****

# **Department of Computer Science & Engineering**

**QUESTION BANK FOR III SEMESTER (Term: Sep-Dec 2020)**

**Data Structures Laboratory (CSL38)**

|  |  |
| --- | --- |
|  | Write a C program to find the fast transpose of a sparse matrix. |
|  | 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). |
|  | 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 |
|  | Write a C program to convert a given infix expression to a postfix expression using a stack. |
|  | Write a C program to evaluate a given postfix expression using a stack. |
|  | Write a C program to implement multiple linked stacks (at least 5) and perform the following operations on them   1. Push an item in ithstack (ii) Pop an item from ithstack(iii) Display ithstack |
|  | Write a C program to implement multiple linked queues (at least 5) and perform the following operations on them   1. Add an item in ithqueue (ii) Delete an item from ithqueue(iii)Display ithqueue |
|  | 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. |
|  | Write a C program to implement a doubly linked circular list with a header node and perform the following operations on it.  (i) Insert a node (iii) Display a doubly linked circular list in forward direction  (ii) Delete a node (iv)Display a doubly linked circular list in reverse direction |
|  | Write a C program to implement a max heap using an array and perform the following operations on it.  (i) Insert an item (ii) Delete an item (iii) Display a heap |
|  | Write a C program to implement a binary search tree using linked representation and perform the following operations on it.  (i) Insert an item (ii) Search an item (iii) Inorder Traversal |
|  | Write a C program to perform depth first search of a graph represented as an adjacency list. |

1. 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;

}

}

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);

}

void 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);

}

1. 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);

}

void main()

{

int i;

char str[30],sub[20];

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

scanf("%s",str);

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

scanf("%s",sub);

fail(sub);

i=match(str,sub);

if(i==-1)

printf("\nNot found");

else

printf("\nFound at position %d",i+1);

}

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

i)Insert an item (ii) Delete an item (iii) Display a circular queue

#include<stdio.h>

#include<stdlib.h>

#define MALLOC(x,size,type)(x=(type\*)malloc(size\*sizeof(type)))

typedef struct

{

int n;

}element;

int front=0, rear=0, capacity;

element \*queue;

void copy(element\* start, element\* end, element\* newQueue)

{

element\* j;

element\* i;

i=newQueue;

j=start;

for(; j<end; j++, i++)

{

\*i=\*j;

}

}

void queueFull()

{

element\* newQueue;

MALLOC(newQueue, capacity\*2, element);

int start=(front+1)%capacity;

if(start < 2) //either 1 or 0, 1 when front at 0, 0 when front at capacity - 1

copy(queue+start, queue+start+capacity-1, newQueue);

else

{

copy(queue+start, queue+capacity , newQueue);

copy(queue, queue+rear+1, newQueue+capacity-start);

}

front=2\*capacity-1;

rear=capacity-1;

capacity\*=2;

free(queue);

queue=newQueue;

}

void addq(element item)

{

rear=(rear+1)%capacity;

if(front==rear)

queueFull();

queue[rear]=item;

}

element deleteq()

{

element item;

if(front==rear)

{

item.n=-1;

return item;

}

front=(front+1)%capacity;

return queue[front];

}

void displayq()

{

int i;

if(front==rear)

{

printf("Queue Empty\n");

return; }

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

printf("%d\t",queue[i].n);

printf("%d", queue[i].n);

printf("\n");

// printf("Front: %d Rear: %d\n", front, rear);

}

void main()

{

int choice;

element item;

printf("Enter intial size");

scanf("%d",&capacity);

MALLOC(queue, capacity, element);

while(1)

{

printf("1. Add\n 2. Delete\n 3. Display\n");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter item to add");

scanf("%d",&item.n);

addq(item);

break;

case 2:

item=deleteq();

if(item.n==-1)

printf("Queue Empty");

else

printf("Item deleted: %d", item.n);

break;

case 3:

displayq();

break;

}

}

}

1. Write a C program to convert a given infix expression to a postfix expression using a stack.

#include<stdio.h>

#define MAX 20

typedef enum{lparen,rparen,plus,minus,times,divide,mod,eos,operand}precedence;

precedence stack[30];

int top=-1;

char EXPR[MAX];

int isp[]={0,19,12,12,13,13,13,0};

int icp[]={20,19,12,12,13,13,13,0};

void push(precedence token)

{

stack[++top]=token;

}

precedence pop()

{

return stack[top--];

}

precedence get\_token(char \*symbol,int \*n)

{

\*symbol=EXPR[(\*n)++];

switch(\*symbol)

{

case '(':return lparen;

case ')':return rparen;

case '+':return plus;

case '-':return minus;

case '\*':return times;

case '/':return divide;

case '%':return mod;

case '\0':return eos;

default:return operand;

}

}

void print\_token(precedence token)

{

switch(token)

{

case plus:printf("+");break;

case minus:printf("-");break;

case times:printf("\*");break;

case divide:printf("/");break;

case mod:printf("%%");break;

}

}

void postfix()

{

char symbol;

precedence token;

int n=0;

top=0;

stack[0]=eos;

for(token=get\_token(&symbol,&n);token!=eos;token=get\_token(&symbol,&n))

{

if(token==operand)

printf("%c",symbol);

else if(token==rparen)

{

while(stack[top]!=lparen)

print\_token(pop());

pop();

}

else

{

while(isp[stack[top]]>=icp[token])

print\_token(pop());

push(token);

}

}

while((token=pop())!=eos)

print\_token(token);

printf("\n");

}

void main()

{

printf("\nEnter the infix expression\n");

scanf("%s",EXPR);

postfix();

}

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

#include<stdio.h>

#define MAX 40

typedef enum{lparen,rparen,plus,minus,times,divide,mod,eos,operand}precedence;

char EXPR[MAX];

int stack[20];

int top=-1;

precedence get\_token(char \*symbol,int \*n)

{

\*symbol=EXPR[(\*n)++];

switch(\*symbol)

{

case '(':return lparen;

case ')':return rparen;

case '+':return plus;

case '-':return minus;

case '\*':return times;

case '/':return divide;

case '%':return mod;

case '\0':return eos;

default:return operand;

}

}

void push(int num)

{

stack[++top]=num;

}

int pop()

{

return stack[top--];

}

int eval()

{

precedence token;

char symbol;

int op1,op2,n=0;

token=get\_token(&symbol,&n);

while(token!=eos)

{

if(token==operand)

push(symbol-'0');

else

{

op2=pop();

op1=pop();

switch(token)

{

case plus:

push(op1+op2);

break;

case minus:

push(op1-op2);

break;

case times:

push(op1\*op2);

break;

case divide:

push(op1/op2);

break;

case mod:

push(op1%op2);

break;

}

}

token=get\_token(&symbol,&n);

}

return pop();

}

void main()

{

int res;

printf("\nEnter the postfix expression\n");

scanf("%s",EXPR);

res=eval();

printf("\nAfter evaluation:\t%d",res);

}

1. Write a C program to implement multiple linked stacks (at least 5) and perform the following operations on them
2. Push an item in ithstack (ii) Pop an item from ithstack(iii) Display ithstack

#include<stdio.h>

#include<stdlib.h>

#define MAXSIZE 10

typedef struct

{

int key;

}element;

struct stack

{

element data;

struct stack \*link;

};

typedef struct stack \*stckptr;

stckptr top[MAXSIZE];

void push(element item, int i)

{

stckptr temp;

temp=(stckptr)malloc(sizeof(stckptr\*));

temp->data=item;

temp->link=top[i];

top[i]=temp;

}

element pop(int i)

{

stckptr temp;

element item;

temp=top[i];

if(temp==NULL)

{

item.key=-1;

return item;

}

else

{

top[i]=top[i]->link;

item=temp->data;

free(temp);

return item;

}

}

void display(int i)

{

stckptr temp=top[i];

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

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

}

void main()

{

int z,ch,i;

element item;

for(z=0;z<MAXSIZE;z++)

top[z]=NULL;

do

{

printf("\n1.Push\n2.Pop\n3.Display\n4.Exit\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\nEnter item to be inserted:\t");

scanf("%d",&item.key);

printf("\nEnter stack number:\t");

scanf("%d",&i);

push(item,i-1);

break;

case 2:

printf("\nEnter stack number from which you would like to pop element:\t");

scanf("%d",&i);

item=pop(i-1);

if(item.key==-1)

printf("\nEmpty stack");

else

printf("\nDeleted element:\t%d",item.key);

break;

case 3:

printf("\nEnter stack number you would like to display:\t");

scanf("%d",&i);

display(i-1);

break;

case 4:

break;

default:

printf("\nWrong choice");

break;

}

}while(ch!=4);

}

1. Write a C program to implement multiple linked queues (at least 5) and perform the following operations on them
2. Add an item in ithqueue (ii) Delete an item from ithqueue(iii)Display ithqueue

#include<stdio.h>

#include<stdlib.h>

typedef struct

{

int key;

}element;

struct queue

{

element data;

struct queue\* link;

};

typedef struct queue\* queueptr;

queueptr front[10],rear[10];

void insert(element item,int i)

{

queueptr temp;

temp=(queueptr)malloc(sizeof(struct queue));

temp->data=item;

if(front[i])

{

rear[i]->link=temp;

}

else

front[i]=temp;

rear[i]=temp;

rear[i]->link=NULL;

}

element delete(int i)

{

queueptr temp;

temp=front[i];

element item;

if(front[i])

{

item=front[i]->data;

front[i]=front[i]->link;

}

else

{

item.key=-1;

}

free(temp);

return item;

}

void display(int i)

{

queueptr temp;

temp=front[i];

if(front[i] == NULL){

printf("Queue is Empty\n");

return;

}

printf("Queue %d is:",i+1);

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

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

printf("\n");

}

int main(){

int choice,qno;

element item;

int n;

printf("Enter How many Queus:");

scanf("%d",&n);

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

front[i]=rear[i]=NULL;

while(1){

printf(" 1.Insert \n 2. Delete \n 3.Display \n 4.Quit \n");

printf("Enter the Choice:");

scanf("%d",&choice);

if(choice != 4)

{

printf("Enter The Queue no from 1 to %d: ",n);

scanf("%d",&qno);

}

switch(choice)

{

case 1:

printf("Enter Data to be inserted:");

scanf("%d",&item.key);

insert(item,qno-1);

break ;

case 2:

item = delete(qno-1);

if(item.key==-1)

printf("Queue empty....\n");

else

printf("Element deleted:%d \n",item.key);

break;

case 3:

display(qno-1);

break;

case 4:

exit(0);

default :

printf("Invalid Input...");

}

}

}

1. 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 COMPARE(x,y)(x>y?1:(x<y?-1:0))

struct node

{

int coeff;

int expo;

struct node\* link;

};

typedef struct node\* polyptr;

polyptr a,b;

void attach(int coefficient, int exponent, polyptr \*ptr)

{

polyptr temp;

temp=(polyptr)malloc(sizeof(struct node));

temp->coeff=coefficient;

temp->expo=exponent;

(\*ptr)->link=temp;

\*ptr=temp;

//(\*ptr)->link=NULL;

}

polyptr cpadd(polyptr a, polyptr b)

{

polyptr c,lastC,startA;

int sum,done=0;

startA=a;

a=a->link;

b=b->link;

c=(polyptr)malloc(sizeof(struct node));

c->expo=-1;

lastC=c;

do

{

//printf("a: %d, b: %d",a->expo,b->expo);

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

{

case -1:

attach(b->coeff,b->expo,&lastC);

b=b->link;

break;

case 0:

if(startA==a)

done=1;

//printf("Equal\n");

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

if(sum)

attach(sum,a->expo,&lastC);

a=a->link;

b=b->link;

break;

case 1:

attach(a->coeff,a->expo,&lastC);

a=a->link;

break;

}

}while(!done);

lastC->link=c;

return c;

}

void printPoly(polyptr a)

{

a=a->link;

while(((a->link)->expo)!=-1)

{

printf("%d x ^ %d + ",a->coeff,a->expo);

a=a->link;

}

printf("%d x ^ %d",a->coeff,a->expo);

printf("\n");

}

void readPoly2(polyptr \*a)

{

\*a=(polyptr)malloc(sizeof(struct node));

polyptr temp;

(\*a)->expo=-1;

temp=\*a;

int expo;

int n;

int coeff;

int i=0;

printf("Enter number of terms: ");

scanf("%d",&n);

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

{

printf("Enter coeff and exponent %d",i);

scanf("%d%d",&coeff,&expo);

attach(coeff, expo, &temp);

}

temp->link=\*a;

}

int main(void) {

polyptr a,b,c;

readPoly2(&a);

printPoly(a);

readPoly2(&b);

printPoly(b);

c=cpadd(a,b);

printPoly(c);

}

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

(i) Insert a node (iii) Display a doubly linked circular list in forward direction

(ii) 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

{

int data;

nodeptr llink;

nodeptr rlink;

}node;

nodeptr head;

void dinsert()

{

int n;

nodeptr temp;

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

printf("Enter the element to be inserted:");

scanf("%d",&n);

temp->data = n;

temp->llink = head;

temp->rlink = head->rlink;

head->rlink->llink = temp;

head->rlink = temp;

}

void ddelete()

{

if(head->rlink==head)

printf("List is empty\n");

else{

nodeptr temp;

temp = head->rlink;

head->rlink = temp->rlink;

temp->rlink->llink = head;

printf("Deleted data:%d\n",temp->data);

free(temp);

}

}

void printfirst()

{

if(head->rlink==head)

printf("List is empty\n");

else{

nodeptr temp;

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

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

printf("\n");

}

}

void printlast()

{

if(head->rlink==head)

printf("List is empty\n");

else{

nodeptr temp;

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

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

printf("\n");

}

}

int main()

{

int c;

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

head->llink=head;

head->rlink=head;

while(1)

{

printf("1.Insert\n2.Delete\n3.Reverse Display\n4.forward Display\n5.Exit\nEnter your choice:");

scanf("%d",&c);

switch(c)

{

case 1:dinsert();

break;

case 2:ddelete();

break;

case 3:printfirst();

break;

case 4:printlast();

break;

case 5:exit(0);

}

}

return 0;

}

1. Write a C program to implement a max heap using an array and perform the following operations on it.

|  |  |
| --- | --- |
| #include<stdio.h> |  |
|  |  | #include <stdlib.h> |
|  |  | #define MAX\_SIZE 10 |
|  |  | typedef struct |
|  |  | { |
|  |  | int key; |
|  |  | }element; |
|  |  | element heap[MAX\_SIZE]; |
|  |  | void insert(element item, int \*n) |
|  |  | { |
|  |  | int i; |
|  |  | if((\*n)==MAX\_SIZE-1) |
|  |  | { |
|  |  | printf("Heap Full\n"); |
|  |  | return; |
|  |  | } |
|  |  | i=++(\*n); |
|  |  | while(i!=1 && item.key>heap[i/2].key) |
|  |  | { |
|  |  | heap[i]=heap[i/2]; |
|  |  | i/=2; |
|  |  | } |
|  |  | heap[i]=item; |
|  |  | } |
|  |  | element deleteHeap(int\* n) |
|  |  | { |
|  |  | int parent, child; |
|  |  | element temp, item; |
|  |  | if(\*n==0) |
|  |  | { |
|  |  | printf("Heap Empty\n"); |
|  |  | item.key=-1; |
|  |  | return item; |
|  |  | } |
|  |  | item = heap[1]; |
|  |  | temp = heap[(\*n)--]; |
|  |  | parent = 1; |
|  |  | child = 2; |
|  |  | while(child<=\*n) |
|  |  | { |
|  |  | if(child<\*n && heap[child].key < heap[child+1].key) |
|  |  | child++; |
|  |  | if(temp.key >= heap[child].key) |
|  |  | break; |
|  |  | heap[parent]=heap[child]; |
|  |  | parent=child; |
|  |  | child=child\*2; |
|  |  | } |
|  |  | heap[parent]=temp; |
|  |  | return item; |
|  |  | } |
|  |  | void display(int n) |
|  |  | { |
|  |  | int i; |
|  |  | for(i=1;i<=n;i++) |
|  |  | { |
|  |  | printf("%d\n",heap[i].key); |
|  |  | } |
|  |  | } |
|  |  | int main() |
|  |  | { |
|  |  | int choice,n=0; |
|  |  | element item; |
|  |  | while(1) |
|  |  | { |
|  |  | printf("Enter\n 1. Insert\n 2. Display\n 3. Delete\n 4. Exit"); |
|  |  | scanf("%d",&choice); |
|  |  | switch(choice) |
|  |  | { |
|  |  | case 1: |
|  |  | printf("Enter element to insert"); |
|  |  | scanf("%d", &item.key); |
|  |  | insert(item, &n); |
|  |  | break; |
|  |  | case 2: |
|  |  | display(n); |
|  |  | break; |
|  |  | case 3: |
|  |  | item = deleteHeap(&n); |
|  |  | if(item.key!=-1) |
|  |  | printf("Element Deleted: %d\n",item.key); |
|  |  | break; |
|  |  | case 4: |
|  |  | exit(0); } |
|  |  | } |
|  |  | } |

1. Write a C program to implement a binary search tree using linked representation and perform the following operations on it.

(i) Insert an item (ii) Search an item (iii) Inorder Traversal

#include<stdio.h>

#include<stdlib.h>

struct tree

{

int data;

struct tree \*rlink;

struct tree \*llink;

};

typedef struct tree \* treeptr;

void insert(treeptr \*root,int item)

{

if(!(\*root))

{

\*root=(treeptr)malloc(sizeof(treeptr\*));

(\*root)->data=item;

(\*root)->llink=NULL;

(\*root)->rlink=NULL;

return;

}

else if((\*root)->data>item)

insert(&(\*root)->llink,item);

else if((\*root)->data<item)

insert(&(\*root)->rlink,item);

}

void inorder(treeptr root)

{

if(root)

{

inorder(root->llink);

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

inorder(root->rlink);

}

}

void search(treeptr root,int item)

{

if(root==NULL)

{

printf("\nNot found");

return;

}

else if(root->data==item)

{

printf("\nFound");

return;

}

else if(root->data>item)

search(root->llink,item);

else if(root->data<item)

search(root->rlink,item);

}

int main()

{

int ch,item;

treeptr root;

root=NULL;

while(1)

{

printf("\n1.Insert\n2.InOrder\n3.Search\n4.Exit\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("\nEnter element to be inserted:\t");

scanf("%d",&item);

insert(&root,item);

break;

case 2:

inorder(root);

break;

case 3:

printf("\nEnter element to be deleted");

scanf("%d",&item);

search(root,item);

break;

case 4:

exit(1);

}

}

}

1. Write a C program to perform depth first search of a graph represented as an adjacency list.

#include<stdio.h>

#include<stdlib.h>

#define MAX 200

typedef struct node

{

struct node \*next;

int vertex;

}node;

void readgraph(); //create an adjecency list

void insert(int vi,int vj); //insert an edge (vi,vj)in adj.list

void DFS(int i);

int visited[MAX];

node \*G[20]; //heads of the linked list

int n;

void main()

{

int i,op;

do

{ printf("\n\n1)Create\n2)DFS\n4)Quit");

printf("\nEnter Your Choice: ");

scanf("%d",&op);

switch(op)

{ case 1: readgraph();break;

case 2: for(i=0;i<n;i++)

visited[i]=0;

printf("\nStarting Node No. : ");

scanf("%d",&i);

DFS(i);break;

}

}while(op!=4);

}

void DFS(int i)

{

node \*p;

visited[i]=1;

printf("\n%d",i);

for(p=G[i];p;p=p->next)

if(!visited[p->vertex])

DFS(p->vertex);

}

void readgraph()

{ int i,vi,vj,no\_of\_edges;

printf("\nEnter no. of vertices :");

scanf("%d",&n);

//initialise G[] with NULL

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

G[i]=NULL;

//read edges and insert them in G[]

printf("\nEnter no of edges :");

scanf("%d",&no\_of\_edges);

for(i=0;i<no\_of\_edges;i++)

{

printf("\nEnter an edge (u,v) :");

scanf("%d%d",&vi,&vj);

insert(vi,vj);

insert(vj,vi);

}

}

void insert(int vi,int vj)

{

node \*p,\*q;

//acquire memory for the new node

q=(node \*)malloc(sizeof(node));

q->vertex=vj;

q->next=NULL;

//insert the node in the linked list for the vertex no. vi

if(G[vi]==NULL)

G[vi]=q;

else

{

// go to the end of linked list

p=G[vi];

while(p->next!=NULL)

p=p->next;

p->next=q;

}

}