**Course Overview**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Structures Laboratory**  [As per Choice Based Credit System (CBCS) & OBE Scheme]  **SEMESTER – III** | | | |
| **Course Code:** | **P21ISL306** | **Credits:** | **01** |
| **Teaching Hours/Week (L:T:P):** | **0:0:2** | **CIE Marks:** | **50** |
| **Total Number of Lab Hours:** | **24** | **SEE Marks:** | **50** |

Data Structures laboratory is spatial and well equipped with the latest hp systems. It provides a wide approach in C programming and enables to apply knowledge in Artificial Intelligence. Guidance’s are provided to the students by a team of faculty experts and lab programmer. The labs are kept open after the college hours to enable the students to engage themselves in getting practiced with Laboratory experiments in their leisure hours. The lab is used to train the students to improve their programming knowledge and make experiments to get familiar with the Core Courses.

|  |  |
| --- | --- |
| **Note:** All programs are to be implemented using C Language | |
| 1. | Create a structure **DISTANCE** with data members ***kms*** and ***meters*** of type integer. Implement a program to perform addition and subtraction on two distances by passing pointer to a structure to function. |
| 2. | Implement a menu driven program to perform the following operations on Singly Linked List.   1. Create SLL of ‘n’ nodes of integers (insert front/rear) 2. Delete the node with specified integer from the list with appropriate message. 3. Display the contents of the SLL. |
| 3. | Implement a menu driven Program for the following operations on Doubly Linked List (DLL) of Library Data with the fields: BOOK\_ID, BOOK\_TITLE, AUTHOR, EDITION   1. Create a DLL of ‘N’ books (Insert front/rear). 2. Count the number of nodes in the DLL. 3. Delete the node at front/rear. 4. Display the contents of DLL. |
| 4. | Implement a menu driven Program for the following operations on Circular Linked List.   1. Create CLL of ‘n’ nodes of string. (insert front/rear) 2. Count the number of nodes in the CLL. 3. Delete the node at front/rear. 4. Display the contents of CLL. |
| 5. | Implement a menu driven Program for the following operations on STACK of Integers (Array Implementation of Stack with maximum size MAX)   1. Push an Element on to Stack (Handle the situation of overflow) 2. Pop an Element from Stack (Handle the situation of underflow) 3. Display the contents of Stack |
| 6. | Implement a Program to convert an infix expression to its equivalent postfix expression. |
| 7. | Implement the following using recursion:   1. Tower\_of\_Hanoi 2. GCD of two numbers 3. Largest of ‘n’ numbers |
| 8. | Implement a menu driven Program for the following operations on QUEUES of Strings using Linked list   1. Insert an Element into Queue 2. Delete an Element from Queue 3. Display the contents of Queue |
| 9. | Implement a menu driven program to perform the following operations on priority queue using linked list.   1. Insert a node based on priority. 2. Delete a node from the queue 3. Display the contents of the queue |
| 10. | Implement a menu driven Program for the following operations on Binary Search Tree (BST) of Integers   1. Create a BST of N Integers 2. Traverse the BST in Inorder, Preorder and Postorder |

#### Course Outcomes

After learning all the units of the course, students is able to:

1. Design algorithms using different data structures like List, Stack, Queue and Trees.
2. Develop programs with suitable data structure based on the requirements of the real time

applications.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CO** | **Statement** | **PO**  **1** | **PO**  **2** | **PO 3** | **PO 4** | **PO 5** | **PO 6** | **PO 7** | **PO 8** | **PO 9** | **PO 10** | **PO 11** | **PO**  **12** | **PSO**  **1** | **PSO**  **2** |
| **CO1** | **Design** algorithms using different data structures like List, Stack, Queue and Trees. | 2 | 2 | 2 |  |  |  |  |  |  |  |  |  | 2 | 2 |
| **CO2** | **Develop** programs with suitable data structure based on the requirements of the real-time applications. | 2 | 2 | 2 |  |  |  |  |  |  |  |  | 1 | 2 | 2 |

**1)** **Create a structure DISTANCE with data member’s kms and meters of type integer.**

**Implement a program to perform addition and subtraction on two distances by passing**

**pointer to a structure to function.**

#include <stdio.h>

#include <conio.h>

typedef struct distance

{

int kms;

int metres;

} DISTANCE;

DISTANCE add\_distance (DISTANCE \*, DISTANCE \*);

DISTANCE subtract\_distance(DISTANCE \*,DISTANCE \*);

DISTANCE dl, d2;

DISTANCE d3,d4;

int main()

{

int option;

clrscr ();

do

{

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

printf ("\n 1. Read the distances ");

printf ("\n 2. Display the distances");

printf ("\n 3. Add the distances ");

printf ("\n 4. Subtract the distances");

printf ("\n 5. EXIT");

printf ("\n Enter your option: ");

scanf("%d", &option);

switch(option)

{

case 1:

printf("\n Enter the first distance in kms and metres: ");

scanf ("%d %d", &dl .kms, &dl .metres);

printf("\n Enter the second distancekms and metres: ");

scanf ("%d %d" , &d2 .kms, &d2 .metres);

break;

case 2:

printf("\n The first distance is: %d kms %d metres " , dl.kms, dl.metres);

printf("\n The second distance is: %d kms %d metres " , d2 .kms, d2 .metres);

break;

case 3:

d3 = add\_distance(&dl,&d2);

printf("\n The sum of two distances is: %d kms %d metres", d3.kms, d3.metres);

break;

case 4:

d4 = subtract\_distance(&dl, &d2);

printf("\n The difference between two distances is: %d kms %d metres ", d4.kms, d4 .metres);

break;

}

}

while(option != 5);

{

getch ();

return 0;

}

}

DISTANCE add\_distance(DISTANCE \*dl, DISTANCE \*d2)

{

DISTANCE sum;

sum.metres = dl->metres + d2-> metres;

sum.kms = dl->kms + d2->kms;

if(sum.metres >= 1000)

{

sum.metres = sum.metres%1000;

sum.kms += 1;

}

return sum;

}

DISTANCE subtract\_distance(DISTANCE \*dl,DISTANCE \*d2)

{

DISTANCE sub;

if(dl->kms > d2->kms)

{

sub.metres = dl->metres - d2-> metres;

sub.kms = dl->kms - d2->kms;

}

else

{

sub.metres = d2->metres - dl-> metres;

sub.kms = d2->kms - dl->kms;

}

if(sub.metres < 0)

{

sub.kms = sub.kms - 1;

sub.metres = sub.metres + 1000;

}

return sub;

}

2**) Implement SLL which performs the following operations.**

**1. Create SLLof 'n' integers(insert front/rear)**

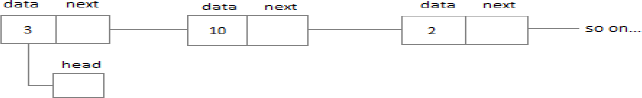
**2. Delete the specified integer from the list with appropriate message.**

**3. Display the contents of the list.**

Linked List is a linear data structure and it is very common data structure which consists of group of nodes in a sequence which is divided in two parts. Each node consists of its own data and the address of the next node and forms a chain. Linked Lists are used to create trees and graphs.

* They are a dynamic in nature which allocates the memory when required.
* Insertion and deletion operations can be easily implemented.
* Stacks and queues can be easily executed.
* Linked List reduces the access time.
* Linked lists are used to implement stacks, queues, graphs, etc.
* Linked lists let you insert elements at the beginning and end of the list.
* In Linked Lists we don’t need to know the size in advance.

**Singly Linked List:** Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in sequence of nodes. The operations we can perform on singly linked lists are insertion, deletion and traversal.



#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#include<malloc.h>

struct node

{

int data;

struct node \*next;

};

struct node \*start=NULL;

struct node \*create\_ll(struct node \*);

struct node \*display(struct node \*);

struct node \*insert\_beg(struct node \*);

struct node \*insert\_end(struct node \*);

struct node \*delete\_node(struct node \*);

int main()

{

int option;

do

{

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

printf("\n 1: Create a list");

printf("\n 2: Display the list");

printf("\n 3: Insert beginning");

printf("\n 4: Insert end");

printf("\n 5: Delete A specified node");

printf("\n\n Enter your option:");

scanf("%d",&option);

switch(option)

{

case 1: start=create\_ll(start);

printf("\nLinked list created");

break;

case 2: start=display(start);

break;

case 3: start=insert\_beg(start);

break;

case 4: start=insert\_end(start);

break;

case 5: start=delete\_node(start);

break;

}

}while(option!=6);

getch();

return 0;

}

struct node \*create\_ll(struct node \*start)

{

struct node \*new\_node,\*ptr;

int num;

printf("\n Enter -1 to end");

printf("\n Enter the data:");

scanf("%d",&num);

while(num !=-1)

{

new\_node=(struct node\*) malloc(sizeof(struct node));

new\_node->data=num;

if(start==NULL)

{

new\_node->next=NULL;

start=new\_node;

}

else

{

ptr=start;

while(ptr->next !=NULL)

ptr=ptr->next;

ptr->next=new\_node;

new\_node->next=NULL;

}

printf("Enter the data :");

scanf("%d",&num);

}

return start;

}

struct node \*display(struct node \*start)

{

struct node \*ptr;

ptr=start;

if(ptr==NULL)

{

printf("\nList is empty");

return 0;

}

while(ptr !=NULL)

{

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

ptr=ptr->next;

}

return start;

}

struct node \*insert\_beg(struct node \*start)

{

struct node \*new\_node;

int num;

printf("\n Enter the data");

scanf("%d",&num);

new\_node=(struct node \*)malloc(sizeof(struct node));

new\_node->data=num;

new\_node->next=start;

start=new\_node;

return start;

}

struct node \*insert\_end(struct node \*start)

{

struct node \*ptr, \*new\_node;

int num;

printf("\n Enter the data");

scanf("%d",&num);

new\_node=(struct node \*)malloc(sizeof(struct node));

new\_node->data=num;

new\_node->next=NULL;

ptr=start;

while(ptr->next !=NULL)

{

ptr=ptr->next;

}

ptr->next=new\_node;

return start;

}

struct node \*delete\_node(struct node \*start)

{

struct node \*ptr,\*preptr;

int val;

printf("\n Enter the value of the node which has to be deleted:");

scanf("%d",&val);

ptr=start;

if(ptr->data ==val)

{

start=start->next;

free(ptr);

return start;

}

else

{

while(ptr->data !=val)

{

preptr=ptr;

ptr=ptr->next;

}

preptr->next=ptr->next;

free(ptr);

return start;

}

}

**3) Implement a menu driven Program in C for the following operations on Doubly Linked**

**List (DLL) of Library Data with the fields: BOOK\_ID, BOOK\_TITLE, AUTHOR,**

**EDITION**

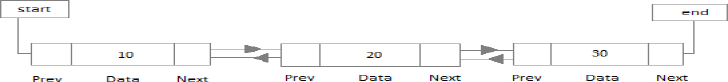
**1. Create an ordered DLL of N books.**

**2. Count the number of nodes in the DLL.**

**3. Delete a node at the specified position.**

**4. Display the contents of DLL.**

**Doubly Linked List:** In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence.



**Doubly Linked List**

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#include<malloc.h>

#include<string.h>

struct node

{

struct node \*prev;

int book\_id;

char book\_title[10];

char author[20];

int edition;

struct node \*next;

};

struct node \*start=NULL;

struct node \*create\_ll(struct node \*);

struct node \*display(struct node \*);

struct node \*count\_node(struct node \*);

struct node \*delete\_pos(struct node \*);

int main()

{

int option;

clrscr();

do

{

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

printf("\n 1: Create a list");

printf("\n 2: Display the list");

printf("\n 3: Count nodes");

printf("\n 4: Delete postion");

printf("\n\n Enter your option:");

scanf("%d",&option);

switch(option)

{

case 1: start=create\_ll(start);

printf("\nDoubly Linked list created");

break;

case 2: start=display(start);

break;

case 3: start=count\_node(start);

break;

case 4: start=delete\_pos(start);

break;

}

}while(option!=5);

getch();

return 0;

}

struct node \*create\_ll(struct node \*start)

{

struct node \*new\_node,\*ptr,\*preptr;

int book\_id,edition;

char book\_title[10],author[20];

printf("\n EnterBokk id as -1 to end");

printf("\n Enter the Book Id:");

scanf("%d",&book\_id);

printf("\n Enter the Book Title:");

scanf("%s",book\_title);

printf("\n Enter the Author:");

scanf("%s",author);

printf("\n Enter the Book Edition:");

scanf("%d",&edition);

while(book\_id!=-1)

{

new\_node=(struct node\*) malloc(sizeof(struct node));

new\_node->prev=NULL;

new\_node->book\_id=book\_id;

strcpy(new\_node->book\_title,book\_title);

strcpy(new\_node->author,author);

new\_node->edition=edition;

new\_node->next=NULL;

if(start==NULL)

{

start=new\_node;

}

else

{

ptr=start;

preptr=NULL;

while(ptr!=NULL&& book\_id> ptr->book\_id)

{

preptr=ptr;

ptr=ptr->next;

}

preptr->next=new\_node;

new\_node->next=ptr;

}

printf("Enter the Bookid:");

scanf("%d",&book\_id);

if(book\_id==-1)

break;

printf("\n Enter the Book Title:");

scanf("%s",book\_title);

printf("\n Enter the Author:");

scanf("%s",author);

printf("\n Enter the Book Edition:");

scanf("%d",&edition);

}

return start;

}

struct node \*display(struct node \*start)

{

struct node \*ptr;

ptr=start;

if(ptr==NULL)

{

printf("\nList is empty");

return 0;

}

while(ptr !=NULL)

{

printf("\t %d",ptr->book\_id);

printf("\t %s",ptr->book\_title);

printf("\t %s",ptr->author);

printf("\t %d",ptr->edition);

ptr=ptr->next;

}

return start;

}

struct node \*count\_node(struct node \*start)

{

struct node \*ptr;

int count=0;

ptr=start;

while(ptr!=NULL)

{

ptr=ptr->next;

count++;

}

printf("\nTotal Numbers of node=%d",count);

return start;

}

struct node \*delete\_pos(struct node \*start)

{

struct node \*ptr,\*temp,\*preptr;

int i,pos;

printf("Enter the postion");

scanf("%d",&pos);

temp=start;

for(i=1;i<pos && temp !=NULL;i++)

{

preptr=temp;

temp=temp->next;

ptr=temp->next;

}

if(ptr!=NULL)

{

preptr->next=ptr;

ptr->prev=preptr;

free(temp);

}

else

{

printf("\nEnter valid postion");

}

return start;

}

**4) Implement a program to add two polynomials.**

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

#include<malloc.h>

struct node

{

int num;

int coeff;

struct node \*next;

};

struct node \*start1=NULL;

struct node \*start2=NULL;

struct node \*start3=NULL;

struct node \*start4=NULL;

struct node \*create\_poly(struct node \*);

struct node \*display\_poly(struct node \*);

struct node \*add\_poly(struct node \*,struct node \*,struct node \*);

struct node \*add\_node(struct node \*,int,int);

int main()

{

int option;

clrscr();

do

{

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

printf("\n 1: Enter the First Polynomial");

printf("\n 2: Display the First Polynomial");

printf("\n 3: Enter the Second Polynomial ");

printf("\n 4: Display the Second Polynomial");

printf("\n 5: Add the polnomials");

printf("\n 6:Display the Result");

printf("\n 7: Subtract the Polnomials");

printf("\n 8: Display subtraction result");

printf("\n\n Enter your option:");

scanf("%d",&option);

switch(option)

{

case 1: start1=create\_poly(start1);

break;

case 2: start1=display\_poly(start1);

break;

case 3: start2=create\_poly(start2);

break;

case 4: start2=display\_poly(start2);

break;

case 5: start3=add\_poly(start1,start2,start3);

break;

case 6: start3=display\_poly(start3);

break;

}

}while(option!=7);

getch();

return 0;

}

struct node \*create\_poly(struct node \*start)

{

struct node \*new\_node,\*ptr;

int n,c;

printf("\n Enter the number:");

scanf("%d",&n);

printf("\t Enter its coefficient:");

scanf("%d",&c);

while(n !=-1)

{

if(start==NULL)

{

new\_node=(struct node\*) malloc(sizeof(struct node));

new\_node->num=n;

new\_node->coeff=c;

new\_node->next=NULL;

start=new\_node;

}

else

{

ptr=start;

while(ptr->next !=NULL)

ptr=ptr->next;

new\_node=(struct node\*) malloc(sizeof(struct node));

new\_node->num=n;

new\_node->coeff=c;

ptr->next=new\_node;

new\_node->next=NULL;

}

printf("Enter the number :");

scanf("%d",&n);

if(n==-1)

break;

printf("\t Enter its Coefficient");

scanf("%d",&c);

}

return start;

}

struct node \*display\_poly(struct node \*start)

{

struct node \*ptr;

ptr=start;

while(ptr !=NULL)

{

printf("\n %d x %d\t",ptr->num,ptr->coeff);

ptr=ptr->next;

}

return start;

}

struct node \*add\_poly(struct node \*start1,struct node \*start2,struct node \*start3)

{

struct node \*ptr1,\*ptr2;

int sum\_num,c;

ptr1=start1,ptr2=start2;

while(ptr1 !=NULL && ptr2 !=NULL)

{

if(ptr1->coeff ==ptr2->coeff)

{

sum\_num=ptr1->num+ptr2->num;

start3=add\_node(start3,sum\_num,ptr1->coeff);

ptr1=ptr1->next;

ptr2=ptr2->next;

}

else if (ptr1->coeff > ptr2->coeff)

{

start3=add\_node(start3,ptr1->num,ptr1->coeff);

ptr1=ptr1->next;

}

else if (ptr1->coeff < ptr2->coeff)

{

start3=add\_node(start3,ptr2->num,ptr2->coeff);

ptr2=ptr2->next;

}

}

if(ptr1 ==NULL)

{

while(ptr2!=NULL)

{

start3=add\_node(start3,ptr2->num,ptr2->coeff);

ptr2=ptr2->next;

}

}

if(ptr2 ==NULL)

{

while(ptr1!=NULL)

{

start3=add\_node(start3,ptr1->num,ptr1->coeff);

ptr1=ptr1->next;

}

}

return start3;

}

struct node \*add\_node(struct node \*start,int n,int c)

{

struct node \*ptr,\*new\_node;

if(start==NULL)

{

new\_node=(struct node\*) malloc(sizeof(struct node));

new\_node->num=n;

new\_node->coeff=c;

new\_node->next=NULL;

start=new\_node;

}

else

{

ptr=start;

while(ptr->next !=NULL)

ptr=ptr->next;

new\_node=(struct node\*) malloc(sizeof(struct node));

new\_node->num=n;

new\_node->coeff=c;

ptr->next=new\_node;

new\_node->next=NULL;

}

return start;

}

**5) Implement a menu driven Program for the following operations on STACK of Integers**

**(Array Implementation of Stack with maximum size MAX)**

**1. Push an Element on to Stack (Handle the situation of overflow)**

**2. Pop an Element from Stack (Handle the situation of underflow)**

**3. Display the status of Stack**

**Support the program with appropriate functions for each of the above operations.**

## ABOUT THE EXPERIMENT:

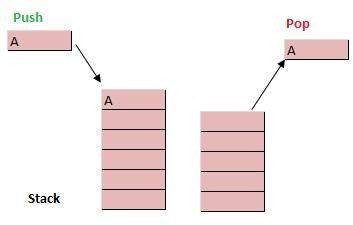
A stack is an abstract data type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack.



A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from top of the stack only. Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack.

This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last is accessed first. In stack terminology, insertion operation is called **PUSH** operation and removal operation is called **POP** operation.

Below given diagram tries to depict a stack and its operations −



A stack can be implemented by means of Array, Structure, Pointer and Linked-List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays which makes it a fixed size stack implementation.

Basic Operations:

* **push()** - pushing (storing) an element on the stack.
* **pop()** - removing (accessing) an element from the stack.

To use a stack efficiently we need to check status of stack as well. For the same purpose, the following functionality is added to stacks;

* **peek()** − get the top data element of the stack, without removing it.
* **isFull()** − check if stack is full.
* **isEmpty()** − check if stack is empty.

## ALGORITHM:

Step 1: Start.

Step 2: Initialize stack size MAX and top of stack -1.

Step 3: Push integer element on to stack and display the contents of the stack. if stack is full give a message as ‘Stack is Overflow’.

Step 3: Pop element from stack along with display the stack contents. if stack is empty give a message as ‘Stack is Underflow’.

Step 4: Stop.

/\* Program to Implement Stack using Array \*/

#include<stdio.h>

#define STACK\_SIZE 5

void push(int,int \*,int []);

int pop(int \*,int []);

void display(int , int []);

void main()

{

int s[STACK\_SIZE],ch,top=-1,item,item\_deleted;

clrscr();

for(;;)

{

printf("Press 1 .Push\n 2. Pop \n 3.Display\n");

printf("Enter your choice\n");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("Enter the item to be inserted\n");

scanf("%d",&item);

push(item,&top,s);

break;

case 2: item\_deleted=pop(&top,s);

if(item\_deleted==0)

{

printf("Stack is Empty\n");

return;

}

printf("Item deleted =%d\n",item\_deleted);

break;

case 3: display(top,s);

break;

default: exit(0);

}//switch

}//for

getch();

}//main

void push(int item,int \*top,int s[])

{

if(\*top==STACK\_SIZE-1)

{

printf("STACK is FULL\n");

return;

}

\*top=\*top+1;

s[\*top]=item;

}

int pop(int \*top,int s[])

{

int item;

if(\*top==-1)

{

return 0;

}

item=s[\*top];

\*top=\*top-1;

return item;

}

void display(int top,int s[])

{

int i;

if(top==-1)

{

printf("Stack is empty\n");

return;

}

printf("Content of the stack\n");

for(i=top;i>=0;i--)

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

}

Output:

Press 1 .Push

2. Pop

3.Display

Enter your choice: 1

Enter the item to be inserted 10

Press 1 .Push

2. Pop

3.Display

Enter your choice: 1

Enter the item to be inserted 20

Press 1 .Push

2. Pop

3.Display

Enter your choice: 1

Enter the item to be inserted 30

Press 1 .Push

2. Pop

3.Display

Enter your choice: 3

Content of the stack

30

20

10

Press 1 .Push

2. Pop

3.Display

Enter your choice: 2

Item deleted =30

Press 1 .Push

2. Pop

3.Display

Enter your choice: 2

Item deleted =20

Press 1 .Push

2. Pop

3.Display

Enter your choice: 2

Stack is Empty

**6) Implement a Program to convert infix to postfix expression.**

## ABOUT THE EXPERIMENT:

**Infix:** Operators are written in-between their operands.

Ex: X + Y

**Prefix:** Operators are written before their operands.

Ex: +X Y

**postfix:** Operators are written after their operands.

Ex: XY+

**Examples of Infix, Prefix, and Postfix**

|  |  |  |
| --- | --- | --- |
| **Infix Expression** | **Prefix Expression** | **Postfix Expression** |
| A + B | + A B | A B + |
| A + B \* C | + A \* B C | ABC\*+ |

**Infix to prefix conversion** Expression = **(A+B^C)\*D+E^5 Step 1.** Reverse the infix expression.

**5^E+D\*)C^B+A(**

**Step 2.** Make Every '(' as ')' and every ')' as '('

**5^E+D\*(C^B+A)**

**Step 3.** Convert expression to postfix form.

**Step 4.** Reverse the expression.

**+\*+A^BCDE**

**Step 5. Result**

+\*+A^BCD^E5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A+(B\*C-(D/E-F)\*G)\*H | | | | |
| **Expression** | | **Stack** | **Output** | **Comment** |
|  | 5^E+D\*(C^B+A) | Empty | - | Initial |
| ^E+D\*(C^B+A) | | Empty | 5 | Print |
|  | E+D\*(C^B+A) | ^ | 5 | Push |
| +D\*(C^B+A) | | ^ | 5E | Push |
|  | D\*(C^B+A) | + | 5E^ | Pop And Push |
| \*(C^B+A) | | + | 5E^D | Print |
|  | (C^B+A) | +\* | 5E^D | Push |
| C^B+A) | | +\*( | 5E^D | Push |

|  |  |  |  |
| --- | --- | --- | --- |
| ^B+A) | +\*( | 5E^DC | Print |
| B+A) | +\*(^ | 5E^DC | Push |
| +A) | +\*(^ | 5E^DCB | Print |
| A) | +\*(+ | 5E^DCB^ | Pop And Push |
| ) | +\*(+ | 5E^DCB^A | Print |
| End | +\* | 5E^DCB^A+ | Pop Until '(' |
| End | Empty | 5E^DCB^A+\*+ | Pop Every element |

**ALGORITHM:**

Step 1: Start.

Step 2: Read an infix expression with parenthesis and without parenthesis.

Step 3: convert the infix expression to postfix expression.

Step 4: Stop

#include<stdio.h>

#include<ctype.h>

int top=-1;

char stack[20];

void push(char x)

{

++top;

stack[top]=x;

}

char pop()

{

if(top == -1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x == '(')

return 1;

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

return 2;

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

return 3;

else

return 0;

}

int main()

{

char ex[21];

char \*p,x;

printf("\n");

printf("Enter an expression in infix form\n");

scanf("%s",ex);

p=ex;

printf("The postfix expression is ");

while(\*p != '\0')

{

if(isalnum(\*p))

{

printf("%c",\*p);

}

else if(\*p=='(')

{

push(\*p);

}

else if(\*p==')')

{

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

printf("%c",x);

}

else

{

while(priority(stack[top])>=priority(\*p))

{

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

}

push(\*p);

}

p++;

}

while(top !=-1)

{

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

}

return 0;

}

**7) Implement the following using recursion:**

**1. Tower\_of\_Hanoi**

**2. GCD of two numbers**

**3. Largest of 'n' numbers**.

1. The **Tower of Hanoi** is a [mathematical game](https://en.wikipedia.org/wiki/Mathematical_game) or [puzzle.](https://en.wikipedia.org/wiki/Puzzle) It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a [conical](https://en.wikipedia.org/wiki/Cone) shape. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:
   * Only one disk can be moved at a time.
   * Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
   * No disk may be placed on top of a smaller disk.

With three disks, the puzzle can be solved in seven moves. The minimum number of moves required to solve a Tower of Hanoi puzzle is 2*n* - 1, where *n* is the number of disks.

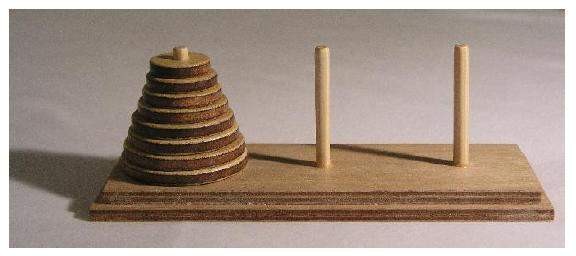
## ALGORITHM:

Step 1: Start.

Step 2: Read N number of discs.

Step 3: Move all the discs from source to destination by using temp rod.

Step 4: Stop.

#include<stdio.h>

int count=0;

void tower(int n,int source,int temp,int destination)

{

if(n==1)

{

printf("Move disc 1 from %c to %c\n",source,destination);

count++;

return;

}

tower(n-1,source,destination,temp);

printf("Move disc %d from %c to %c\n",n,source,destination);

count++;

tower(n-1,temp,source,destination);

}

void main()

{

int n;

clrscr();

printf("Enter the number of disc\n");

scanf("%d",&n);

tower(n,'A','B','C');

printf("Total number of disc moves=%d",count);

getch();

}

ii) GCD of two numbers using Recursion

#include<stdio.h>

#include<conio.h>

int fact(int n)

{

if(n==0) return 1;

return n\*fact(n-1);

}

void main()

{

int n,fac;

clrscr();

printf("Enter a number\n");

scanf("%d",&n);

fac=fact(n);

printf("Factoriali(%d) is %d\n",n,fac);

getch();

}

**8) Implement a menu driven Program for the following operations on QUEUES of Strings using**

**Linked list**

**1. Insert an Element into Queue**

**2. Delete an Element from Queue**

**3. Display the status of Queue**

#include < stdio.h >

#include < stdlib.h >

#include<string.h>

struct node {

char data[10];

struct node \* next;

};

struct node \* front = NULL;

struct node \* rear = NULL;

void enqueue(char value[])

{

struct node \* ptr;

ptr = (struct node \* ) malloc(sizeof(struct node));

//ptr -> data = value;

strcpy(ptr->data,value);

ptr -> next = NULL;

if ((front == NULL) && (rear == NULL)) {

front = rear = ptr;

}

else

{

rear -> next = ptr;

rear = ptr;

}

printf("Node is Inserted\n\n");

}

void dequeue()

{

struct node \* temp = front;

char temp\_data[10];

strcpy(temp\_data,front->data);

if (front == NULL)

{

printf("\nUnderflow\n");

exit(0);

}

else

{

front = front -> next;

free(temp);

printf("Deleted Element is %s\n",temp\_data);

}

}

void display()

{

struct node \* temp;

if ((front == NULL) && (rear == NULL))

{

printf("\nQueue is Empty\n");

}

else

{

printf("The queue is \n");

temp = front;

while (temp) {

printf("%s--->", temp -> data);

temp = temp -> next;

}

printf("NULL\n\n");

}

}

int main()

{

int choice;

char value[10];

printf("\nImplementation of Queue using Linked List\n");

while (choice != 4)

{

printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");

printf("\nEnter your choice : ");

scanf("%d", & choice);

switch (choice)

{

case 1:

printf("\nEnter the value to insert: ");

scanf("%s", &value);

enqueue(value);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

exit(0);

break;

default:

printf("\nWrong Choice\n");

}

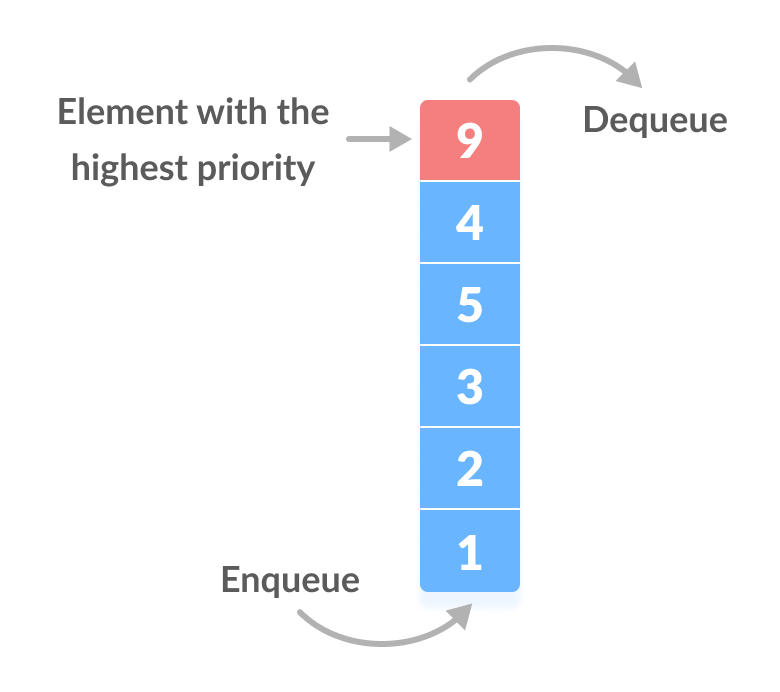
}

return 0;

}

**9) Implement a program to perform the operations on priority queue.**

The priority queue in the data structure is **an extension of the “normal” queue**. It is an abstract data type that contains a group of items. It is like the “normal” queue except that the dequeuing elements follow a priority order. The priority order dequeues those items first that have the highest priority.



/\* Program to implement Priority Queue \*/

#include<stdio.h>

#define QUEUE\_SIZE 5

void insert\_item(int,int \*,int []);

int delete\_front(int \*,int \*,int []);

void display(int ,int, int []);

void main()

{

int q[QUEUE\_SIZE],ch,f=0,r=-1,item,item\_deleted;

clrscr();

for(;;)

{

printf("Press 1 .INSERT ITEM\n 2. DELETE FRONT \n 3 Display\n");

printf("Enter your choice\n");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("Enter the item to be inserted\n");

scanf("%d",&item);

insert\_item(item,&r,q);

break;

case 2: item\_deleted=delete\_front(&f,&r,q);

if(item\_deleted==0)

printf("Queue is Empty\n");

else

printf("Item deleted =%d\n",item\_deleted);

break;

case 3: display(f,r,q);

break;

default: exit(0);

}//switch

}//for

getch();

}//main

void insert\_item(int item,int \*r,int q[])

{

int j;

if(\*r==QUEUE\_SIZE-1)

{

printf("Queue is Full\n");

return;

}

j=\*r;

while(j>=0 && item<q[j])

{

q[j+1]=q[j];

j--;

}

q[j+1]=item;

\*r=\*r+1;

}

int delete\_front(int \*f,int \*r,int q[])

{

int item;

if(qempty(\*f,\*r))

{

//printf("QUEUE is Empty\n");

return 0;

}

item=q[\*f];

\*f=\*f+1;

if(\*f>\*r)

{

\*f=0;

\*r=-1;

}

return item;

}

void display(int f,int r,int q[])

{

int i;

if(qempty(f,r))

{

printf("QUEUE is empty\n");

return;

}

printf("Content of the QUEUE\n");

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

printf("%d\t",q[i]);

printf("\n");

}

int qfull(int r)

{

if(r==QUEUE\_SIZE-1)

return 1;

else

return 0;

}

int qempty(int f,int r)

{

if(f>r)

return 1;

else

return 0;

}

**10) Implement a menu driven Program for the following operations on Binary Search Tree**

**(BST) of Integers**

**1. Create a BST of N Integers**

**2. Traverse the BST in Inorder, Preorder and Postorder**

#include<stdio.h>

#include<conio.h>

#include<alloc.h>

#include<process.h>

#include<string.h>

struct node

{

int info;

struct node \*llink;

struct node \*rlink;

};

typedef struct node\* NODE;

NODE insert(int,NODE);

NODE getnode();

void inorder(NODE);

void preorder(NODE);

void postorder(NODE);

void main()

{

int ch,item,flag,item\_deleted;

NODE root=NULL,min;

clrscr();

for(;;)

{

printf("Press 1 .Insert \n 2. Inorder\n 3 preorder\n");

printf("4.postorder \n");

printf("Enter your choice\n");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("Enter the item to be inserted\n");

scanf("%d",&item);

root=insert(item,root);

break;

case 2: printf("Inorder tree traversal\n");

inorder(root);

break;

case 3: printf("preorder tree traversal\n");

preorder(root);

break;

case 4: printf("postorder tree traversal\n");

postorder(root);

break;

default: exit(0);

}//switch

}//for

getch();

}//main

NODE getnode()

{

NODE x;

x=(NODE)malloc(sizeof(struct node));

if(x==NULL)

{

printf("Out of memeory\n");

exit(0);

}

return x;

}

NODE insert(int item,NODE root)

{

NODE temp,cur,prev;

temp=getnode();

temp->info=item;

temp->llink=temp->rlink=NULL;

if(root==NULL) return temp;

prev=NULL;

cur=root;

while( cur !=NULL)

{

prev=cur;

if(item <cur->info)

cur=cur->llink;

else

cur=cur->rlink;

}

if(item< prev->info)

prev->llink=temp;

else

prev->rlink=temp;

return root;

}

void inorder(NODE root)

{

if(root !=NULL)

{

inorder(root->llink);

printf("%d ",root->info);

inorder(root->rlink);

}

}

void preorder(NODE root)

{

if(root !=NULL)

{

printf("%d ",root->info);

preorder(root->llink);

preorder(root->rlink);

}

}

void postorder(NODE root)

{

if(root !=NULL)

{

postorder(root->llink);

postorder(root->rlink);

printf("%d ",root->info);

}

}