        if(top[i] == NULL)

           printf("Stack Empty\n--------------------\n");

        else

        {

            for(j = top[i]; j != NULL ; j = j->link)

                printf("%d\t",j->data.key);

            printf("\n--------------------\n");

        }

    }

}

int main()

{

    int choice=4, i, j;

    ele x;

    for(i=0;i<MAX\_STACKS;i++)

        top[i] = NULL;

    while(1)

    {

        printf("1.push\n2.pop\n3.display\n4.exit\n");

        printf("Enter your choice\n");

        scanf("%d",&choice);

        switch(choice)

        {

            case 1:

                printf("Enter the stack number(0-%d) and element to be added\n",MAX\_STACKS-1);

                scanf("%d%d",&i ,&x.key);//x is the element to be pushed

                push(i,x.key);

                break;

            case 2:

                printf("Enter the queue number(0-%d)\n",MAX\_STACKS-1);

                scanf("%d",&i);

                if(top[i] == NULL)

                    printf("Queue Empty\n");

                else

                    pop(i);

                break;

            case 3:

                display();

                break;

            case 4:

                exit(0);

                break;

            default :

                printf("Invalid Choice");

        }

    }

    return 0;

}

7.Write a C program to implement multiple linked queues (at least 5) and perform the following operations on them.

(i) Add an item in i th queue (ii) Delete an item from i th queue (iii) Display i th queue

#include<stdio.h>

#include<stdlib.h>

#define MAXQUEUES 10

typedef struct node \*nodePtr;

typedef struct node

{

    int data;

    nodePtr link;

}node;

nodePtr front[MAXQUEUES];

nodePtr rear[MAXQUEUES];

void push(int i, int data)

{

    nodePtr newNode = (nodePtr)malloc(sizeof(node));

    newNode->data = data;

    newNode->link =NULL;

    if(front[i]==NULL)

        front[i] = newNode;

    else

        rear[i]->link = newNode;

    rear[i] = newNode;

}

void pop(int i)

{

    if(front[i])

    {

        nodePtr temp = front[i];

        printf("Popped : %d from Queue no.%d\n", front[i]->data, i);

        front[i] = front[i]->link;

        free(temp);

    }

    else

    {

        printf("Queue no.%d is EMPTY\n", i);

    }

}

void display(int i)

{

    printf("\nQueue no.%d\n", i);

    if(front[i])

    {

        nodePtr temp = front[i];

        for(; temp!=NULL; temp = temp->link)

            printf("%5d", temp->data);

    }

    else

    {

        printf("Queue %d Empty", i);

    }

    printf("\n");

}

int main()

{

    for(int i=0;i<MAXQUEUES; i++)

    {

        front[i] = NULL;

        rear[i] = NULL;

    }

    int choice, i, data;

    printf("MENU\n1.push\n2.pop\n3.display\n4.exit\n\n\n");

    do {

        printf("choice : ");

        scanf("%d", &choice);

        switch(choice)

        {

            case 1:

            printf("Queue no(0-9) : ");

            scanf("%d", &i);

            printf("Element : ");

            scanf("%d", &data);

            push(i, data);

            break;

            case 2:

            printf("Queue no(0-9) : ");

            scanf("%d", &i);

            pop(i);

            break;

            case 3:

            printf("Queue no(0-9) : ");

            scanf("%d", &i);

            display(i);

            break;

            case 4:

            printf("Exit\n");

            break;

            default:printf("Invalid\n");

        }

        printf("\n");

    } while(choice!=4);

    return 0;

}

8.Write a C program to add two polynomials represented as circular linked lists with header nodes.

Display both polynomials and the resultant polynomial after addition

#include<stdio.h>

#include<stdlib.h>

#define TRUE 1

#define FALSE 0

#define COMPARE(x,y)  (((x) < (y)) ? -1: ((x) == (y)) ? 0 : 1)

typedef struct polyNode\* polyPtr;

typedef struct polyNode

{

    int coef;

    int expon;

    polyPtr link;

} polyNode;

polyPtr headA, headB, headC;

void attach(int c, int e, polyPtr \*ptr)

{

    polyPtr temp;

    temp=(polyPtr)malloc(sizeof(polyNode));

    temp->coef = c;

    temp->expon = e;

    (\*ptr)->link= temp;

    \*ptr = temp;

}

void cpadd(polyPtr a, polyPtr b)

{

    polyPtr startA, lastC;

    int sum, done=FALSE;

    startA = a;

    a= a->link;

    b= b->link;

    headC= (polyPtr) malloc(sizeof(polyNode));

    headC->expon = -1;

    lastC=headC;

    do

    {

        switch(COMPARE(a->expon, b->expon))

        {

            case -1: attach(b->coef, b->expon, &lastC);

                     b=b->link;

                     break;

            case 0: if(startA == a) done = TRUE;

                    else

                    {

                        sum= a->coef + b->coef;

                        if(sum) attach(sum, a->expon, &lastC);

                        a=a->link;

                        b=b->link;

                    }

                    break;

            case 1: attach(a->coef, a->expon, &lastC);

                    a=a->link;

                    break;

        }

    } while(!done);

    lastC->link = headC;

}

int main()

{

        polyPtr lastA, lastB, temp, start;

        int c, e, i, n;

        headA = (polyPtr) malloc(sizeof(polyNode));

        headB = (polyPtr) malloc(sizeof(polyNode));

        headA->expon = -1;

        headB->expon = -1;

        lastA=headA;

        lastB=headB;

        printf("How many terms in polynomial A\n");

        scanf("%d",&n);

        for(i=0;i<n;i++)

        {

            printf("enter the coef & exponent\n");

            scanf("%d%d", &c, &e);

            attach(c,e,&lastA);

        }

        lastA->link=headA;

        printf("How many terms in polynomial B\n");

        scanf("%d",&n);

        for(i=0;i<n;i++)

        {

            printf("enter the coef & exponent\n");

            scanf("%d%d", &c, &e);

            attach(c,e,&lastB);

        }

        lastB->link=headB;

        cpadd(headA,headB);

        printf("\nPolynomial A is:\n");

        for(start= headA->link ;start != headA; start=start->link)

            printf("%d \*x%d + ",start->coef, start->expon);

        printf("\nPolynomial B is:\n");

        for(start= headB->link ;start != headB; start=start->link)

            printf("%d \*x%d + ",start->coef, start->expon);

        printf("\nSum Polynomial is:\n");

        for(start= headC->link ;start != headC; start=start->link)

            printf("%d \*x%d + ",start->coef, start->expon);

        printf("\n\n");

    for(start= headA->link ;start != headA; start=start->link)

        free(start);

        free(headA);

    for(start= headB->link ;start != headB; start=start->link)

        free(start);

        free(headB);

    for(start= headC->link ;start != headC; start=start->link)

        free(start);

        free(headC);

    return 0;

}

9.Write a C program to implement a doubly linked circular list with a header node and perform the

following operations on it.

* 1. Insert a node (iii) Display a doubly linked circular list in forward direction
  2. Delete a node (iv) Display a doubly linked circular list in reverse direction

#include<stdio.h>

#include<stdlib.h>

typedef struct node \*nodePtr;

typedef struct node {

nodePtr llink;

int data;

nodePtr rlink;

}node;

nodePtr head;

void dinsert()

{

    int n;

    nodePtr temp;

    printf("Enter the info for the new node");

    scanf("%d", &n);

    temp=(nodePtr)malloc(sizeof(node));

    temp->data=n;

    temp->llink = head;

    temp->rlink = head->rlink;

    head->rlink-> llink = temp;

    head->rlink = temp;

}

void ddelete()

{

    nodePtr temp=head->rlink;

    if (head->rlink == head)

        printf("Deletion of head node not permitted.\n");

    else

    {

        head->rlink = temp->rlink;

        temp->rlink->llink = head;

        printf("removing node with data %d\n",temp->data);

        free(temp);

    }

}

void displayRight()

{

    nodePtr temp;

    if (head->rlink == head)

        printf("Empty list.\n");

    else

    {

        for(temp=head->rlink; temp->rlink != head; temp = temp->rlink)

            printf("%d\t", temp->data);

        printf("%d\t", temp->data);

        printf("\n\n");

    }

}

void displayLeft()

{

    nodePtr temp;

    if (head->llink == head)

        printf("Empty list.\n");

    else

    {

        for(temp=head->llink; temp->llink != head; temp = temp->llink)

            printf("%d\t", temp->data);

        printf("%d\t", temp->data);

        printf("\n\n");

    }

}

int main()

{

    unsigned int choice;

    head=(nodePtr)malloc(sizeof(node));

    head->rlink=head;

    head->llink=head;

    while(1)

    {

        printf("1:insert a node in DLL \n2:delete a node from DLL \n3:display the DLL forward\n4:display the DLL forward\n5:exit\n");

        scanf("%u", &choice);

        switch(choice)

        {

            case 1: dinsert();

                    break;

            case 2: ddelete();

                    break;

            case 3: displayRight();

                    break;

            case 4: displayLeft();

                    break;

            case 5: exit(0);

                    break;

            default: printf("Invalid choice... try again\n");

        }

    }

    return 0;

}