Government Girls' Polytechnic, Bilaspur

Name of the Lab: **Programming Lab**

Practical: **Data Structure Lab**

Class: **4th Semester (Computer Science & Engineering)**

Teachers Assessment: 30   End Semester Examination:70

EXPERIMENT NO:-1

OBJECTIVE :- Program to search an element of array using linear search.

HARDWARE & SYSTEM SOFTWARE REQUIRED :-Not Required

SOFTWARE REQUIRED :- TURBO C++

THEORY :- Searching is the process of finding the desired record in a search table . The search may be internal or external. The linear search technique is the simplest of all the techniques that searches through the entire table.

FLOW CHART (IF REQUIRED) :- No flow chart

PROGRAM INPUTS & OUTPUT :-

```
 #include<stdio.h>
#include<conio.h>
void main()
{
int a[100]n,item,loc=-1;

clrscr();
printf("/n how many elements 0");
scanf("%d",& n");
printf("enter the number/n");
for(i=0,il=n-1,i++);
{
scanf("%d",& a[i];
}pritf("enter the number to be search/n");
scanf("%d",& item);
for(i=0; i<=n-1; i++)
{
if(item==a[i]
{
loc=i
break;
}
if(loc>=0);
printf("/n%d is found in position %d,item,loc+1);
else
printf("/n item does not exist");
}
getch();
}
```

INPUT:

```
how many elements :5
enter the number : 10 50 60 30 40
enter the number to be search: 60
```

OUTPUT:

```
60 is found in position 3
```

OBSERVATIONS :- program is running successfully.

EXPERIMENT NO:-2

OBJECTIVE :- Program to reverse the element of array.

HARDWARE & SYSTEM SOFTWARE REQUIRED :- Not required.

SOFTWARE REQUIRED :- TURBO C++

THEORY :- An array can be defined as an infinite collection of homogeneous elements. Arrays are always stored in consecutive memory locations. Array name is actually a pointer to the first location of the memory block allocated to the name of the array. An array can be either be a integer ,character ,or floating data type can be initialized only during declaration time, and not afterwards.

FLOW CHART (IF REQUIRED) :- Not Required

PROGRAM INPUTS & OUTPUT :-

```
 #include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
int len [i];
char pointer [80];
printf("enter any string");
gets (pointer);
len=strrev(pointer)
for= (i=len; 1>0;i--);
printf ("%c",pointer[i];
getch();
```

INPUT: friend

OUTPUT:dneirf

OBSERVATION :- String has been reversed.

EXPERIMENT NO:-3

OBJECTIVE :- Insertion and deletion on array at specified position.

HARDWARE & SYSTEM SOFTWARE REQUIRED : Not Required

SOFTWARE REQUIRED :- TURBO C++

THEORY :- An array can be defined as an infinite collection of homogeneous elements. Arrays are always stored in consecutive memory locations .Array name is actually a pointer to the first location of the memory block allocated to the name of the array.

FLOW CHART (IF REQUIRED) :- Not Required

PROGRAM INPUTS & OUTPUT :-

```
#include <stdio.h>
#include <conio.h>
int i, len, pos, number;
main()
{
int a [100];
void insert (int a[], int, int, int);
clrscr();
printf ("enter array to be read");
scanf("%d &len);
printf("enter int);
for (i=0; i<len-1; i++)
{
scanf("%d"&a[i]);
}
printf("enter position in the array for insertion");
scanf("%d", & pos);
_ _ pos;
insert (a, len, pos, num);
}
void insert int a[]; int len int int pos, int num);
{
a[i+] = a[i]; /*shift down by 1 postion*/
}
a[pos] =num;
if(pos>len)
{
printf("insertion outside the array");
}
len++;
printf("new array");
for(i=0; i<len; i++)
{
```

```
printf("%d, \n", a(i));
}
getch();
}
```

Input: Enter array to be read : 20 30 40 50
Enter position in the array for insertion: 3, 15
Output: new array: 20 30 15 40 50

OBSERVATIONS :- Program has been successfully accessed.

EXPERIMENT NO:-4

OBJECTIVE :- Program based on structure union.

HARDWARE & SYSTEM SOFTWARE REQUIRED :- Not Required

SOFTWARE REQUIRED :- TURBO C++

THEORY :-A structure is usually used to store dissimilar data together. Structure elements can be accessed through a structure variable using a dot(.) operator. It is possible to create an array of structures.

FLOW CHART (IF REQUIRED) :- Not Required

PROGRAM:-

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
clrscr();
int i,j;
struct student
{
int roll no;
char sex;
float height, weight;
};
struct student data[3]={
{121 ; m :5.5: 52};
{122 ; f; 5.3: 54 }
{123;m; 5.2: 42}
};
printf("the intialized content are\n");
for(i=0;i<=n;i++)
{
printf("\n**record no %d**\n",i+1);
printf("%d\n",data[j].roll no);
printf("%e\n",data[i].sex);
printf("%s\n",data[i]height);
printf("%s\n",data[j]weight);
}
getch();
}
```

OUTPUT:

```
121, m ,5.5, 52
122,f,5.3,54
 123,m, 5.2,42
```

OBSERVATIONS: Program has been run successfully.

EXPERIMENT NO:-5

OBJECTIVE :- Program to implement PUSH and POP operation on stack.

HARDWARE & SYSTEM SOFTWARE REQUIRED :- Not Required

SOFTWARE REQUIRED :- Turbo c++

THEORY :-PUSH- The process of adding a new element to the top stack is called PUSH operation. Pushing an element in the stack in the stack invoke adding of element , as the new element will be insertion at the top after every push operation the top is incremented by one. In case the array is full and no new element can be accommodated, it is called STACH-FULL condition**.**
POP- The process of deletion an element from the top of stack is called POP operation .After every pop operation the stack is decrement by one . IT there is no element on the stack and the pop is performed then this will result into STACK UNDERFLOW CONDITION condition.

FLOW CHART (IF REQUIRED) :- Not Required

PROGRAM :-
```
#include<stdio.h>
#include<conio.h>
#include<---->
#define maxsize=10
void push();
intpop();
void---
int stack[maxsize];
inttop=-1;
void main()
{
int choice;
char ch;
do
{
clrscr();
printf("\n1.push");
printf("\n2.pop");
printf("\n3.traverle");
printf("\n.enter yourchoice");
scanf("%d",&choice);
switch(choice);
{
case1:push();
break();
case p
rintf("\n the delete element");
break();case3:travers();
break();
```

```
default:printf("\n you entered wrong choice");
]
printf("\n do you wish to continue(y\n");
flush(stdio);
scanf("%e",&ch);
}
void push()
{
int item;
if(top==maxsize-1)
{
printf("\n the stack is full");
getch();
exit(0);
}
else
{
printf("enter the element to be inserted");
scanf("%d",&item);
top=top+1;
stack[top]=item;
}
int top();
{
int item;
if(top==-1);
{
int item;
if(top==-1);
{
printf("the stack is empty");
getch();
exit(0);
}
else
{
item=stack[top];
top=top-1;
}
return=(item);
}
void trave---()
{
int i;
if(top==-1)
{
printf("the stack is empty");
getch();
exit(0);
}
else
}
for(i=top;i>=0;i--)
{
printf("--the element");
printf("\n %d",stack(i));
}
}
}
```

INPUT: Enter the elements of stack : 20 30 40 50
Enter your choice: 1
Enter the element to be inserted: 80
OUTPUT: New stack : 20 30 40 50 80

OBSERVATIONS :- Program has been run successfully.

EXPERIMENT NO:- 6

OBJECTIVE :- One program based on - In fix to post fix or infix to prefix using stack Concept - Recursion using stack.
HARDWARE & SYSTEM SOFTWARE REQUIRED :- Not Required
SOFTWARE REQUIRED :- TURBO C++
THEORY :- INFIX – The infix is what we come across in our general mathematics ,where the operator is written in between the operands.
Prefix-The prefix notation in which the operator is written before the operand, it is also called polish notation.
Postfix –in the notation the operator are written after the operands ,so it is called the postfix notation.
FLOW CHART (IF REQUIRED) :- Not Required
PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char stack[50];
int top=-1;
void in_to_post(char infix[ ]);
void push(char);
char pop();
void main ()
{
char infix [25];
printf("enter the infix expression");
gets(infix);
in_to_post(infix);
getch();
}
void push (char symb)
{
if(top>=49)
{
printf("stack overflow");
getch();
return;
}
else
{
top=top+1;
stack [top]=symb;
}
}
char pop ()
{
char item;
if(top==-1)
{
printf("stack empty");
```

```
getch();
return(0);
}
else
{
item=stack[top];
top--;
}
}
int preced(char ch)
{
if (ch==47)
{
return (5);
}
else
if(ch=42)
{
return(4);
}
else if(ch=43)
{
return(3);
}
else
return(2);
}
void in_to_post(char infix [])
{
int length;
static int index =0,pos=0;
char symbol,temp;
char postfix[40];
length=stren(infix);
push('#');
while (index< length)
{
symbol=infix[index];
switch(symbol)
{
case'(' :push(symbol));
break;
caes')': temp=pop();
while (temp!=<("))
{
postfix[pos]=temp;
pos++;
temp=pop();
}
break:
case'+':
```

```
case’-‘:
case’*’:
case’/’:
case’^,:
while(preced(stack[top])>=preced(symbol))
{
temp=pop();
postfix[pos]=temp;
pos++;
}
push(symbol);
break:
default:postfix[pos++]=symbol;
break;
}
index++;
}
while(top>0)
{
temp=pop();
postfix[pos++]=temp;
}
postfix[pos++]=’\0’;
puts(postfix):
return;
}
```

OUTPUT: enter the infix a+b
ab+

OBSERVATIONS : Program has been run successfully.

EXPERIMENT NO:-7

OBJECTIVE :- Program based on queue & their operations for an application.
HARDWARE & SYSTEM SOFTWARE REQUIRED :- Not Required
SOFTWARE REQUIRED :- Turbo C++
THEORY :- A queue is anon primitive linear data structure . It is an homogeneous collection of element in which new element are added at one end called the Rear end , and the existing element are deleted form other end called the front end.
FLOW CHART (IF REQUIRED) :-
PROGRAM INPUTS :-

```
#include<stdio.h>
#include<conio.h>
int queue[5];
long front,rear;
void main()
{
int choice, infor;
clrscr();
initqueue();  //initilizing queue
while(1)
{
clrscr();
printf("menu \n");  //displaying menu
printf("1. insertion element in queue\n");
printf("2. delete an element from queue\n");
printf("3.display the queue\n");
printf("4.exit!\n");
printf("your choice");
scanf("%i",& choice);
switch(choice);
{
case1:if(rear<4)
{
printf("enter the number");
scanf("%d",&infor);
if(front==-1)
{
front=0;
rear=0;
}
else
rear=rear=1;
queue[rear]=infor;
}
else
printf("queue is full");
getch();
break;
case2: int infor;
if(front!=rear)
{
infor=queue[front];
if(front==rear)
{
front=-1;
rear=-1;
}
else
front=front+1;
printf("no deleted is =%d",infor);
}
else
printf("queue is empty");
getch();
break;
case3:display();
getch();
break();
case4:exit();
break();
default:printf("you entered wrong choice!");
getch();
break;
}
}
}
void initqueue()
{
front=rear=-1; //initilizing front & queue
}
void display() //display the current position of the queue
{
int i; //for loop driver
for(i=front;i<=rear;i++)
printf("%d \n",queue[i]); //displaying element in queue.
}
```

```
OUTPUT: Enter your choice: 1
        Enter the elements of queue: 20 30 40 50
        Enter the element to be inserted: 45
        New Queue is: 20 30 40 50 45
```

OBSERVATIONS :- Program has been run successfully.

EXPERIMENT NO:-8

OBJECTIVE :-A Program for implementation of circular queue.
HARDWARE & SYSTEM SOFTWARE REQUIRED :- Not Required
SOFTWARE REQUIRED :- TURBO C++
THEORY :-
A circular queue is one in which the insertion of a new element is done at the very first location of the queue if the last location of the queue is full. If we have a queue Q of n elements, then after inserting an element in last location of the array , the next element will be inserted at the very first location.
A circular queue overcomes the problem of unutilized space in linear queues implemented as arrays.s

FLOW CHART (IF REQUIRED) :- Not required.
PROGRAM INPUTS & OUTPUT :
```
#include<stdio.h>
#include<conio.h>
#define MAXSIZE=5
int cq [10];
int front =-1,rear=0;
int choice i;
char ch;
void main()
{
do
{
clrscr();
printf("1.insert  \n");
printf("2.delete \n");
printf("3.display \n");
printf("4.exit \n");
printf("enter your choice \n");
scanf("%d",&choice);
switch(choice);
{
case1:cq insert();
break;
case2:cq delete();
break;
return;
int num;
if(front == -10)
{
printf("queue is empty \n");
return;
}
else
{
num=cq[front];
printf("delete element is=%d \n",cq[front]);
```

```
if(front == rear)
(front == rear+1;
else
front=(front=(front+1)%maxsize;
}
return(num);
}
cq display();
{
int i;
if(front==-1)
{
printf("queue is empty \n");
return;
}
case3:cqdisplay();
break;
case4:return:
}
flush(stdin);
}
while(choice!=4);
]
cq insert()
{
int num;
if(front==(rear+1)%maxsize)
{
printf("queue is full\n");
return;
}
else
{
printf("enter the element to be inserted\n");
scanf("%d",&num);
if(front==-1)
front=rear=0
else
rear=(rear+1)%maxsize;
cq[rear]=num;
}
else
{
printf("\n the status of the queue \n");
for(i=front;i<=rear;i++)
{
printf("%d",cq[i]);
}
}
if(front>rear)
{
```

```
for(i=front;i<maxsize;i++)
{
printf("%d",cq[i]);
}
for(i=0;i<=rear;i++)
{
printf("%d",cq[i]);
}
}
printf("\n");
}
```

OUTPUT :
```
Enter the element of queue:
Enter your choice: 2
Queue is empty
```

OBSERVATIONS :-Program has been run successfully.

EXPERIMENT NO:-9

OBJECTIVE :- Program based on list operations and its application.

HARDWARE & SYSTEM SOFTWARE REQUIRED :- Not Required

SOFTWARE REQUIRED :- TURBO C++

THEORY :- Linked lists are special list of some data elements linked to one another. The logical ordering is represented by having each elements pointing to the next element .Each element is called a node which has two parts. INFO part which stores the information and the pointer which points to the next element.linked list are of four types;

1. Singly linked list
2. Doubly linked list
3. Circular linked list
4. Doubly circular linked list

Applications: Linked list are widely used to represent and manipulate polynomials. Polynomials are the expressions containing numbers of terms with non zero coefficients and exponents.

FLOW CHART (IF REQUIRED) :-
PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
void main()
{
struct node
{
int num;
struct node*ptr;
};
tuprdef struct node NODE;
NODE *head.*frist,*temp;
int count=0;
char choice;
clrscr();
first=NULL;
do
{
head=(NODE*)malloc(sizeof(NODE));
printf("Enter the data item\n");
scanf("%d", & head-> num);
if(first !=NULL)
{
temp-> ptr=head;
temp=head;
}
else
{
first=temp=head;
}
fflush(stdin);
printf("do you want to continue(type yor n)?\n");
scanf("%c"&choice);
}
while((choice=='y')||(choice=='Y'));
temp-> ptr=NULL;
temp=first;
printf("status of the linked list is \n");
while(temp!=NULL)
{
printf("%d\n", temp-> num);
count++;
temp=temp->ptr;
}
printf("No of nodes in the list=%d\n",count);
getch();
}
```

OUTPUT: enter the data item 12

```
do you want to continue(type y or n)?
y
enter the data item 24
do you want to continue(type y or n)?
y

enter the data item 34
do you want to continue(type y or n)?
y

enter the data item 25
do you want to continue(type y or n)?
y

enter the data item 20
do you want to continue(type y or n)?
n
Status of the linked list is
12
24
34
25
20
No. of nodes in the list =5
```

OBSERVATIONS :- Program has been run successfully.

EXPERIMENT NO:-10

OBJECTIVE :- Program based on pointers in C.

HARDWARE & SYSTEM SOFTWARE REQUIRED :- Not Required

SOFTWARE REQUIRED :- TURBO C++

THEORY :- Pointers are the variables that store the memory addresses. A pointer enables us to access a variable that is defined outside the function. It reduces the length and complexity of a program. The use of a pointer array to character strings results in saving of data storage space in memory.Pointers can be declare as: data type *pt_name.
The * tells that the variable pt_name is a pointer variable.
pt_name needs a memory allocation.
pt_name points to a variable of type data type.
FLOW CHART (IF REQUIRED) :- Not required

PROGRAM INPUTS & OUTPUT :-

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,*p1, *p2,x ,y ,z;
a=12;
b=4;
p1=&a;
p2=&b;
x= *p1 * *p2-6;
y=4* - *p2/ *p1 + 10;
printf (“address of a= % u\n”, p1);
printf (“address of b= % u\n”, p2);
printf(“\n”);
printf(“a= %d, y= %d \n”, x, y);
printf(“x=%d, y= %d\n”, x, y);
*p2=*p2 +3;
*p1= *p2-5;
z= *p1 * * p2 -6;
printf(“\n a=%d,b= %d”,a,b);
printf(“z=%d\n”,z);
}
```

OUTPUT

```
Address of a=4020
address of b= 4016
a=12, b= 4
x=42, y=9
a=2; b=7, z=8
s
```

OBSERVATIONS :- Program has been run successfully.

EXPERIMENT NO:-11

OBJECTIVE :- Implementation of tree using linked list.

HARDWARE & SYSTEM SOFTWARE REQUIRED :- Not Required

SOFTWARE REQUIRED :- TURBO C++

THEORY :- Three representations are commonly used for graphs. These are adjacent matrix, adjacency list representation and multi-list representation. Minimum spanning tree are trees are free trees designed only on undirected graphs. Binary trees can be represented either using array representation or using a linked list representation.. The node consists of three fields such as:
data field: holds the value to be given.
Left child: is a link field which contains the address of its left node
Right child: contains the address of the right node.
FLOW CHART (IF REQUIRED) :- Not required.

PROGRAM INPUTS & OUTPUT :-
Algorithm: Creation of binary tree.

```
Create_tree(info, node)
Where info-> information for which we have to create a node.
Node-> structure type variable to pointer both left and right child.
Step 1: [check whether the tree is empty]
If node=null
Node = create a node
Left_child[node] =null
Right_child[node]=null

Step 2:[test for the left child]
If info[node]>=info
Left_child[node] =call create_tree
(info, left_child [node])
Else
Right_ child[node]= call create_tree
(info, right_child[node])
Step 3 Return(node)
```

OBSERVATIONS :- Program has been run successfully.

EXPERIMENT NO:-12

OBJECTIVE :- Implementation of different types of sorting techniques.

HARDWARE & SYSTEM SOFTWARE REQUIRED :- Not required.

SOFTWARE REQUIRED :- TURBO C++

THEORY :- Sorting an array is the technique of arranging the array elements in a specified order i.e. either in ascending order or descending order. There are seven types of sorting techniques. They are as follows:

1. Bubble sort: Each element is compared with its adjacent element. If the first element is larger than the second one then the position of the element are interchanged.
2. Selection sort: By means of a nest of for loops, a pass through the array is made to locate the minimum value. Once this found, it is placed in the first position of the array.
3. Insertion sort: It is one that sorts a set pf values by inserting values into an existing sorted file
4. Quick sort: It works by partitioning the array to be sorted. And each partition is in turn sorted recursively.
5. Bucket sort:Bucket or radix sort is a method, can be used to sort a list of names alphabetically. Here the base or radix is 26.s
6. Merge sort: It is a sorting technique which divides the array into subarrays of size 2 and merge adjacent pairs.
7. Heap sort: In this method, we will interpret the array to be sorted as a binary tree, in a sequential representation of the binary tree.

FLOW CHART (IF REQUIRED) :- Not required.

PROGRAM INPUTS & OUTPUT :-
Algorithm for insertion sort
Let a be an array of n elements which we want to sort temp be a temporary variable to interchange the two values . k be the total no. of passes and j be another control variable.

```
1. Set k=1
2. For k=1 to (n-1)
   Set temp = a[k]
   Set j=k-1
   While temp< a[j] and (j>=0) perform the following steps.
   Set a[j+1]= a[j]
   [ end of loop structure]
    Assign the value of temp to a[j+1]
   [end of for loop structure]
3. Exit.
```

OBSERVATIONS :- Algorithm worked properly.

EXPERIMENT NO:-13

OBJECTIVE :- Implementation of Binary search Algorithm using Binary tree.

HARDWARE & SYSTEM SOFTWARE REQUIRED :- Not Required

SOFTWARE REQUIRED :- TURBO C++

THEORY :- A binary tree is a finite set of data items which is either empty or consists of a single item called the root and two disjoint binary trees called the left sub tree. In a binary tree the maximum degree of any node is atmost two .Algorithm:

```
1. Set beg =lb, end= ub and mid= int ((beg+ end))/2
2. Repeat steps 3 and 4 while beg<= end and a [mid] != item
3. if item < a[mid], then
   set end = mid-1
   else
   set beg=mid+1
4. set mid= int((beg+ end))/2
5. if a[mid] = item then
   set loc=mid
   else
   set loc= null
   6 exit
```

FLOW CHART (IF REQUIRED) :- Not Required

PROGRAM INPUTS & OUTPUT :- elements of binary tree: 19 24 35 46 57 68

```
search 46
46 is found in 4th position
search is successful.
```

OBSERVATIONS :- Algorithm worked correctly.

EXPERIMENT NO:-14

OBJECTIVE :- Assignment based on graph theory.
HARDWARE & SYSTEM SOFTWARE REQUIRED :- Not Required
SOFTWARE REQUIRED :- TURBO C++

THEORY :- A graph data structure consists of a finite (and possibly mutable) set of ordered pairs, called edges or arcs, of certain entities called nodes or vertices. As in mathematics, an edge (*x*,*y*) is said to point or go from *x* to *y*. The nodes may be part of the graph structure, or may be external entities represented by integer indices. Graph traversal means visiting all the nodes of the graph. There are two methods of graph traversal:

1. BSF (breadth first traversal): one node is selected as the start position. It is visited and marked , then all unvisited node adjacent to the next node are visited and marked in some sequential order. BSF proceeds level by level.
2. DFS(Depth first traversal): DFS follows first a path from the starting node to an ending node, then another path from the start to the end and so forth until all nodes have been visited.

Analysis of DFS
Setting/getting a vertex/edge label takes O(1) time
Each vertex is labeled twice
once as UNEXPLORED
once as VISITED
Each edge is labeled twice
once as UNEXPLORED
once as DISCOVERY or BACK
Method incident Edges is called once for each vertex
DFS runs in O(n + m) time provided the graph is
represented by the adjacency list structure $\Sigma v \deg(v) = 2m$
FLOW CHART (IF REQUIRED) :- Not required.

PROGRAM INPUT & OUTPUT :- Not Required
OBSERVATIONS :- Graph theory has been studied.