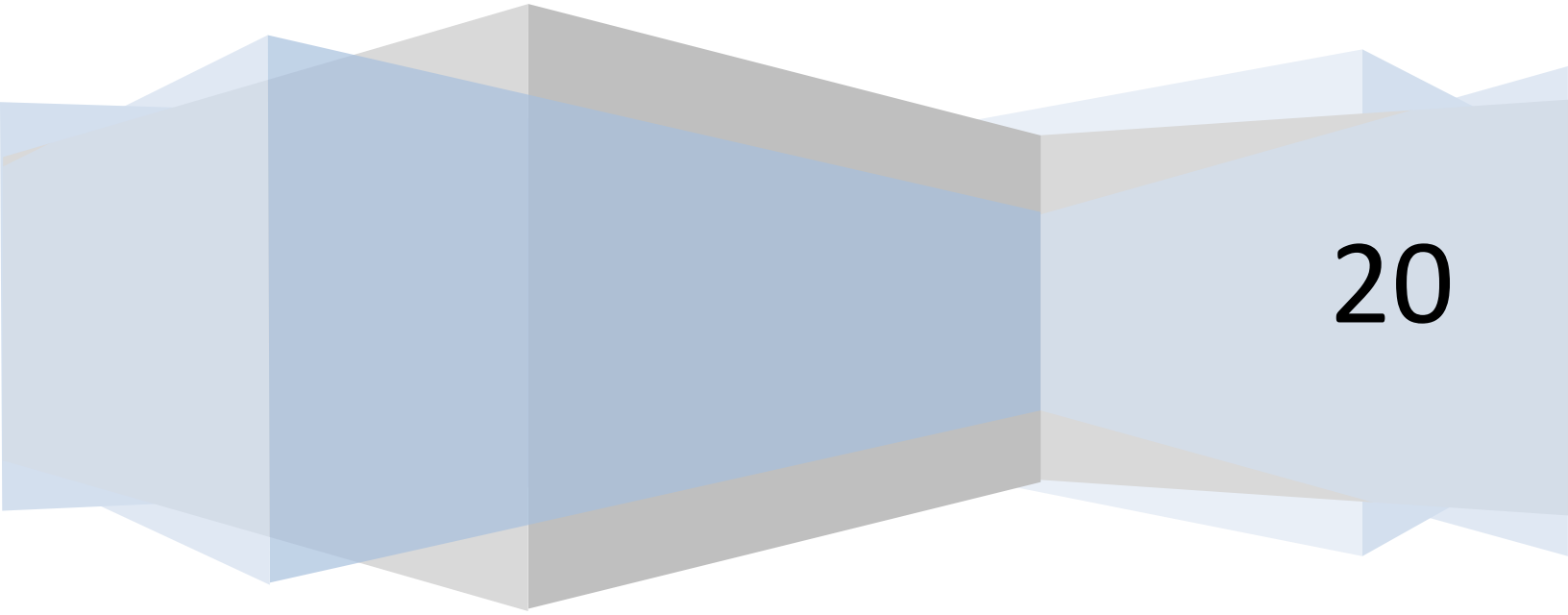
MIET Career Guidance & Student Counseling Cell (CGSC)

‘DATA STRUCTURE’ Programming Training Module

Classroom

HODCS



20

'DATA STRUCTURE' PROGRAMMING TRAINING MODULE

The C programming training module will make the reader accustomed to C language. This material will help the reader in understanding the basic of the C language, C functions & the use of C language to build simple projects.

DELIVERY METHOD	• 25% Self-paced Learning • 75% Instructor led Training
VERSION	• 2020
LEARNING OBJECTIVES	• Illustrate features, application, errors & structure of C program • Identify keywords, identifier, constant & Choose appropriate datatype • Classification of operators, precedence & associativity • Use different Input/output statement, Escape sequences • Type casting, simple C programs • Contrast the use of simple if, if else statement • Contrast the use of nested if else with programs • Contrast the use of if else ladder with programs • Criticize the limitation of conditional operator &goto statement • Examine the use switch statement & write programs based on choice • Write switch programs based on choices & other programs • Discuss use of for, while & do while loop with syntax • Simple & complex while & do while loop programs • Compare while with Do While, Distinguish break & continue • Determine the use of different type of pointer & its importance • Define Function & Contrast the use of call by value & call by reference • Make programs based on modularizing approach • Concept of storage class & programs related to recursion • Concept & program related to 1d, 2d array

	• Use of string & string handling library functions • Implementation of string handling library functions • Concept & program related to structure • Concept of union & enumerated data type • Appraise the usage of preprocessor directives like macro substitution, file inclusion etc. • Make use of different file handling library functions • Write simple & complex programs related to file handling • Concepts & dynamic memory allocation programs • Complex programs related to pointer & DMA • Discuss the concept of data structure & introduction to stack, queue & linked list • Make use of command line argument while writing any c program
PREREQUISITES SKILLS	• Computer Science Fundamentals • Basic Knowledge of Applied Mathematics & Algorithm
DURATION	• 30 Hours
SKILL LEVEL	• Basic-Intermediate
HARDWARE REQUIREMENTS	• Processor: 2 GHZ or Higher • GB RAM: 8 GB • GB DISK: 80 GB • Network Requirement: Yes

Notes:

The following unit and exercise durations are estimates and might not reflect every class experience. The estimates do not include the duration of optional exercises or sections. Student in this course use a dosbox, gcc or online compiler to perform the exercises.

COURSE AGENDA: MODULE-2

Lecture I–Searching Technique

Duration: 3 Hrs.

Overview	This unit explains concept of Searching for finding out whether a particular element is present in the list. The method that we use for this depends on how the elements of the list are organized. If the list is an unordered list, then we use linear or sequential search, whereas if the list is an ordered list, then we use binary search.
Learning Objectives	After completing this unit, you should be able to • Learn Sequential Search • Binary Search
Lab Exercise	• Write a program to implement sequential Search • Write a program to implement Binary Search

Lecture II– Sorting Technique

Duration: 3Hrs.

Overview	This unit allows you to do sorting for the following reasons : a) By keeping a data file sorted, we can do binary search on it. b) Doing certain operations, like matching data in two different files, become much faster. There are various methods for sorting: Bubble sort, Insertion sort, Selection sort, Quick sort etc.
Learning Objectives	After completing this unit, you should be able to • Use in large variety of important applications so as to arrange the data in ascending or descending order.
Lab Exercise	• Write a program to implement Bubble Sort. • Write a program to implement Insertion Sort. • Write a program to implement Selection Sort. • Write a program to implement Quick Sort.

Lecture III– Stack

Duration: 3 Hrs.

Overview	A stack is simply a list of elements with insertions and deletions permitted at one end—called the stack top. That means that it is possible to remove elements from a stack in reverse order from the insertion of elements into the stack. Thus, a stack data structure exhibits the LIFO (last in first out) property. Push and pop are the operations that are provided for insertion of an element into the stack and the removal of an element from the stack.
Learning Objectives	After completing this unit, you should be able to • Implement Stack • Understand the performance of the implementations of basic linear data structure • To use stacks to convert expressions from infix to prefix.
Lab Exercise	• Write a program to implement a stack using an array. • Write a program for implementation of a stack using the linked list • Write a program to convert an infix expression to prefix form.

Lecture IV– Queue

Duration: 3 Hrs.

Overview	A queue is also a list of elements with insertions permitted at one end—called the rear, and deletions permitted from the other end—called the front. This means that the removal of elements from a queue is possible in the same order in which the insertion of elements is made into the queue. Thus, a queue data structure exhibits the FIFO (first in first out) property.
Learning Objectives	After completing this unit, you should be able to • Do operations on Queue. • Write array representation of Queue. • Write Linked representation of Queue. • Learn types of Queues
Lab Exercise	• Write a program to implement a queue by using an array. • Write a program for implementation of a Queue using the linked list. • Write a program to implement a priority queue

Lecture V– Singly Linked List

Duration: 3 Hrs.

Overview	A linked list is a data structure that is used to model such a dynamic list of data items; It is a very commonly used linear data structure which consists of group of nodes in a sequence. Each node holds its own data and the address of the next node hence forming a chain like structure.
Learning Objectives	After completing this unit, you should be able to • Learn that Linked Lists can be used to implement Stacks , Queues. • Insertion and deletion operations can be easily implemented.
Lab Exercise	• Write a program for building and printing the elements of the linked list. • Write a program to reverse the list. • Write a program to delete the specific node in a singly linked list. • Write a program to insert a node after the specific node in a singly linked list. • Write a program to delete a linked list.

Lecture VI– Doubly and Circular Linked List

Duration: 3 Hrs.

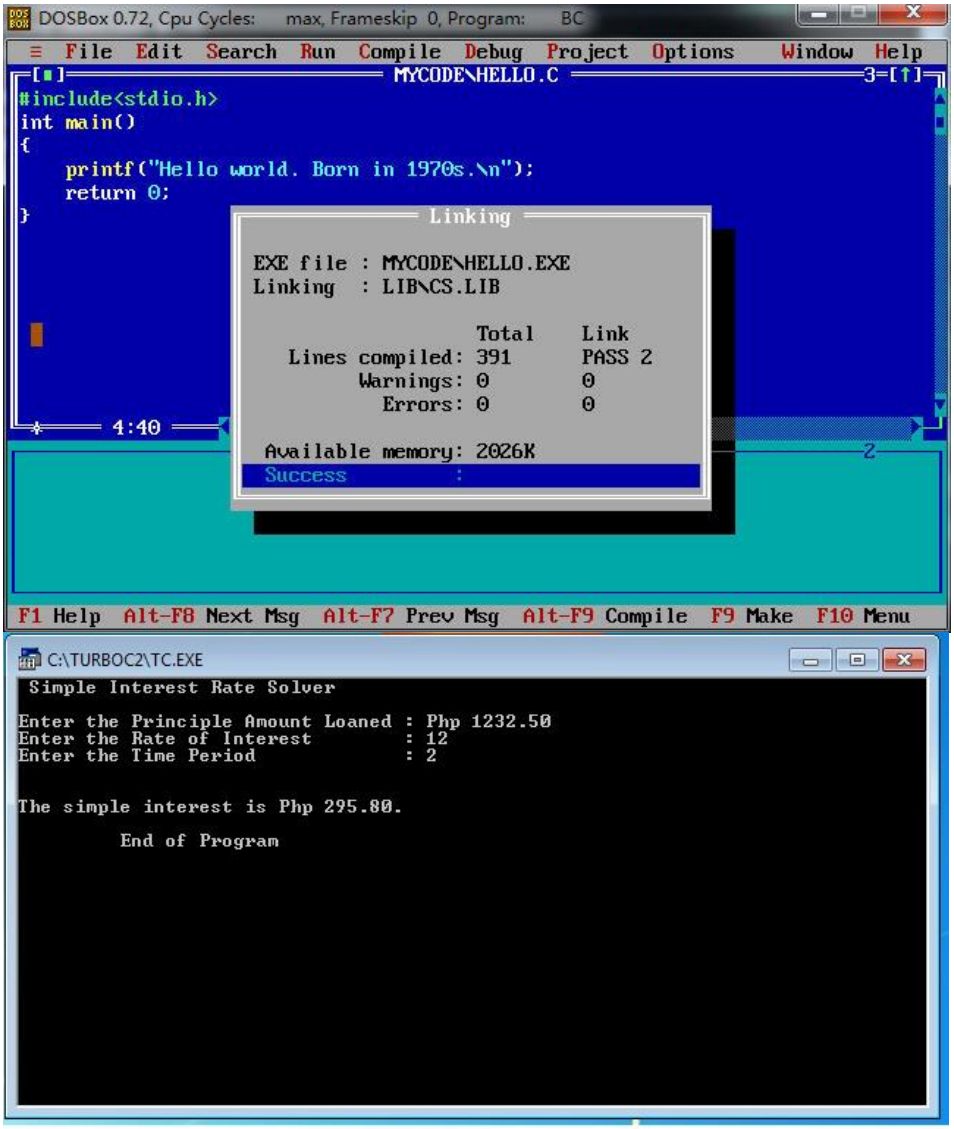
Overview	In a doubly linked list, each node contains a data part and two addresses, one for the previous node and one for the next node. In circular linked list the last node of the list holds the address of the first node hence forming a circular chain.
Learning Objectives	After completing this unit, you should be able to • Implement Doubly Linked and Circular Linked List. • Learn the traversal of the list in both direction(Left & Right)
Lab Exercise	• Write a program for building and printing the elements of a doubly linked list. • Write a program to inserts the data in a doubly linked list. • Write a program to delete a specific node from a doubly linked list. • Write a program for building and printing the elements of the circular linked list.

Lecture VII– Tree

Duration: 3 Hrs.

Overview	This Lecture gives the knowledge about trees (non-linear data structure) that are used to impose a hierarchical structure on a collection of data items. A tree is a set of one or more nodes T such that: i. there is a specially designated node called a root ii. The remaining nodes are partitioned into n disjointed set of nodes T1, T2,…,Tn, each of which is a tree.
Learning Objectives	After completing this unit, you should be able to • Implement Binary Search Tree. • Searching& Deletion operation in Binary Search Tree.
Lab Exercise	• Write a program to implement Binary Search Tree. • Write a program to count the number of nodes in Binary Search Tree. • Write a program to search a target key in Binary Search Tree. • Write a program to delete a node in Binary Search Tree.

SAMPLE INTERFACE



Sample Program Output

PROGRAM SOLUTION

Problem Statement	Write a program to implement sequential Search.
Concept	• Initialize and declare an array of any size. • Enter the value to be identified by using the linear search. • Linear search starts from the index value 0. • Returns the element if found, else it will increment the index by value 1. • Again scans for the element until that element has been scanned. • Returns the element as the output.

Code	#include <stdio.h>
	#define MAX 10
	Void lsearch(int list[], int n, int element)
	{
	int i, flag = 0;
	for(i=0;i<n;i++)
	if(list[i] == element)
	{
	printf(" The element whose value is %d is present at position %d in list\n", element, i);
	flag =1;
	break;
	}
	if(flag == 0)
	printf("The element whose value is %d is not present in the list\n", element);
	}
	void readlist(int list[],int n)
	{
	int i;
	printf("Enter the elements\n");
	for(i=0;i<n;i++)
	scanf("%d", &list[i]);
	}
	void printlist(int list[],int n)
	{
	int i;
	printf("The elements of the list are: \n");
	for(i=0;i<n;i++)
	printf("%d\t", list[i]);
	}
	void main()
	{

```
int list[MAX], n, element;
printf ("Enter the number of elements in the list max = 10\n");
scanf ("%d",&n);
readlist (list,n);
printf ("\n The list before sorting is:\n");
printlist (list, n);
printf ("\n Enter the element to be searched\n");
scanf ("%d", &element);
lsearch (list, n, element);
}
```

Problem Statement	Write a program to implement Binary Search.
Concept	• Find the middle element • Check the middle element with the element to be found • If the middle element is equal to that element, then it will provide the output. • If the value is not same, then it will check whether the middle element value is less than or greater than the element to be found. • If the value is less than that element, then the search will start with the elements next to the middle element. • If the value is high than that element, then the search will start with the elements before to the middle element. • This process continues, until that particular element has been found.

Code	#include<stdio.h>
	#include<conio.h>
	void main()
	{
	int first, last, middle, size, i, sElement, list[100];
	clrscr();
	printf("Enter the size of the list: ");
	scanf("%d",&size);
	printf("Enter %d integer values in Ascending order\n", size);
	for (i = 0; i < size; i++)
	scanf("%d",&list[i]);
	printf("Enter value to be search: ");
	scanf("%d", &sElement);
	first = 0;
	last = size - 1;
	middle = (first+last)/2;
	while (first <= last)
	{
	if (list[middle] < sElement)
	first = middle + 1;
	else if (list[middle] == sElement)
	{
	printf("Element found at index %d.\n",middle);
	break;
	}
	else
	last = middle - 1;

```
    middle = (first + last)/2;
  }
  if (first > last)
    printf("Element Not found in the list.");
  getch();
}
```

Problem Statement	Write a program to implement Bubble Sort.
Concept	• Starting with the first element (index = 0), compare the current element with the next element of the array. • If the current element is greater than the next element of the array, swap them. • If the current element is less than the next element, move to the next element. Repeat Step 1.

Code	#include <stdio.h>
	#define MAX 10
	void swap(int *x, int *y)
	{
	int temp; temp = *x;
	*x = *y;
	*y = temp;
	}
	void bsort(int list[], int n)
	{
	int i,j;
	for(i=0;i<(n-1);i++)
	for(j=0;j<(n-(i+1));j++)
	if(list[j] > list[j+1])
	swap(&list[j],&list[j+1]);
	}
	void readlist(int list[],int n)
	{
	int i;
	printf("Enter the elements\n");
	for(i=0;i<n;i++)
	scanf("%d",&list[i]);
	}
	void printlist(int list[],int n)
	{
	int i;
	printf("The elements of the list are: \n"); for(i=0;i<n;i++)
	printf("%d\t",list[i]);
	}
	void main()
	{

```
int list[MAX], n;
printf("Enter the number of elements in the list max = 10\n");
scanf("%d",&n);
readlist(list,n);
printf("The list before sorting is:\n");
 printlist(list,n);
bsort(list,n);
printf("The list after sorting is:\n");
 printlist(list,n);
 }
```

Problem Statement	Write a program to implement Insertion Sort.
Concept	• If it is the first element, it is already sorted, return 1. • Pick next element. • Compare with all elements in the sorted sub-list. • Shift all the elements in the sorted sub-list that is greater than the value to be sorted. • Insert the value. • Repeat until list is sorted.

Code	#include<stdio.h>
	#include<conio.h>
	void main()
	{
	int size, i, j, temp, list[100];
	printf("Enter the size of the list: ");
	scanf("%d", &size);
	printf("Enter %d integer values: ", size);
	for (i = 0; i < size; i++)
	scanf("%d", &list[i]);
	for (i = 1; i < size; i++) {
	temp = list[i];
	j = i - 1;
	while ((temp < list[j]) && (j >= 0)) {
	list[j + 1] = list[j];
	j = j - 1;
	}
	list[j + 1] = temp;
	}
	printf("List after Sorting is: ");
	for (i = 0; i < size; i++)
	printf(" %d", list[i]);
	getch();
	}

Problem Statement	Write a program to implement Selection Sort
Concept	• Set MIN to location 0 • Search the minimum element in the list • Swap with value at location MIN • Increment MIN to point to next element • Repeat until list is sorted

Code

```
#include<stdio.h>
#include<conio.h>

void main(){

 int size,i,j,temp,list[100];
 clrscr();

 printf("Enter the size of the List: ");
 scanf("%d",&size);

 printf("Enter %d integer values: ",size);
 for(i=0; i<size; i++)
    scanf("%d",&list[i]);


 for(i=0; i<size; i++){
    for(j=i+1; j<size; j++){
        if(list[i] > list[j])
            {
           temp=list[i];
           list[i]=list[j];
           list[j]=temp;
        }
    }
 }

 printf("List after sorting is: ");
 for(i=0; i<size; i++)
    printf(" %d",list[i]);
 getch();
}
```

Problem Statement	Write a program to implement Quick Sort.
Concept	• Consider the first element of the list as pivot (i.e., Element at first position in the list). • Define two variables i and j. Set i and j to first and last elements of the list respectively. • Increment i until list[i] > pivot then stop.(step-3) • Decrement j until list[j] < pivot then stop.(step-4) • If i < j then exchange list[i] and list[j].(step-5) • Repeat steps 3,4 & 5 until i > j. • Exchange the pivot element with list[j] element.

Code	#include<stdio.h>
	#include<conio.h>
	
	void quickSort(int [10],int,int);
	
	void main(){
	  int list[20],size,i;
	
	  printf("Enter size of the list: ");
	  scanf("%d",&size);
	
	  printf("Enter %d integer values: ",size);
	  for(i = 0; i < size; i++)
	  scanf("%d",&list[i]);
	
	  quickSort(list,0,size-1);
	
	  printf("List after sorting is: ");
	  for(i = 0; i < size; i++)
	  printf(" %d",list[i]);
	
	  getch();
	}
	
	void quickSort(int list[10],int first,int last){

```
    int pivot,i,j,temp;

     if(first < last){
        pivot = first;
        i = first;
        j = last;

        while(i < j){
           while(list[i] <= list[pivot] && i < last)
              i++;
           while(list[j] > list[pivot])
              j--;
           if(i <j){
               temp = list[i];
               list[i] = list[j];
               list[j] = temp;
           }
        }

        temp = list[pivot];
        list[pivot] = list[j];
        list[j] = temp;
        quickSort(list,first,j-1);
        quickSort(list,j+1,last);
    }
}
```

Problem Statement	Write a program to implement a stack using an array.
Concept	• Include all the header files which are used in the program and define a constant 'SIZE' with specific value. • Declare all the functions used in stack implementation. • Create a one dimensional array with fixed size (int stack[SIZE]). • Define a integer variable 'top' and initialize with '-1'. (int top = -1). • In main method, display menu with list of operations and make suitable function calls to perform operation selected by the user on the stack. **push(value) - Inserting value into the stack** Step 1 - Check whether stack is FULL. (top == SIZE-1) Step 2 - If it is FULL, then display "Stack is FULL!!! Insertion is not possible!!!" and terminate the function. Step 3 - If it is NOT FULL, then increment top value by one (top++) and set stack[top] to value (stack[top] = value). **pop() - Delete a value from the Stack** Step 1 - Check whether stack is EMPTY. (top == -1) Step 2 - If it is EMPTY, then display "Stack is EMPTY!!! Deletion is not possible!!!" and terminate the function. Step 3 - If it is NOT EMPTY, then delete stack [top] and decrement top value by one (top--). **display() - Displays the elements of a Stack** Step 1 - Check whether stack is EMPTY. (top == -1) Step 2 - If it is EMPTY, then display "Stack is EMPTY!!!" and terminate the function. Step 3 - If it is NOT EMPTY, then define a variable 'i' and initialize with top. Display stack[i] value and decrement i value by one (i--). Step 3 - Repeat above step until i value becomes '0'.

Code	#include<stdio.h>
	#include<conio.h>
	
	#define SIZE 10
	
	void push(int);
	void pop();
	void display();

```
int stack[SIZE], top = -1;

void main()
{
  int value, choice;
  clrscr();
  while(1){
    printf("\n\n***** MENU *****\n");
    printf("1. Push\n2. Pop\n3. Display\n4. Exit");
    printf("\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice){
          case 1: printf("Enter the value to be insert: ");
                  scanf("%d",&value);
                  push(value);
                  break;
          case 2: pop();
                  break;
          case 3: display();
                  break;
          case 4: exit(0);
          default: printf("\nWrong selection!!! Try again!!!");
    }
  }
}
void push(int value){
  if(top == SIZE-1)
    printf("\nStack is Full!!! Insertion is not possible!!!");
  else{
    top++;
    stack[top] = value;
    printf("\nInsertion success!!!");
  }
}
void pop(){
  if(top == -1)
    printf("\nStack is Empty!!! Deletion is not possible!!!");
```

```
  else{
    printf("\nDeleted : %d", stack[top]);
    top--;
  }
}
void display(){
  if(top == -1)
    printf("\nStack is Empty!!!");
  else{
    int i;
    printf("\nStack elements are:\n");
    for(i=top; i>=0; i--)
          printf("%d\n",stack[i]);
  }
}
```

Problem Statement	Write a program for implementation of a stack using the linked list.
Concept	To implement a stack using a linked list, we need to set the following things before implementing actual operations. • Include all the header files which are used in the program. And declare all the user defined functions. • Define a 'Node' structure with two members data and next. • Define a Node pointer 'top' and set it to NULL. • Implement the main method by displaying Menu with list of operations and make suitable function calls in the main method. **push(value) - Inserting an element into the Stack** We can use the following steps to insert a new node into the stack... Step 1 - Create a newNode with given value. Step 2 - Check whether stack is Empty (top == NULL) Step 3 - If it is Empty, then set newNode → next = NULL. Step 4 - If it is Not Empty, then set newNode → next = top. Step 5 - Finally, set top = newNode. **pop() - Deleting an Element from a Stack** We can use the following steps to delete a node from the stack... Step 1 - Check whether stack is Empty (top == NULL). Step 2 - If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the function Step 3 - If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'. Step 4 - Then set 'top = top → next'. Step 5 - Finally, delete 'temp'. (free(temp)). **display() - Displaying stack of elements** We can use the following steps to display the elements (nodes) of a stack... Step 1 - Check whether stack is Empty (top == NULL). Step 2 - If it is Empty, then display 'Stack is Empty!!!' and terminate the function. Step 3 - If it is Not Empty, then define a Node pointer 'temp' and initialize with top. Step 4 - Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to the first node in the stack. (temp → next

	!= NULL). Step 5 - Finally! Display 'temp → data ---> NULL'.

Code	#include<stdio.h>
	#include<conio.h>
	
	struct Node
	{
	  int data;
	  struct Node *next;
	}*top = NULL;
	
	void push(int);
	void pop();
	void display();
	
	void main()
	{
	  int choice, value;
	  clrscr();
	  printf("\n:: Stack using Linked List ::\n");
	  while(1){
	  printf("\n****** MENU ******\n");
	  printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
	  printf("Enter your choice: ");
	  scanf("%d",&choice);
	  switch(choice){
	  case 1: printf("Enter the value to be insert: ");
	  scanf("%d", &value);
	  push(value);
	  break;
	  case 2: pop(); break;
	  case 3: display(); break;
	  case 4: exit(0);
	  default: printf("\nWrong selection!!! Please try again!!!\n");
	  }
	  }
	}
	void push(int value)
	{
	  struct Node *newNode;
	  newNode = (struct Node*)malloc(sizeof(struct Node));
	  newNode->data = value;
	  if(top == NULL)
	  newNode->next = NULL;

```
  else
    newNode->next = top;
  top = newNode;
  printf("\nInsertion is Success!!!\n");
}
void pop()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct Node *temp = top;
    printf("\nDeleted element: %d", temp->data);
    top = temp->next;
    free(temp);
  }
}
void display()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct Node *temp = top;
    while(temp->next != NULL){
          printf("%d--->",temp->data);
          temp = temp -> next;
    }
    printf("%d--->NULL",temp->data);
  }
}
```

Problem Statement	Write a program to convert an infix expression to prefix form.
Concept	• In an infix expression, a binary operator separates its operands (a unary operator precedes its operand). In a postfix expression, the operands of an operator precede the operator. In a prefix expression, the operator precedes its operands. Like postfix, a prefix expression is parenthesis-free, that is, any infix expression can be unambiguously written in its prefix equivalent without the need for parentheses. • To convert an infix expression to reverse-prefix, it is scanned from right to left. A queue of operands is maintained noting that the order of operands in infix and prefix remains the same. Thus, while scanning the infix expression, whenever an operand is encountered, it is pushed in a queue. If the scanned element is a right parenthesis (')'), it is pushed in a stack of operators. If the scanned element is a left parenthesis ('('), the queue of operands is emptied to the prefix output, followed by the popping of all the operators up to, but excluding, a right parenthesis in the operator stack. • If the scanned element is an arbitrary operator o, then the stack of operators is checked for operators with a greater priority then o. Such operators are popped and written to the prefix output after emptying the operand queue. The operator o is finally pushed to the stack. • When the scanning of the infix expression is complete, first the operand queue, and then the operator stack, are emptied to the prefix output. Any whitespace in the infix input is ignored. Thus the prefix output can be reversed to get the required prefix expression of the infix input.

Code	#include <stdio.h>
	#include <string.h>
	#include <ctype.h>
	#define N 80
	typedef enum {FALSE, TRUE} bool;
	
	#include "stack.h"
	#include "queue.h"
	
	#define NOPS 7
	
	char operators [] = "()^/*+-";
	int priorities[] = {4,4,3,2,2,1,1}; char associates[] = " RLLLL";
	
	char t[N]; char *tptr = t;
	int getIndex(char op) {
	inti;
	for(i=0; i<NOPS; ++i)

```
if( operators[i] == op ) return i;
return -1;
}
int getPriority( char op ) {
return priorities[ getIndex(op) ];
}
char getAssociativity( char op ) {
return associates[ getIndex(op) ];
}

void processOp( char op, queue *q, stack *s ) {
switch(op){
case')':
printf( "\t S pushing )...\n" ); sPush( s, op );
break; case '(':
while( !qEmpty(q) ) {
*tptr++ = qPop(q);
printf( "\tQ popping %c...\n", *(tptr-1) );
}
while( !sEmpty(s) ) {
char popop = sPop(s);
printf( "\tS popping %c...\n", popop );
 if( popop == ')' )
break;
*tptr++ = popop;
}
break;
 default:
{
intpriop;
chartopop;
intpritop;
charasstop;
while( !sEmpty(s) ){
priop = getPriority(op);
topop = sTop(s);
pritop = getPriority(topop);
 asstop = getAssociativity(topop);
if( pritop < priop || (pritop == priop && asstop == 'L')
|| topop == ')' )
break;
while( !qEmpty(q) ) {
*tptr++ = qPop(q);
printf( "\tQ popping %c...\n", *(tptr-1) );
```

```
}
*tptr++ = sPop(s);
printf( "\tS popping %c...\n", *(tptr-1) );
}
printf( "\tS pushing %c...\n", op ); sPush( s, op );
break;
}
}
}
bool isop( char op ) {
/*
* is op an operator?
*/
return (getIndex(op) != -1);
}

char *in2pre( char *str ) { /*
char *sptr;
queue q = {NULL};
stack s = NULL;
char *res = (char *)malloc( N*sizeof(char) );
char *resptr = res;
tptr = t;
for( sptr=str+strlen(str)-1; sptr!=str-1; -sptr )
 {
printf( "processing %c tptr-t=%d...\n", *sptr, tptr-t);
 if( isalpha(*sptr) )
qPush( &q, *sptr );
else if(isop(*sptr))
processOp( *sptr, &q, &s);
else if(isspace(*sptr))
;
 else {
}
}
fprintf( stderr, "ERROR:invalid char %c.\n", *sptr ); return "";
while( !qEmpty(&q) ) {
*tptr++ = qPop(&q);
printf( "\tQ popping %c...\n", *(tptr-1) );
}
while( !sEmpty(&s) ) {
*tptr++ = sPop(&s);
printf( "\tS popping %c...\n", *(tptr-1) );
}
```

```
*tptr = 0;
printf( "t=%s.\n", t );
for( -tptr; tptr!=t-1; -tptr ) {
*resptr++ = *tptr;
}
*resptr = 0;

return res;
}

int main() {
char s[N];

puts( "enter infix freespaces max 80." ); gets(s);
while(*s){
puts( in2pre(s) ); gets(s);
}
return0;
}
```

Problem Statement	Write a program to implement a queue by using an array.
Concept	Before we implement actual operations, first follow the below steps to create an empty queue. • Include all the header files which are used in the program and define a constant 'SIZE' with specific value. • Declare all the user defined functions which are used in queue implementation. • Create a one dimensional array with above defined SIZE (int queue[SIZE]) • Define two integer variables 'front' and 'rear' and initialize both with '-1'. (int front = -1, rear = -1) • Then implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on queue. **enQueue(value) - Inserting value into the queue** Step 1 - Check whether queue is FULL. (rear == SIZE-1) Step 2 - If it is FULL, then display "Queue is FULL!!! Insertion is not possible!!!" and terminate the function. Step 3 - If it is NOT FULL, then increment rear value by one (rear++) and set queue[rear] = value. **deQueue() - Deleting a value from the Queue** Step 1 - Check whether queue is EMPTY. (front == rear) Step 2 - If it is EMPTY, then display "Queue is EMPTY!!! Deletion is not possible!!!" and terminate the function. Step 3 - If it is NOT EMPTY, then increment the front value by one (front ++). Then display queue[front] as deleted element. Then check whether both front and rear are equal (front == rear), if it TRUE, then set both front and rear to '-1' (front = rear = -1). **display() - Displays the elements of a Queue** We can use the following steps to display the elements of a queue... Step 1 - Check whether queue is EMPTY. (front == rear) Step 2 - If it is EMPTY, then display "Queue is EMPTY!!!" and terminate the function. Step 3 - If it is NOT EMPTY, then define an integer variable 'i' and set 'i = front+1'. Step 4 - Display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until 'i' value reaches to rear (i <= rear)

Code

```
#include<stdio.h>
#include<conio.h>
#define SIZE 10

void enQueue(int);
void deQueue();
void display();

int queue[SIZE], front = -1, rear = -1;

void main()
{
  int value, choice;
  clrscr();
  while(1){
    printf("\n\n***** MENU *****\n");
    printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");
    printf("\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice){
          case 1: printf("Enter the value to be insert: ");
                  scanf("%d",&value);
                  enQueue(value);
                  break;
          case 2: deQueue();
                  break;
          case 3: display();
                  break;
          case 4: exit(0);
          default: printf("\nWrong selection!!! Try again!!!");
    }
  }
}
void enQueue(int value){
  if(rear == SIZE-1)
    printf("\nQueue is Full!!! Insertion is not possible!!!");
  else{
    if(front == -1)
          front = 0;
    rear++;
    queue[rear] = value;
    printf("\nInsertion success!!!");
  }
}
```

```
void deQueue(){
  if(front == rear)
    printf("\nQueue is Empty!!! Deletion is not possible!!!");
  else{
    printf("\nDeleted : %d", queue[front]);
    front++;
    if(front == rear)
          front = rear = -1;
  }
}
void display(){
  if(rear == -1)
    printf("\nQueue is Empty!!!");
  else{
    int i;
    printf("\nQueue elements are:\n");
    for(i=front; i<=rear; i++)
          printf("%d\t",queue[i]);
  }
}
```

Problem Statement	Write a program for implementation of a Queue using the linked list
Concept	To implement queue using linked list, we need to set the following things before implementing actual operations. • Include all the header files which are used in the program. And declare all the user defined functions. • Define a 'Node' structure with two members data and next. • Define two Node pointers 'front' and 'rear' and set both to NULL. • Implement the main method by displaying Menu of list of operations and make suitable function calls in the main method to perform user selected operation. **enQueue(value) - Inserting an element into the Queue** We can use the following steps to insert a new node into the queue... Step 1 - Create a newNode with given value and set 'newNode → next' to NULL. Step 2 - Check whether queue is Empty (rear == NULL) Step 3 - If it is Empty then, set front = newNode and rear = newNode. Step 4 - If it is Not Empty then, set rear → next = newNode and rear = newNode. **deQueue() - Deleting an Element from Queue** We can use the following steps to delete a node from the queue... Step 1 - Check whether queue is Empty (front == NULL). Step 2 - If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function Step 3 - If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'. Step 4 - Then set 'front = front → next' and delete 'temp' (free(temp)). **display() - Displaying the elements of Queue** We can use the following steps to display the elements (nodes) of a queue... Step 1 - Check whether queue is Empty (front == NULL). Step 2 - If it is Empty then, display 'Queue is Empty!!!' and terminate the function. Step 3 - If it is Not Empty then, define a Node pointer 'temp' and initialize with front. Step 4 - Display 'temp → data --->' and move it to the next node. Repeat

	the same until 'temp' reaches to 'rear' (temp → next != NULL). Step 5 - Finally! Display 'temp → data ---> NULL'.

Code	#include<stdio.h>
	#include<conio.h>
	
	struct Node
	{
	  int data;
	  struct Node *next;
	}*front = NULL,*rear = NULL;
	
	void insert(int);
	void delete();
	void display();
	
	void main()
	{
	  int choice, value;
	  clrscr();
	  printf("\n:: Queue Implementation using Linked List ::\n");
	  while(1){
	  printf("\n****** MENU ******\n");
	  printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
	  printf("Enter your choice: ");
	  scanf("%d",&choice);
	  switch(choice){
	  case 1: printf("Enter the value to be insert: ");
	  scanf("%d", &value);
	  insert(value);
	  break;
	  case 2: delete(); break;
	  case 3: display(); break;
	  case 4: exit(0);
	  default: printf("\nWrong selection!!! Please try again!!!\n");
	  }
	  }
	}
	void insert(int value)
	{
	  struct Node *newNode;
	  newNode = (struct Node*)malloc(sizeof(struct Node));
	  newNode->data = value;
	  newNode -> next = NULL;
	  if(front == NULL)

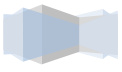
```
    front = rear = newNode;
  else{
    rear -> next = newNode;
    rear = newNode;
  }
  printf("\nInsertion is Success!!!\n");
}
void delete()
{
  if(front == NULL)
    printf("\nQueue is Empty!!!\n");
  else{
    struct Node *temp = front;
    front = front -> next;
    printf("\nDeleted element: %d\n", temp->data);
    free(temp);
  }
}
void display()
{
  if(front == NULL)
    printf("\nQueue is Empty!!!\n");
  else{
    struct Node *temp = front;
    while(temp->next != NULL){
        printf("%d--->",temp->data);
        temp = temp -> next;
    }
    printf("%d--->NULL\n",temp->data);
  }
}
```

Problem Statement	Write a program to implement a priority queue
Concept	In the implementation of priority queues required to be maintained by the scheduler of an operating system. It is a queue in which each element has a priority value and the elements are required to be inserted in the queue in decreasing order of priority. This requires a change in the function that is used for insertion of an element into the queue. No change is required in the delete function.

Code	# include <stdio.h>
	# include <stdlib.h>
	structnode
	{
	int data;
	int priority; struct node *link;
	};
	void insert(struct node **front, struct node **rear, int value, int priority)
	{
	struct node *temp,*temp1;
	temp=(struct node *)malloc(sizeof(struct node));
	if(temp==NULL)
	{
	printf("No Memory available Error\n");
	exit(0);
	}
	temp->data = value;
	temp->priority = priority; temp->link=NULL;
	if(*rear == NULL)
	{
	*rear = temp;
	*front = *rear;
	}
	else
	{
	if((*front)->priority < priority)
	{
	}
	else
	
	temp->link = *front;

```
*front = temp;
if( (*rear)->priority > priority)
{
(*rear)->link = temp;
*rear = temp;
}
else
{
temp1 = *front;
while((temp1->link)->priority >= priority)
 temp1=temp1->link; temp->link = temp1->link;
temp1->link = temp;
}
}
void delete(struct node **front, struct node **rear, int *value, int*priority)
{
struct node *temp;
if((*front == *rear) && (*rear == NULL))
{
printf(" The queue is empty cannot delete Error\n");
exit(0);
}
*value = (*front)->data;
*priority = (*front)->priority; temp = *front;
*front = (*front)->link; if(*rear == temp)
*rear = (*rear)->link; free(temp);
}

void main()
{
struct node *front=NULL,*rear = NULL; int n,value, priority;
do



{
do
{
printf("Enter the element to be inserted and its priority\n");
```

```
scanf("%d%d",&value,&priority);
insert(&front,&rear,value,priority);
printf("Enter 1 to continue\n"); scanf("%d",&n);
} while(n == 1);

printf("Enter 1 to delete an element\n");
scanf("%d",&n);
while( n == 1)
{
delete(&front,&rear,&value,&priority);
printf("The value deleted is %d\ and its priority is %d \n", value,priority);
printf("Enter 1 to delete an element\n");
 scanf("%d",&n);
}
printf("Enter 1 to delete anelement\n");
scanf("%d",&n);
}          while( n ==1)
}
```

Problem Statement	Write a program for building and printing the elements of the linked list.
Concept	We can use the following steps to display the elements of a single linked list... • Check whether list is Empty (head == NULL) • If it is Empty then, display 'List is Empty!!!' and terminate the function. • If it is Not Empty then, define a Node pointer 'temp' and initialize with head. • Keep displaying temp → data with an arrow (--->) until temp reaches to the last node • Finally display temp → data with arrow pointing to NULL (temp → data ---> NULL).

Code	# include <stdio.h>
	# include <stdlib.h>
	structnode
	{
	int data;
	struct node *link;
	};
	struct node *insert(struct node *p, int n)
	{
	struct node *temp;
	if(p==NULL)
	{
	p=(struct node *)malloc(sizeof(struct node));
	if(p==NULL)
	{
	printf("Error\n");
	exit(0);
	}
	p-> data = n;
	p- > link = p;
	}
	else
	{
	temp = p;
	while (temp-> link != p) temp = temp-> link;
	temp-> link = (struct node *)malloc(sizeof(struct node));
	if(temp -> link == NULL)
	{
	printf("Error\n"); exit(0);
	}

```
temp = temp-> link; temp-> data = n; temp-> link = p;
}
return (p);
}
void printlist ( struct node *p )
{
struct node *temp; temp = p;
printf("The data values in the list are\n");
if(p!= NULL)
{
do
{
printf("%d\t",temp->data);
temp=temp->link;
} while (temp!= p);
}
else
printf("The list is empty\n");
}
void main()
{
int n; int x;
struct node *start = NULL ;
printf("Enter the nodes to be created \n");
 scanf("%d",&n);
while ( n -- > 0)
{
printf( "Enter the data values to be placed in anode\n");
scanf("%d",&x);
start = insert ( start, x );
}
printf("The created list is\n");
printlist ( start );
}
```

Problem Statement	Write a program to reverse the list.
Concept	• To reverse a list, maintain a pointer each to the previous and the next node. • Then make the link field of the current node point to the previous, make the previous equal to the current, and the current equal to the next

Code	# include <stdio.h>
	# include <stdlib.h>
	structnode
	{
	int data;
	struct node *link;
	};
	struct node *insert(struct node *p, int n)
	{
	struct node *temp; if(p==NULL)
	{
	p=(struct node *)malloc(sizeof(struct node));
	if(p==NULL)
	{
	printf("Error\n");
	exit(0);
	}
	p-> data = n;
	p-> link = NULL;
	}
	else
	{
	temp = p;
	while (temp-> link!= NULL) temp = temp-> link;
	temp-> link = (struct node *)malloc(sizeof(struct node));
	if(temp -> link == NULL)
	{
	printf("Error\n");
	exit(0);
	}
	temp = temp-> link;
	temp-> data = n;
	temp-> link = null;
	}
	return(p);
	}
	
	void printlist (struct node *p)

```
{
printf("The data values in the listare\n");
while (p!=NULL)
{
printf("%d\t",p-> data);
p = p->link;
}
}
 struct node *reverse(struct node *p)
{
struct node *prev, *curr;
 prev = NULL;
curr = p;
while (curr != NULL)
{
p = p-> link;
curr-> link = prev;
prev = curr;
curr = p;
}
return(prev);
}
struct node *sortlist(struct node*p)
{
struct node *temp1,*temp2,*min,*prev,*q;
 q = NULL;
while(p != NULL)
{
prev = NULL;
min = temp1 = p;
temp2 = p ->link;
while ( temp2 != NULL )
{
if(min -> data > temp2 -> data)
{
min = temp2;
prev = temp1;
}
temp1 = temp2;
temp2 = temp2-> link;
}
if(prev == NULL)
p = min -> link;
else
```

```
prev -> link = min -> link;
min -> link = NULL; if( q == NULL)
q = min;
else
{
temp1 = q;
while( temp1 -> link != NULL)
temp1 = temp1 -> link;
temp1 -> link = min;
}
}
return (q);
}
void main()
{
int n;
 int x;
struct node *start = NULL ;
printf("Enter the nodes to be created \n");
 scanf("%d",&n);
while ( n- > 0 )
{
printf( "Enter the data values to be placed in a node\n");
scanf("%d",&x);
start = insert ( start,x);
}
printf("The created list is\n");
 printlist ( start);
start = sortlist(start);
printf("The sorted list is\n");
 printlist ( start);
start = reverse(start);
 printf("The reversed listis\n");
printlist ( start);
}
```

Problem Statement	Write a program to delete the specific node in a singly linked list.
Concept	• Check whether list is Empty (head == NULL) • If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function. • If it is Not Empty then, define two Node pointers 'temp1' and 'temp2' and initialize 'temp1' with head. • Keep moving the temp1 until it reaches to the exact node to be deleted or to the last node. And every time set 'temp2 = temp1' before moving the 'temp1' to its next node. • If it is reached to the last node then display 'Given node not found in the list! Deletion not possible!!!'. And terminate the function. • If it is reached to the exact node which we want to delete, then check whether list is having only one node or not • If list has only one node and that is the node to be deleted, then set head = NULL and delete temp1 (free(temp1)). • If list contains multiple nodes, then check whether temp1 is the first node in the list (temp1 == head). • If temp1 is the first node then move the head to the next node (head = head → next) and delete temp1. • If temp1 is not first node then check whether it is last node in the list (temp1 → next == NULL). • If temp1 is last node then set temp2 → next = NULL and delete temp1 (free(temp1)). • If temp1 is not first node and not last node then set temp2 → next = temp1 → next and delete temp1 (free(temp1))

Code	# include <stdio.h>
	# include<stdlib.h>
	struct node *delet (struct node *, int);
	int length (struct node *);
	struct node
	{
	int data;
	struct node *link;
	};
	struct node *insert(struct node *p, int n)
	{
	struct node *temp; if(p==NULL)
	{
	p=(struct node *)malloc(sizeof(struct node)); if(p==NULL)
	{
	printf("Error\n");
	exit(0);

```
}
p-> data = n;
p-> link = NULL;
}
else
{
temp = p;
while (temp-> link != NULL) temp = temp-> link;
temp-> link = (struct node *)malloc(sizeof(struct node));
if(temp -> link == NULL)
{
printf("Error\n");
exit(0);
}
temp = temp-> link;
 temp-> data = n;
temp-> link = NULL;
}
return (p);
}

void printlist ( struct node *p )
{
printf("The data values in the listare\n");
while (p!=NULL)
{
printf("%d\t",p-> data); p = p->link;
}
}

void main()
{
int n;
 int x;
struct node *start = NULL;
printf("Enter the nodes to be created \n");
 scanf("%d",&n);
while ( n- > 0 )
{
printf( "Enter the data values to be placed in a node\n");
 scanf("%d",&x);
start = insert ( start, x );
}
printf(" The list before deletion id\n");
```

```
printlist ( start );
printf("% \n Enter the node no \n");
scanf ( " %d",&n);
start = delet (start , n );
printf(" The list after deletion is\n");
printlist ( start );
}

struct node *delet ( struct node *p, int node_no )
{

struct node *prev, *curr ;
 int i;

if (p == NULL )
{
printf("There is no node to be deleted \n");
}
else
{
if ( node_no > length (p))
{
}
else
{
printf("Error\n");
prev = NULL; curr = p;
i = 1 ;
while ( i < node_no )
{
prev = curr;
curr = curr-> link;
 i = i+1;
}
if ( prev == NULL )
{
}
else
{

p = curr -> link; free ( curr );
prev -> link = curr -> link ; free ( curr );
}
}
```

```
}
return(p);
}
int length ( struct node *p )
{
int count = 0 ;
 while ( p != NULL )
{
count++;
p = p->link;
}
return ( count ) ;
}
```

Problem Statement	Write a program to insert a node after the specific node in a singly linked list.
Concept	We can use the following steps to insert a new node after a node in the single linked list... • Create a newNode with given value. • Check whether list is Empty (head == NULL) • If it is Empty then, set newNode → next = NULL and head = newNode. • If it is Not Empty then, define a node pointer temp and initialize with head. • Keep moving the temp to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode). • Every time check whether temp is reached to last node or not. If it is reached to last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp to next node. • Finally, Set 'newNode → next = temp → next' and 'temp → next = newNode'

Code	# include <stdio.h>
	# include<stdlib.h>
	int length (struct node *);
	struct node
	{
	int data;
	struct node *link;
	};
	struct node *insert(struct node *p, int n)
	{
	struct node *temp; if(p==NULL)
	{
	p=(struct node *)malloc(sizeof(struct node));
	if(p==NULL)
	{
	printf("Error\n");
	exit(0);
	}
	p-> data = n;
	p-> link = NULL;
	}
	else
	{

```
temp = p;
while (temp-> link != NULL) temp = temp-> link;
temp-> link = (struct node *)malloc(sizeof(struct node));
 if(temp -> link == NULL)
{
printf("Error\n");
exit(0);
}
temp = temp-> link; temp-> data = n;
 temp-> link= NULL;
}
return (p);
}
struct node * newinsert ( struct node *p, int node_no, int value )
{
struct node *temp, * temp1;
 int i;
if ( node_no <= 0 || node_no > length (p))
{
printf("Error! the specified node does not exist\n");
exit(0);
}
if ( node_no == 0)
{
temp = ( struct node * )malloc ( sizeof ( struct node ));
 if ( temp == NULL )
{
printf( " Cannot allocate \n");
 exit (0);
}


}
else
{

temp -> data = value; temp -> link = p;
p = temp ;
temp = p ;
 i = 1;
while ( i < node_no )
{
i = i+1;
temp = temp-> link ;
```

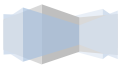
```
}
temp1 = ( struct node * )malloc ( sizeof(struct node));
 if ( temp == NULL )
{
printf ("Cannot allocate \n");
 exit(0)
}
temp1 -> data = value ;  temp1 -> link = temp -> link; temp -> link =temp1;
}
return (p);
}
void printlist ( struct node *p )
{
printf("The data values in the listare\n");
 while (p!=NULL)
{
printf("%d\t",p-> data);
 p = p->link;
}
}
void main ()
{
int n;
 int x;
struct node *start = NULL;
printf("Enter the nodes to be created \n");
 scanf("%d",&n);
while ( n- > 0 )
{
printf( "Enter the data values to be placed in a node\n");
scanf("%d",&x);
start = insert ( start, x );
}
printf(" The list before deletion is\n");
 printlist ( start );
printf(" \n Enter the node no after which the insertion is to be done\n");
scanf ( " %d",&n);
printf("Enter the value of the node\n");
scanf("%d",&x);
start = newinsert(start,n,x);
printf("The list after insertion is \n");
 printlist(start);
}
```

Problem Statement	Write a program to delete a linked list
Concept	• Erasing a linked list involves traversing the list starting from the first node, freeing the storage allocated to the nodes, and then setting the pointer to the list to NULL. • Erasing a list is to mark all the nodes of the list to be erased as free nodes without actually freeing the storage of these nodes. That means to maintain this list, a list of free nodes, so that if a new node is required it can be obtained from this list of free nodes.

Code	# include <stdio.h>
	# include <stdlib.h>
	structnode
	{
	int data;
	struct node *link;
	};
	struct node *insert(struct node *, int);
	void erase(struct node **,struct node **);
	void printlist (struct node *);
	void erase (struct node **p, struct node **free)
	{
	struct node *temp; temp = *p;
	while (temp->link != NULL) temp = temp ->link;
	temp->link = (*free);
	*free = *p;
	*p = NULL;
	}
	struct node *insert(struct node *p, int n)
	{
	struct node *temp;
	if(p==NULL)
	{
	p=(struct node *)malloc(sizeof(struct node));
	if(p==NULL)
	{
	printf("Error\n");
	exit(0);
	}
	p-> data = n;
	p-> link =NULL;
	}
	else
	{
	temp = p;

```
while (temp-> link!= NULL) temp = temp-> link;
temp-> link = (struct node *)malloc(sizeof(struct node));
 if(temp -> link == NULL)
{
printf("Error\n");
exit(0);
}
temp = temp-> link;
 temp-> data = n;
temp-> link = NULL;
}
return (p);
}

void printlist ( struct node *p )
{
printf("The data values in the list are\n");
while (p!=NULL)
{
printf("%d\t",p-> data);
p = p->link;
}
}
void main()
{
int n;
int x;
struct node *start = NULL ;
 struct node *free=NULL;

printf("Enter the number of nodes in the initial free list \n");
scanf("%d",&n);
 while ( n-- > 0 )
{
printf( "Enter the data values to be placed in a node\n");
scanf("%d",&x);
free = insert ( free,x);
}
printf("Enter the number of nodes in the list to be created for erasing \n");
scanf("%d",&n);
while ( n-- > 0 )
{
printf( "Enter the data values to be placed in a node\n");
scanf("%d",&x);
```

```
start = insert ( start,x);
}
printf("The free list islist is:\n");
 printlist ( free );
printf("The list to be erased is:\n");
printlist ( start); erase(&start,&free);
printf("The free list after adding all the nodes from the list to be erased is:\n");
 printlist ( free );
}
```

Problem Statement	Write a program for building and printing the elements of a doubly linked list.
Concept	• Check whether list is Empty (head == NULL) • If it is Empty, then display 'List is Empty!!!' and terminate the function. • If it is not Empty, then define a Node pointer 'temp' and initialize with head. • Display 'NULL <--- '. • Keep displaying temp → data with an arrow (<===>) until temp reaches to the last node • Finally, display temp → data with arrow pointing to NULL (temp → data ---> NULL).

Code	# include <stdio.h>
	# include <stdlib.h>
	structdnode
	{
	int data;
	struct dnode *left, *right;
	};
	struct dnode *insert(struct dnode *p, struct dnode **q, int n)
	{
	struct dnode *temp;
	if(p==NULL)
	{
	p=(struct dnode *)malloc(sizeof(struct dnode));
	if(p==NULL)
	{
	printf("Error\n");
	exit(0);
	}
	p->data =n;
	p-> left = p->right =NULL;
	*q =p;
	}
	else
	{
	temp = (struct dnode *)malloc(sizeof(struct dnode));
	if(temp ==NULL)
	{
	printf("Error\n");
	exit(0);
	}
	temp->data = n; temp->left = (*q); temp->right = NULL; (*q)->right = temp; (*q) = temp;

```
}
return (p);
}
void printfor( struct dnode *p )
{
printf("The data values in the list in the forward orderare:\n");
 while (p!=NULL)
{
printf("%d\t",p-> data);
 p =p->right;
}
}
void printrev( struct dnode *p )
{
printf("The data values in the list in the reverse order are:\n");
while (p!= NULL)
{
printf("%d\t",p->data);
p = p->left;
}
}
void main()
{
int n;
int x;
struct dnode *start = NULL ;
 struct dnode *end = NULL;
printf("Enter the nodes to be created \n");
 scanf("%d",&n);
while ( n-- > 0 )
{
printf( "Enter the data values to be placed in a node\n");
scanf("%d",&x);
start = insert ( start, &end,x );
}
printf("The created list is\n");
 printfor ( start );
 printrev(end);
}
```

Problem Statement	Write a program to inserts the data in a doubly linked list.
Concept	**Inserting At Beginning of the list** • Create a newNode with given value and newNode → previous as NULL. • Check whether list is Empty (head == NULL) • If it is Empty then, assign NULL to newNode → next and newNode to head. • If it is not Empty then, assign head to newNode → next and newNode to head. **Inserting At End of the list** • Create a newNode with given value and newNode → next as NULL. • Check whether list is Empty (head == NULL) • If it is Empty, then assign NULL to newNode → previous and newNode to head. • If it is not Empty, then, define a node pointer temp and initialize with head. • Keep moving the temp to its next node until it reaches to the last node in the list (until temp → next is equal to NULL). • Assign newNode to temp → next and temp to newNode → previous. **Inserting At Specific location in the list (After a Node)** • Create a newNode with given value. • Check whether list is Empty (head == NULL) • If it is Empty then, assign NULL to both newNode → previous & newNode → next and set newNode to head. • If it is not Empty then, define two node pointers temp1 & temp2 and initialize temp1 with head. • Keep moving the temp1 to its next node until it reaches to the node after which we want to insert the newNode (until temp1 → data is equal to location, here location is the node value after which we want to insert the newNode). • Every time check whether temp1 is reached to the last node. If it is reached to the last node then display 'Given node is not found in the list!!! Insertion not possible!!!' and terminate the function. Otherwise move the temp1 to next node. • Assign temp1 → next to temp2, newNode to temp1 → next, temp1 to newNode → previous, temp2 to newNode → next and newNode to temp2 → previous.
Code	# include <stdio.h> # include <stdlib.h> structdnode { int data;

```
struct node *left, *right;
};
struct dnode *insert(struct dnode *p, struct dnode **q, int n)
{
struct dnode *temp;
if(p==NULL)
{
p=(struct dnode *)malloc(sizeof(struct dnode));
if(p==NULL)
{
printf("Error\n");
exit(0);
}
p-> data = n;
p-> left = p->right =NULL;
*q =p
}
else
{
temp = (struct dnode *)malloc(sizeof(struct dnode));
if(temp == NULL)
{
printf("Error\n");
exit(0);
}
temp-> data = n;
temp->left = (*q);
temp->right = NULL;
(*q) =temp;
}
return (p);
}
void printfor( struct dnode *p )
{
printf("The data values in the list in the forward order are:\n");
while (p!= NULL)
{
printf("%d\t",p-> data); p = p-> right;
}
}
int nodecount (struct dnode *p )
{
int count=0;
while (p !=NULL)
```

```
{
count ++;
p = p->right;
}
return(count);
}

struct node * newinsert ( struct dnode *p, int node_no, int value )
{
struct dnode *temp, * temp1;
 int i;
if ( node_no <= 0 || node_no > nodecount (p))
{
printf("Error! the specified node does not exist\n");
 exit(0);
}
if ( node_no == 0)
{
temp = ( struct dnode * )malloc ( sizeof ( struct dnode ));
 if ( temp == NULL )
{
printf( " Cannot allocate \n");
exit (0);
}
temp -> data = value;
temp -> right = p;
 temp->left = NULL
p = temp ;
}
else
{
temp = p ;
i = 1;
while ( i < node_no )
{
i = i+1;
temp = temp-> right ;
}
temp1 = ( struct dnode * )malloc ( sizeof(struct dnode));
if ( temp == NULL )
{
printf("Cannot allocate \n");
exit(0);
}
```

```
temp1 -> data = value ;
temp1 -> right = temp -> right;
 temp1 -> left = temp;
temp1->right->left = temp1;
 temp1->left->right = temp1
}
return (p);
}
void main()
{
int n;
int x;
struct dnode *start = NULL ;
 struct dnode *end = NULL;
printf("Enter the nodes to be created \n");
 scanf("%d",&n);
while ( n- > 0 )
{
printf( "Enter the data values to be placed in a node\n");
 scanf("%d",&x);
start = insert ( start, &end,x );
}
printf("The created list is\n");
 printfor ( start );
printf("enter the node number after which the new node is to be inserted\n");
scanf("%d",&n);
printf("enter the data value to be placed in the newnode\n");
 scanf("%d",&x);
start=newinsert(start,n,x);
 printfor(start);
}
```

Problem Statement	Write a program to delete a specific node from a doubly linked list.
Concept	• Check whether list is Empty (head == NULL) • If it is Empty then, display 'List is Empty!!! Deletion is not possible' and terminate the function. • If it is not Empty, then define a Node pointer 'temp' and initialize with head. • Keep moving the temp until it reaches to the exact node to be deleted or to the last node. • If it is reached to the last node, then display 'Given node not found in the list! Deletion not possible!!!' and terminate the fuction. • If it is reached to the exact node which we want to delete, then check whether list is having only one node or not • If list has only one node and that is the node which is to be deleted then set head to NULL and delete temp (free(temp)). • If list contains multiple nodes, then check whether temp is the first node in the list (temp == head). • If temp is the first node, then move the head to the next node (head = head → next), set head of previous to NULL (head → previous = NULL) and delete temp. • If temp is not the first node, then check whether it is the last node in the list (temp → next == NULL). • If temp is the last node then set temp of previous of next to NULL (temp → previous → next = NULL) and delete temp (free(temp)). • If temp is not the first node and not the last node, then set temp of previous of next to temp of next (temp → previous → next = temp → next), temp of next of previous to temp of previous (temp → next → previous = temp → previous) and delete temp (free(temp)).

Code	# include <stdio.h>
	# include <stdlib.h>
	structdnode
	{
	int data;
	struct dnode *left, *right;
	};
	struct dnode *insert(struct dnode *p, struct dnode **q, int n)
	{
	struct dnode *temp;
	if(p==NULL)
	{
	p=(struct dnode *)malloc(sizeof(struct dnode));
	if(p==NULL)
	{

```
printf("Error\n");
exit(0);
}
p-> data = n;
p-> left = p->right =NULL;
*q =p;
}
else
{
temp = (struct dnode *)malloc(sizeof(struct dnode));
if(temp == NULL)
{
printf("Error\n");
exit(0);
}
temp-> data = n;
temp->left = (*q);
temp->right = NULL; (*q)->right = temp;
 (*q) = temp;
}
return (p);
}
void printfor( struct dnode *p )
{
printf("The data values in the list in the forward order are:\n"); while (p!= NULL)
{
printf("%d\t",p-> data); p = p-> right;
}
}
int nodecount (struct dnode *p )
{
int count=0;
 while (p !=NULL)
{
count ++;
p =p->right;
}
return(count);
}
struct dnode * delete( struct dnode *p, int node_no, int *val)
{
struct dnode *temp ,*prev=NULL;
 int i;
if ( node_no <= 0 || node_no > nodecount (p))
```

```
printf("The created list is\n");
printfor ( start );
printf("enter the number of the node which is to be deleted\n");
 scanf("%d",&n);
start=delete(start,n,&x);
printf("The data value of the node deleted from list is : %d\n",x);
printf("The list after deletion of the specified node is :\n");
printfor(start);
}
```

Problem Statement	Write a program for building and printing the elements of the circular linked list.
Concept	• Check whether list is Empty (head == NULL) • If it is Empty, then display 'List is Empty!!!' and terminate the function. • If it is Not Empty then, define a Node pointer 'temp' and initialize with head. • Keep displaying temp → data with an arrow (--->) until temp reaches to the last node • Finally display temp → data with arrow pointing to head → data.

Code	#include <stdio.h>
	#include <stdlib.h>
	struct node {
	int num;
	struct node * nextptr;
	}*stnode;
	void ClListcreation(int n);
	void displayClList();
	int main()
	{
	int n;
	stnode = NULL;
	printf("\n\n Circular Linked List : Create and display a circular linked list :\n");
	printf("--\n");
	
	printf(" Input the number of nodes : ");
	scanf("%d", &n);
	
	ClListcreation(n);
	displayClList();
	return 0;
	}
	
	void ClListcreation(int n)
	{
	int i, num;
	struct node *preptr, *newnode;
	
	if(n >= 1)
	{
	stnode = (struct node *)malloc(sizeof(struct node));
	
	printf(" Input data for node 1 : ");

```
        scanf("%d", &num);
        stnode->num = num;
        stnode->nextptr = NULL;
        preptr = stnode;
        for(i=2; i<=n; i++)
        {
           newnode = (struct node *)malloc(sizeof(struct node));
           printf(" Input data for node %d : ", i);
           scanf("%d", &num);
           newnode->num = num;
           newnode->nextptr = NULL;
           preptr->nextptr = newnode;
           preptr = newnode;
           preptr->nextptr = stnode;                    }
}

void displayClList()
{
   struct node *tmp;
   int n = 1;

   if(stnode == NULL)
   {
      printf(" No data found in the List yet.");
   }
   else
   {
      tmp = stnode;
      printf("\n\n Data entered in the list are :\n");

      do {
         printf(" Data %d = %d\n", n, tmp->num);

         tmp = tmp->nextptr;
         n++;
      }while(tmp != stnode);
   }
}
```

Problem Statement	Write a program to implement Binary Search Tree.
Concept	• Create a newNode with given value and set its left and right to NULL. • Check whether tree is Empty. • If the tree is Empty, then set root to newNode. • If the tree is Not Empty, then check whether the value of newNode is smaller or larger than the node (here it is root node). • If newNode is smaller than or equal to the node then move to its left child. If newNode is larger than the node then move to its right child. • Repeat the above steps until we reach to the leaf node (i.e., reaches to NULL). • After reaching the leaf node, insert the newNode as left child if the newNode is smaller or equal to that leaf node or else insert it as right child.

Code	#include <stdio.h>
	#include <stdlib.h>
	structtnode
	{
	intdata;
	struct tnode *lchild, *rchild;
	};
	
	struct tnode *insert(struct tnode *p,int val)
	{
	struct tnode *temp1,*temp2;
	if(p == NULL)
	{
	p = (struct tnode *) malloc(sizeof(struct tnode))
	
	if(p == NULL)
	{
	printf("Cannot allocate\n");
	exit(0);
	}
	p->data = val;
	p->lchild=p->rchild=NULL;
	}
	else
	{
	temp1 = p;
	while(temp1 != NULL)
	{
	temp2 = temp1;

```
if( temp1 ->data > val) temp1 = temp1->lchild;
else
}
temp1 = temp1->rchild;
if( temp2->data > val)
{
temp2->lchild = (struct tnode*)malloc(sizeof(struct tnode));
temp2 = temp2->lchild; if(temp2 == NULL)
{
printf("Cannot allocate\n");
exit(0);
}
temp2->data = val;
temp2->lchild=temp2->rchild = NULL;
}
else
{
temp2->rchild = (struct tnode*)malloc(sizeof(struct tnode));
temp2 = temp2->rchild; if(temp2 == NULL)
{
printf("Cannot allocate\n");
 exit(0);
}
temp2->data = val;
temp2->lchild=temp2->rchild = NULL;
}
}
return(p);
}
void inorder(struct tnode *p)
{
if(p != NULL)
{
inorder(p->lchild);
printf("%d\t",p->data);
inorder(p->rchild);
}
}
voidmain()
{
struct tnode *root = NULL;
 intn,x;
printf("Enter the number ofnodes\n");
scanf("%d",&n);
```

```
while( n - >0)
{
printf("Enter the data value\n");
scanf("%d",&x);
root = insert(root,x);
}
inorder(root);
}
```

Problem Statement	Write a program to count the number of nodes in Binary Search Tree.
Concept	• To count the number of nodes in a given binary tree, the tree is required to be traversed recursively until a leaf node is encountered. • When a leaf node is encountered, a count of 1 is returned to its previous activation (which is an activation for its parent), which takes the count returned from both the children's activation, adds 1 to it, and returns this value to the activation of its parent. This way, when the activation for the root of the tree returns, it returns the count of the total number of the nodes in the tree.

Code	#include <stdio.h>
	#include <stdlib.h>
	structtnode
	{
	intdata;
	struct tnode *lchild, *rchild;
	};
	int count(struct tnode *p)
	{
	if(p == NULL)
	return(0);
	else
	if(p->lchild == NULL && p->rchild == NULL) return(1);
	else
	return(1 + (count(p->lchild) + count(p->rchild)));
	
	}
	
	struct tnode *insert(struct tnode *p,int val)
	{
	struct tnode *temp1,*temp2; if(p == NULL)
	{
	p = (struct tnode *) malloc(sizeof(struct tnode));
	
	if(p == NULL)
	{
	printf("Cannot allocate\n");
	exit(0);
	}
	p->data = val;
	p->lchild=p->rchild=NULL;
	}
	else

```
{
temp1 = p;
while(temp1 != NULL)
{
temp2 = temp1;
if( temp1 ->data > val) temp1 = temp1->lchild;
else
}
 temp1 = temp1->rchild;

if( temp2->data > val)
{
temp2->lchild = (struct tnode*)malloc(sizeof(struct tnode));
temp2 = temp2->lchild; if(temp2 == NULL)
{
printf("Cannot allocate\n");
exit(0);
}
temp2->data = val;
temp2->lchild=temp2->rchild = NULL;
}
else
{
temp2->rchild = (struct tnode*)malloc(sizeof(struct tnode));
temp2 = temp2->rchild; if(temp2 == NULL)
{
printf("Cannot allocate\n");
 exit(0);
}
temp2->data = val;
temp2->lchild=temp2->rchild = NULL;
}
}
return(p);
}
 void inorder(struct tnode *p)
{
if(p != NULL)
{
inorder(p->lchild);
printf("%d\t",p->data);
inorder(p->rchild);
}
}
```

```
voidmain()
{
struct tnode *root = NULL;
 intn,x;
printf("Enter the number of nodes\n");
scanf("%d",&n);
while( n --- > 0)
{
printf("Enter the data value\n");
scanf("%d",&x);
root = insert(root,x);
}
inorder(root);
printf("\nThe number of nodes in tree are :%d\n",count(root));
}
```

Problem Statement	Write a program to search a target key in Binary Search Tree.
Concept	• Read the search element from the user. • Compare the search element with the value of root node in the tree. • If both are matched, then display "Given node is found!!!" and terminate the function • If both are not matched, then check whether search element is smaller or larger than that node value. • If search element is smaller, then continue the search process in left subtree. • If search element is larger, then continue the search process in right subtree. • Repeat the same until we find the exact element or until the search element is compared with the leaf node • If we reach to the node having the value equal to the search value then display "Element is found" and terminate the function. • If we reach to the leaf node and if it is also not matched with the search element, then display "Element is not found" and terminate the function.

Code	#include <stdio.h>
	#include <stdlib.h>
	struct tnode
	{
	int data;
	struct tnode *lchild, *rchild;
	};
	struct tnode *search(struct tnode *p,int key)
	{
	struct tnode *temp;
	temp = p;
	while(temp != NULL)
	{
	if(temp->data == key)
	return(temp);
	else
	if(temp->data > key) temp = temp->lchild;
	else
	temp = temp->rchild;
	}
	return(NULL);
	}
	void inorder1(struct tnode *p)
	{
	struct tnode *stack[100];

```
 int top;
top = −1; if(p != NULL)
{
top++;
stack[top] = p;
 p = p->lchild;
while(top >= 0)
{
while ( p!= NULL)
{
top++;
 stack[top] =p;
 p = p->lchild;
}
p = stack[top];
 top-;
printf("%d\t",p->data);
 p = p->rchild;
if ( p != NULL)
{
 top++;
p = p->lchild;
}
stack[top] = p;
}
}
}
struct tnode *insert(struct tnode *p,int val)
{
struct tnode *temp1,*temp2;
 if(p == NULL)
{
p = (struct tnode *) malloc(sizeof(struct tnode)); if(p ==NULL)
{
printf("Cannot allocate\n");
 exit(0);
}
p->data = val;
p->lchild=p->rchild=NULL;
}
else
{
temp1 = p;
while(temp1 != NULL)
```

```
{
temp2 = temp1;
if( temp1 ->data > val) temp1 = temp1->lchild;
else
}
temp1 = temp1->rchild;
if( temp2->data > val)
{
temp2->lchild = (struct tnode*)malloc(sizeof(struct tnode));
temp2 = temp2->lchild; if(temp2 == NULL)
{
printf("Cannot allocate\n");
exit(0);
}
temp2->data = val;
temp2->lchild=temp2->rchild = NULL;
}
else
{
temp2->rchild = (struct tnode*)malloc(sizeof(struct tnode))
temp2 = temp2->rchild; if(temp2 == NULL)
{
printf("Cannot allocate\n");
 exit(0);
}
temp2->data = val;
temp2->lchild=temp2->rchild = NULL;
}
}
return(p);
}
voidmain()
{
struct tnode *root = NULL, *temp =NULL;
intn,x;
printf("Enter the number of nodes in thetree\n");
 scanf("%d",&n);
while( n - >0)
{
printf("Enter the data value\n");
scanf("%d",&x);
root = insert(root,x);
}
printf("The created tree is:\n");
```

```
inorder1(root);
printf("\n Enter the value of the node to besearched\n");
 scanf("%d",&n);
temp=search(root,n);
if(temp != NULL)
printf("The data value is present in the tree \n");
 else
printf("The data value is not present in the tree \n");
}
```

Problem Statement	Write a program to delete a node in Binary Search Tree.
Concept	**Deleting a leaf node** Step 1 - Find the node to be deleted using search operation Step 2 - Delete the node using free function (If it is a leaf) and terminate the function. **Deleting a node with one child** Step 1 - Find the node to be deleted using search operation Step 2 - If it has only one child then create a link between its parent node and child node. Step 3 - Delete the node using free function and terminate the function. **Deleting a node with two children** Step 1 - Find the node to be deleted using search operation Step 2 - If it has two children, then find the largest node in its left subtree (OR) the smallest node in its right subtree. Step 3 - Swap both deleting node and node which is found in the above step. Step 4 - Then check whether deleting node came to case 1 or case 2 or else goto step 2 Step 5 - If it comes to case 1, then delete using case 1 logic. Step 6- If it comes to case 2, then delete using case 2 logic. Step 7 - Repeat the same process until the node is deleted from the tree.

Code	#include <stdio.h>
	#include <stdlib.h>
	struct tnode
	{
	int data;
	struct tnode *lchild, *rchild;
	};
	struct tnode *getptr(struct tnode *p, int key, struct tnode **y)
	{
	struct tnode *temp;
	if(p == NULL)
	return(NULL);
	temp = p;
	*y = NULL;
	while(temp !=NULL)
	{
	if(temp->data == key) return(temp);
	else
	{

```
*y = temp;
if(temp->data > key) temp = temp->lchild;
else
temp = temp->rchild;
}
}
return(NULL);
}
 struct tnode *delete(struct tnode *p,int val)
{
struct tnode *x, *y, *temp;
x = getptr(p,val,&y);
if( x == NULL)
{
printf("The node does not exists\n");
return(p);
}
else
{
if( x == p)
{
temp = x->lchild;
 y = x->rchild;
p = temp;
while(temp->rchild != NULL) temp = temp->rchild;
temp->rchild=y; free(x);
return(p);
}
if( x->lchild != NULL && x->rchild != NULL)
{
if(y->lchild == x)
{
temp = x->lchild;
y->lchild = x->lchild;
 while(temp->rchild != NULL)
temp = temp->rchild;
temp->rchild=x->rchild;
 x->lchild=NULL;
x->rchild=NULL;
}
else
{
temp = x->rchild;
y->rchild = x->rchild;
```

```
while(temp->lchild != NULL)
temp = temp->lchild;
 temp->lchild=x->lchild;
x->lchild=NULL;
x->rchild=NULL;
}

free(x);
return(p);
}
if(x->lchild == NULL && x->rchild !== NULL)
{
if(y->lchild == x) y->lchild = x->rchild;
else
y->rchild = x->rchild;
 x->rchild; = NULL;
free(x);
return(p);
}
if( x->lchild != NULL && x->rchild == NULL)
{
if(y->lchild == x)
y->lchild = x->lchild ;
else
y->rchild = x->lchild;
 x->lchild = NULL;
 free(x);
return(p);
}
if(x->lchild == NULL && x->rchild == NULL)
{
if(y->lchild == x)
y->lchild = NULL ;
else
y->rchild = NULL;
free(x);
return(p);
}
}
}
 void inorder1(struct tnode *p)
{
struct tnode *stack[100]; int top;
top = −1;
```

```
if(p != NULL)
{
top++; stack[top] = p;
 p = p->lchild;
while(top >= 0)
{
while ( p!= NULL)
{
top++;
stack[top] =p;
p = p->lchild;
}
p = stack[top];
top-;
printf("%d\t",p->data);
p = p->rchild;
if ( p != NULL)
{
top++; stack[top] = p;
p = p->lchild;
}
}
}
}
struct tnode *insert(struct tnode *p,int val)
{
struct tnode *temp1,*temp2; if(p == NULL)
{
p = (struct tnode *) malloc(sizeof(struct tnode));
{
printf("Cannot allocate\n");
exit(0);
}
p->data = val;
p->lchild=p->rchild=NULL;
}
else
{
temp1 = p;
 while(temp1 != NULL)
{
temp2 = temp1;
if( temp1 ->data > val) temp1 = temp1->lchild;
else
```

```
}
temp1 = temp1->rchild;
if( temp2->data > val)
{
temp2->lchild = (struct tnode*)malloc(sizeof(struct tnode))
temp2 = temp2->lchild;
 if(temp2 == NULL)
{
printf("Cannot allocate\n");
exit(0);
}
temp2->data = val;
temp2->lchild=temp2->rchild = NULL;
}
else
{
temp2->rchild = (struct tnode*)malloc(sizeof(struct tnode));
temp2 = temp2->rchild; if(temp2 == NULL)
{
printf("Cannot allocate\n");
exit(0);
}
temp2->data = val;
temp2->lchild=temp2->rchild = NULL;
}
}
return(p);
}

void main()
{
struct tnode *root = NULL; int n,x;
printf("Enter the number of nodes in the tree\n");
scanf("%d",&n); while( n - > 0)
{
printf("Enter the data value\n");
scanf("%d",&x);
root = insert(root,x);
}
printf("The created tree is:\n");
 inorder1(root);
printf("\n Enter the value of the node to bedeleted\n");
 scanf("%d",&n);
root=delete(root,n);
```

```
printf("The tree after deletion is \n");
inorder1(root);
}
```