

POORNIMA

COLLEGE OF ENGINEERING

ISI-6, RIICO Institutional Area, Sitapura, Jaipur-302022, Rajasthan

Phone/Fax: 0141-2770790-92, www.pce.poornima.org

Data Structure & Algorithm Lab Manual

(Lab Code: 3CS4-21)

3th Semester, 2nd Year



Department of Computer Engineering

Session: 2022-23

TABLE OF CONTENT

S. No.	Topic/Name of Experiment	Page Number
GENERALDETAILS		
1	Vision & Mission of Institute and Department	iii
2	RTU Syllabus and Marking Scheme	iv
3	Lab Outcomes and its Mapping with Pos and PSOs	v-vii
4	Rubrics of Lab	viii-ix
5	Lab Conduction Plan	x
6	Lab Rotor Plan	xi
7	General Lab Instructions	xii-xiii
8	Lab Specific Safety Rules	xiii
LIST OF EXPERIMENTS (AS PER RTU SYLLABUS)		
1	Zero Lab	1-2
2	Introduction: Objective, scope and outcome of the course.	3
3	Write a simple C program on a 32 bit compiler to understand the concept of array storage, size of a word. The program shall be written illustrating the concept of row major and column major storage. Find the address of element and verify it with the theoretical value. Program may be written for arrays upto 4-dimensions.	4
4	Simulate a stack, queue, circular queue and dequeue using a one dimensional array as storage element. The program should implement the basic addition, deletion and traversal operations.	5-18
5	Represent a 2-variable polynomial using array. Use this representation to implement addition of polynomials.	19-24
6	Represent a sparse matrix using array. Implement addition and transposition operations using the representation.	25-27
7	Implement singly, doubly and circularly connected linked lists illustrating operations like addition at different locations, deletion from specified locations and traversal.	28-43
8	Repeat exercises 2, 3 & 4 with linked structures.	44-50
9	Implementation of binary tree with operations like addition, deletion, traversal.	51-56
10	Depth first and breadth first traversal of graphs represented using adjacency matrix and list.	56-63
11	Implementation of binary search in arrays and on linked Binary Search Tree.	64-65

12	Implementation of different sorting algorithm like insertion, quick, heap, bubble and many more sorting algorithms.	66-71
13	Beyond the Syllabus Exp -1	72-74
14	Beyond the Syllabus Exp -2	75-76

INSTITUTE VISION&MISSION

VISION

To create knowledge-based society with scientific temper, team spirit and dignity of labor to face the global competitive challenges.

MISSION

To evolve and develop skill-based systems for effective delivery of knowledge so as to equip young professionals with dedication & commitment to excellence in all spheres of life.

DEPARTMENT VISION & MISSION

VISION

Evolve as a center of excellence with wider recognition and to adapt the rapid innovation in Computer Engineering.

MISSION

- To provide a learning-centered environment that will enable students and faculty members to achieve their goals empowering them to compete globally for the most desirable careers in academia and industry.
- To contribute significantly to the research and the discovery of new arenas of knowledge and methods in the rapid developing field of Computer Engineering.
- To support society through participation and transfer of advanced technology from one sector to another.

RTU SYLLABUS AND MARKIN SCHEME

3CS4-21: Data Structure & Algorithm Lab	
Credit:1	Max.Marks:100(IA:60,ETE:40)
0L+0T+3P	EndTermExam:2Hours
S. No.	NAME OF EXPERIMENTS
1	Introduction: Objective, scope and outcome of the course.
2	Write a simple C program on a 32 bit compiler to understand the concept of array storage, size of a word. The program shall be written illustrating the concept of row major and column major storage. Find the address of element and verify it with the theoretical value. Program may be written for arrays upto 4-dimensions.
3	Simulate a stack, queue, circular queue and dequeue using a one dimensional array as storage element. The program should implement the basic addition, deletion and traversal operations.
4	Represent a 2-variable polynomial using array. Use this representation to implement addition of polynomials.
5	Represent a sparse matrix using array. Implement addition and transposition operations using the representation.
6	Implement singly, doubly and circularly connected linked lists illustrating operations like addition at different locations, deletion from specified locations and traversal.
7	Repeat exercises 2, 3 & 4 with linked structures.
8	Implementation of binary tree with operations like addition, deletion, traversal.
9	Depth first and breadth first traversal of graphs represented using adjacency matrix and list.
10	Implementation of binary search in arrays and on linked Binary Search Tree.
11	Implementation of different sorting algorithm like insertion, quick, heap, bubble and many more sorting algorithms.

EVALUATIONSCHEME

I+II Mid Term Examination			Attendance and performance			End Term Examination			Total Marks
Experiment	Viva	Total	Attendance	Performance	Total	Experiment	Viva	Total	
40	20	60	20	40	60	30	10	40	100

DISTRIBUTIONOFMARKSFOREACHEXPERIMENT

Attendance	Record	Performance	Total
2	3	5	10

LAB OUTCOME AND ITS MAPPING WITH PO& PSO**LAB OUTCOMES**

After completion of this course, students will be able to–

3CS4-21.1	To Utilize searching and sorting algorithms on given values.
3CS4-21.2	To analyze the time and space efficiency of the data structure.
3CS4-21.3	To Evaluate traversing, insertion and deletion operations on Linear and non-linear data structures.
3CS4-21.4	To construct the solutions for real time applications.

LO-PO-PSOMAPPINGMATRIXOFCOURSE

LO/PO/PSO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
3CS4-21.1	2	-	-	-	2	-	-	-	-	2	-	-	2	-	-
3CS4-21.2	-	-	-	-	-	2	-	-	-	-	-	-	2	-	-
3CS4-21.3	-	-	-	-	-	-	2	-	-	-	-	2	-	2	-
3CS4-21.4	-	-	-	-	2	-	-	-	2	-	-	-	-	-	3

PROGRAM OUTCOMES (POs)

PO1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
PO2	Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	Individual and teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multi-disciplinary settings.
PO10	Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	Project management and finance: Demonstrate knowledge and understanding of the

	engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	Life-long learning: Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO1	The ability to understand and apply knowledge of mathematics, system analysis & design, Data Modelling, Cloud Technology, and latest tools to develop computer based solutions in the areas of system software, Multimedia, Web Applications, Big data analytics, IOT, Business Intelligence and Networking systems
PSO2	The ability to understand the evolutionary changes in computing, apply standards and ethical practices in project development using latest tools & Technologies to solve societal problems and meet the challenges of the future.
PSO3	The ability to employ modern computing tools and platforms to be an entrepreneur, lifelong learning and higher studies

RUBRICS FOR LAB**Laboratory Evaluation Rubrics:**

S. No.	Criteria	SubCriteriaandMarksDistribution			Outstanding(>90%)	Admirable(70-90%)	Average(40-69%)	Inadequate(<40%)
		Mid-Term	End-Team	Continues Evaluation				
A	PERFORMANCE(PO1,PO8,PO9)	Procedure Followed M.M.50=3 M.M.75=4 M.M.100=6	Procedure Followed M.M.50=3 M.M.75=4 M.M.100=6	Procedure Followed M.M.50=1 M.M.75=2 M.M.100=2	<ul style="list-style-type: none"> All possible system and Input/Output variables are taken intoaccount Performancemeasuresare properlydefined Experimental scenarios arevery welldefined 	<ul style="list-style-type: none"> Most of the system andInput/ Output variables aretaken intoaccount Most of the Performancemeasures are properlydefined Experimental scenariosaredefined correctly 	<ul style="list-style-type: none"> Some of the system and Input/Output variables are taken intoaccount Some of the Performancemeasuresareproperlydefined Experimental scenarios aredefinedbutnot sufficient 	<ul style="list-style-type: none"> System and Input/ Outputvariablesarenotdefined Performance measures are notproperly defined Experimentalscenariosnotdefined
		Individual/Team Work M.M.50=3 M.M.75=4 M.M.100=6	Individual/Team Work M.M.50=3 M.M.75=4 M.M.100=6	Individual/Team Work M.M.50=1 M.M.75=2 M.M.100=2	<ul style="list-style-type: none"> Coordinationamongthegroup membersinperforming theexperimentwas excellent 	<ul style="list-style-type: none"> Coordination among thegroup members inperforming the experimentwasgood 	<ul style="list-style-type: none"> Coordinationamongthegroup membersinperforming theexperimentwas average 	<ul style="list-style-type: none"> Coordinationamongthegroup membersinperforming theexperimentwas verypoor
		Precision indatacollection M.M.50=3 M.M.75=4 M.M.100=6	Precision indatacollection M.M.50=3 M.M.75=4 M.M.100=6	Precision indatacollection M.M.50=2 M.M.75=2 M.M.100=4	<ul style="list-style-type: none"> Data collected is correct insize and from the experimentperformed 	<ul style="list-style-type: none"> Datacollectedisappropriate in size and butnotfromproper sources. 	<ul style="list-style-type: none"> Data collected is not soappropriate in size and but frompropersources. 	<ul style="list-style-type: none"> Datacollectedis neitherappropriate in size and norfrompropersources
B	LAB RECORD/WRITTENWORK	NA	NA	Timing ofEvaluation ofExperiment M.M.50=3 M.M.75=4 M.M.100=6	<ul style="list-style-type: none"> On the Same Date ofPerformance 	<ul style="list-style-type: none"> On the Next Turn fromPerformance 	<ul style="list-style-type: none"> BeforeDead Line 	<ul style="list-style-type: none"> On theDead Line
		DataAnalysis M.M.50=3 M.M.75=5 M.M.100=6	DataAnalysis M.M.50=3 M.M.75=5 M.M.100=6	DataAnalysis M.M.50=2 M.M.75=3 M.M.100=4	<ul style="list-style-type: none"> Data collected is exhaustivelyanalyzed & appropriate featuresareselected 	<ul style="list-style-type: none"> Datacollectedisanalyzed & but appropriate featuresarenot selected 	<ul style="list-style-type: none"> Data collected is not analyzedproperly. Features selected arenotappropriate 	<ul style="list-style-type: none"> Datacollectedis notanalyzed & the featuresarenot selected

		Results and Discussion M.M.50=3 M.M.75=5 M.M.100=6	Results and Discussion M.M.50=3 M.M.75=5 M.M.100=6	Results and Discussion M.M.50=2 M.M.75=3 M.M.100=4	<ul style="list-style-type: none"> • All results are very well presented with all variables • Well prepared neat diagrams/plots/tables for all performance measures • Discussed critically behavior of the system with reference to performance measures • Very well discussed pros n cons of outcome 	<ul style="list-style-type: none"> • All results presented but not all variables mentioned • Prepared diagrams/plots/tables for all performance measures but not neat • Discussed behavior of the system with reference to performance measures but not critical • Discussed pros n cons of outcome in brief 	<ul style="list-style-type: none"> • Partial results are included • Prepared diagrams/plots/tables partially for the performance measures • Behavior of the system with reference to performance measures has been superficially presented • Discussed pros n cons of outcome but not so relevant 	<ul style="list-style-type: none"> • Results are included but not as per experimental scenarios • No proper diagrams/plots/tables are prepared • Behavior of the system with reference to performance measures has not been presented • Did not discuss pros n cons of outcome
C	VIVA (PO1, PO10)	Way of presentation M.M.50=2.5 M.M.75=4 M.M.100=5	Way of presentation M.M.50=2.5 M.M.75=4 M.M.100=5	Way of presentation M.M.50=2 M.M.75=3 M.M.100=4	• Presentation was very good	• Presentation was good	• Presentation was satisfactory	• Presentation was poor
		Concept Explanation M.M.50=2.5 M.M.75=4 M.M.100=5	Concept Explanation M.M.50=2.5 M.M.75=4 M.M.100=5	Concept Explanation M.M.50=2 M.M.75=3 M.M.100=4	• Conceptual explanation was excellent	• Conceptual explanation was good	• Conceptual explanation was somewhat good	• Conceptual explanation was Poor
D	ATTENDANCE	NA	NA	Attendance M.M.50=5 M.M.75=8 M.M.100=10	• Present more than 90% of lab sessions	• Present more than 75% of lab sessions	• Present more than 60% of lab sessions	• Present in less than 60% lab sessions

LAB CONDUCTION PLAN**Total number of Experiments - 13****Total number of turns required - 13****Number of turns required for: -**

Experiment Number	Scheduled Week
Experiment-1	Week-1
Experiment-2	Week-2
Experiment-3	Week-3
Experiment-4	Week-4
Experiment-5	Week-5
Experiment-6	Week-6
Experiment-7	Week-7
I Mid Term	Week-8
Experiment-8	Week-9
Experiment-9	Week-10
Experiment-10	Week-11
Experiment-11	Week-12
Experiment-12	Week-13
Experiment-13	Week-14
II Mid Term	Week15

DISTRIBUTION OF LAB HOURS

S. No.	Activity	Distribution of Lab Hours	
		Time (180minute)	Time (120minute)
1	Attendance	5	5
2	Explanation of Experiment & Logic	30	30
3	Performing the Experiment	60	30
4	File Checking	40	20
5	Viva/Quiz	30	20
6	Solving of Queries	15	15

LAB ROTAR PLAN**ROTOR-1**

Ex. No.	NAME OF EXPERIMENTS
1	Introduction: Objective, scope and outcome of the course.
2	Write a simple C program on a 32 bit compiler to understand the concept of array storage, size of a word. The program shall be written illustrating the concept of row major and column major storage. Find the address of element and verify it with the theoretical value. Program may be written for arrays upto 4-dimensions.
3	Simulate a stack, queue, circular queue and dequeue using a one dimensional array as storage element. The program should implement the basic addition, deletion and traversal operations.
4	Represent a 2-variable polynomial using array. Use this representation to implement addition of polynomials.
5	Represent a sparse matrix using array. Implement addition and transposition operations using the representation.
6	Implement singly, doubly and circularly connected linked lists illustrating operations like addition at different locations, deletion from specified locations and traversal.

ROTOR-2

Ex. No.	NAME OF EXPERIMENTS
7	Repeat exercises 2, 3 & 4 with linked structures.
8	Implementation of binary tree with operations like addition, deletion, traversal.
9	Depth first and breadth first traversal of graphs represented using adjacency matrix and list.
10	Implementation of binary search in arrays and on linked Binary Search Tree.
11	Implementation of different sorting algorithm like insertion, quick, heap, bubble and many more sorting algorithms.

GENERAL LAB INSTRUCTIONS

DO'S

1. Enter the lab on time and leave at proper time.
2. Wait for the previous class to leave before the next class enters.
3. Keep the bag outside in the respective racks.
4. Utilize lab hours in the corresponding.
5. Turn off the machine before leaving the lab unless a member of lab staff has specifically told you not to do so.
6. Leave the labs at least as nice as you found them.
7. If you notice a problem with a piece of equipment (e.g., a computer doesn't respond) or the room in general (e.g., cooling, heating, lighting) please report it to lab staff immediately. Do not attempt to fix the problem yourself.

DON'TS

1. Don't abuse the equipment.
2. Do not adjust the heat or air conditioners. If you feel the temperature is not properly set, inform lab staff; we will attempt to maintain a balance that is healthy for people and machines.
3. Do not attempt to reboot a computer. Report problems to lab staff.
4. Do not remove or modify any software or file without permission.
5. Do not remove printers and machines from the network without being explicitly told to do so by lab staff.
6. Don't monopolize equipment. If you're going to be away from your machine for more than 10 or 15 minutes, logout before leaving. This is both for the security of your account, and to ensure that others are able to use the lab resources while you are not.
7. Don't use internet, internet chat of any kind in your regular lab schedule.
8. Do not download or upload of MP3, JPG or MPEG files.
9. No games are allowed in the lab sessions.

10. No hardware including USB drives can be connected or disconnected in the labs without prior permission of the lab in-charge.
11. No food or drink is allowed in the lab or near any of the equipment. Aside from the fact that it leaves a mess and attracts pests, spilling anything on a keyboard or other piece of computer equipment could cause permanent, irreparable, and costly damage. (and in fact *has*) If you need to eat or drink, take a break and do so in the canteen.
12. Don't bring any external material in the lab, except your lab record, copy and books.
13. Don't bring the mobile phones in the lab. If necessary, then keep them in silence mode.
14. Please be considerate of those around you, especially in terms of noise level. While labs are a natural place for conversations of all types, kindly keep the volume turned down.
15. If you are having problems or questions, please go to either the faculty, lab in-charge or the lab supporting staff. They will help you. We need your full support and cooperation for smooth functioning of the lab.

LAB SPECIFIC SAFETY RULES

Before entering in the lab

1. All the students are supposed to prepare the theory regarding the next experiment/Program.
2. Students are supposed to bring their lab records as per their lab schedule.
3. Previous experiment/program should be written in the lab record.
4. If applicable trace paper/graph paper must be pasted in lab record with proper labeling.
5. All the students must follow the instructions, failing which he/she may not be allowed in the lab.

While working in the lab

1. Adhere to experimental schedule as instructed by the lab in-charge/faculty.
2. Get the previously performed experiment/ program signed by the faculty/ lab in charge.
3. Get the Output of current experiment/program checked by the faculty/lab in charge in the lab copy.
4. Each student should work on his/her assigned computer at each turn of the lab.
5. Take responsibility of valuable accessories.

Zero Lab

Turbo C++

C++ tutorial provides basic and advanced concepts of C++. Our C++ tutorial is designed for beginners and professionals.

C++ is an object-oriented programming language. It is an extension to C programming.

Our C++ tutorial includes all topics of C++ such as first example, control statements, objects and classes, inheritance, constructor, destructor, this, static, polymorphism, abstraction, abstract class, interface, namespace, encapsulation, arrays, strings, exception handling, File IO, etc.

What is C++?

C++ is a general purpose, case-sensitive, free-form programming language that supports object-oriented, procedural, and generic programming. C++ is a middle-level language, as it encapsulates both high and low-level language features.

Object-Oriented Programming (OOPs)

C++ supports the object-oriented programming, the four major pillar of object-oriented programming (OOPs) used in C++ are:

1. Inheritance
2. Polymorphism
3. Encapsulation
4. Abstraction

C++ Standard Libraries

Standard C++ programming is divided into three important parts:

- The core library includes the data types, variables and literals, etc.
- The standard library includes the set of functions manipulating strings, files, etc.
- The Standard Template Library (STL) includes the set of methods manipulating a data structure.

Usage of C++

By the help of C++ programming language, we can develop different types of secured and robust applications:

- Window application
- Client-Server application
- Device drivers
- Embedded firmware etc

C++ Program: In this tutorial, all C++ programs are given with C++ compiler so that you can easily change the C++ program code.

File: main.cpp

1. `#include <iostream>`
2. `using namespace std;`
3. `int main() {`
4. `cout << "Hello C++ Programming";`
5. `return 0;`
6. `}`

History of C++ language is interesting to know. Here we are going to discuss brief history of C++ language. **C++ programming language** was developed in 1980 by Bjarne Stroustrup at bell laboratories of AT&T (American Telephone & Telegraph), located in U.S.A.

Bjarne Stroustrup is known as the **founder of C++ language**.

It was developed for adding a feature of **OOP (Object Oriented Programming)** in C without significantly changing the C component.

There are many compilers available for C++. You need to download any one. Here, we are going to use **Turbo C++**. It will work for both C and C++. To install the Turbo C++ software, you need to follow following steps.

1. Download Turbo C++
2. Create turbo c directory inside c drive and extract the tc3.zip inside c:\turboc
3. Double click on install.exe file
4. Click on the tc application file located inside c:\TC\BIN to write the c program

EXPERIMENT-1

OBJECTIVE

Introduction: Objective, scope and outcome of the course.

THEORY

The course is designed to develop skills to design and analyze simple linear and nonlinear data structures. It strengthens the ability to the students to identify and apply the suitable data structure for the given real-world problem. It enables them to gain knowledge in practical applications of data structures. At the end of this lab session, the student will

- Be able to design and analyze the time and space efficiency of the data structure
- Be capable to identify the appropriate data structure for given problem
- Have practical knowledge on the applications of data structures

SCOPE

DSA has great importance in the recruitment process of software companies as well. Recruiters use DSA to test the ability of the programmer because it shows the problem-solving capability of the candidate.

OUTCOMES

Upon the completion of Data Structure & Algorithm practical course, the student will be able to:

1. To utilize searching and sorting algorithms on given values.
2. To analyze the time and space efficiency of the data structure.
3. To evaluate traversing, insertion and deletion operations on Linear and nonlinear data structures.
4. To construct the solutions for real time applications.

EXPERIMENT-2

OBJECTIVE

Write a simple C program on a 32 bit compiler to understand the concept of array storage, size of a word. The program shall be written illustrating the concept of row major and column major storage. Find the address of element and verify it with the theoretical value. Program may be written for arrays up to 4-dimensions.

PROGRAM**Part 1**

```
#include <stdio.h>
int main()
{
    int arr[] = { 1, 2, 3, 4, 7, 98, 0, 12, 35, 99, 14 };
    printf("Number of elements:%d", sizeof(arr) / sizeof(arr[0]));
    return 0;
}
```

Part 2

```
#include <stdio.h>
void main ()
{
    int arr[3][3],i,j;
    for (i=0;i<3;i++)
    {
        for (j=0;j<3;j++)
        {
            printf("Enter a[%d][%d]: ",i,j);
            scanf("%d", &arr[i][j]);
        }
    }
    printf("\n printing the elements ....\n");
    for (i=0;i<3;i++)
    {
        printf("\n");
        for (j=0;j<3;j++)
        {
            printf("%d\t", arr[i][j]);
        }
    }
}
```

EXPERIMENT-3**OBJECTIVE**

Simulate a stack, queue, circular queue and dequeue using a one-dimensional array as storage element.

The program should implement the basic addition, deletion and traversal operations.

PROGRAM**Part 1**

```
#include<stdio.h>

int stack[10], choice, n, top, x, i; // Declaration of variables


void push();
void pop();
void display();


int main()
{
    top = -1; // Initially there is no element in stack
    printf("\n Enter the size of STACK : ");
    scanf("%d", &n);
    printf("\nSTACK IMPLEMENTATION USING ARRAYS\n");
    do
    {
        printf("\n1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n");
        printf("\nEnter the choice : ");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
```

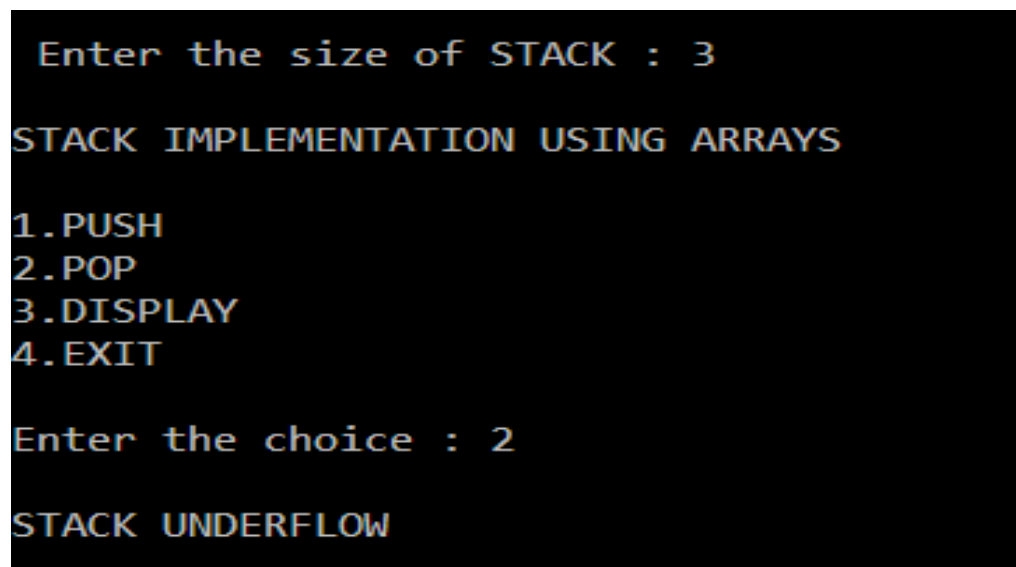
```
pop();
break;
}
case 3:
{
display();
break;
}
case 4:
{
break;
}
default:
{
printf ("\nInvalid Choice\n");
} } }
while(choice!=4);
return 0;
}

void push()
{
if(top >= n - 1)
{
printf ("\nSTACK OVERFLOW\n");
}
else
{
printf("Enter a value to be pushed : ");
scanf("%d",&x);
top++;          // TOP is incremented after an element is pushed
stack[top] = x; // The pushed element is made as TOP
} }

void pop()
{
if(top <= -1)
```

```
{
printf("\nSTACK UNDERFLOW\n");
}
else
{
printf("\nThe popped element is %d",stack[top]);
top--; // Decrement TOP after a pop
}}
void display()
{
if(top >= 0)
{
// Print the stack
printf("\nELEMENTS IN THE STACK\n\n");
for(i = top ; i >= 0 ; i--)
printf("%d\t",stack[i]);
}
else
{
printf("\nEMPTY STACK\n");
}}
}
```

Output:



```
Enter the size of STACK : 3

STACK IMPLEMENTATION USING ARRAYS

1.PUSH
2.POP
3.DISPLAY
4.EXIT

Enter the choice : 2

STACK UNDERFLOW
```

Part 2

```
#include<stdio.h>
#define MAX 50
void enqueue();
void dequeue();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
main()
{
    int choice;
    while(1)
    {
        printf("1.Insert element to queue \n");
        printf("2.Delete element from queue \n");
        printf("3.Display all elements of queue \n");
        printf("4.Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(1);
            default:
                printf("Wrong choice \n");
        } /* End of switch */
    } /* End of while */
} /* End of main() */

void enqueue()
{
    int add_item;
    if (rear == MAX - 1)
        printf("Queue Overflow \n");
```

```
else
{
    if (front == - 1)
        /*If queue is initially empty */
        front = 0;
    printf("Inset the element in queue : ");
    scanf("%d", &add_item);
    rear = rear + 1;
    queue_array[rear] = add_item;
}
} /* End of insert() */

void dequeue()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n", queue_array[front]);
        front = front + 1;
    }
} /* End of delete() */

void display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
}
```

Output:

1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit

Enter your choice : 1

Inset the element in queue : 10

1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit

Enter your choice : 1

Inset the element in queue : 15

1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit

Enter your choice : 1

Inset the element in queue : 20

1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit

Enter your choice : 1

Inset the element in queue : 30

1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit

Enter your choice : 2

Element deleted from queue is : 10

1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit

Enter your choice: 3

Queue is: 15 20 30

1. Insert element to queue
2. Delete element from queue
3. Display all elements of queue
4. Quit

Enter your choice : 4

Part 3

```
#include <stdio.h>
#define max 6
int queue[max]; // array declaration
int front=-1;
int rear=-1;
// function to insert an element in a circular queue
void enqueue(int element)
{
    if(front== -1 && rear== -1) // condition to check queue is empty
    {
        front=0;
        rear=0;
        queue[rear]=element;
    }
    else if((rear+1)%max==front) // condition to check queue is full
    {
        printf("Queue is overflow...");
    }
    else
    {
        rear=(rear+1)%max;    // rear is incremented
        queue[rear]=element;  // assigning a value to the queue at the rear position.
    } }
// function to delete the element from the queue
int dequeue()
{
    if((front== -1) && (rear== -1)) // condition to check queue is empty
    {
        printf("\nQueue is underflow...");
    }
    else if(front==rear)
    {
        printf("\nThe dequeued element is %d", queue[front]);
        front=-1;
    }
}
```

```

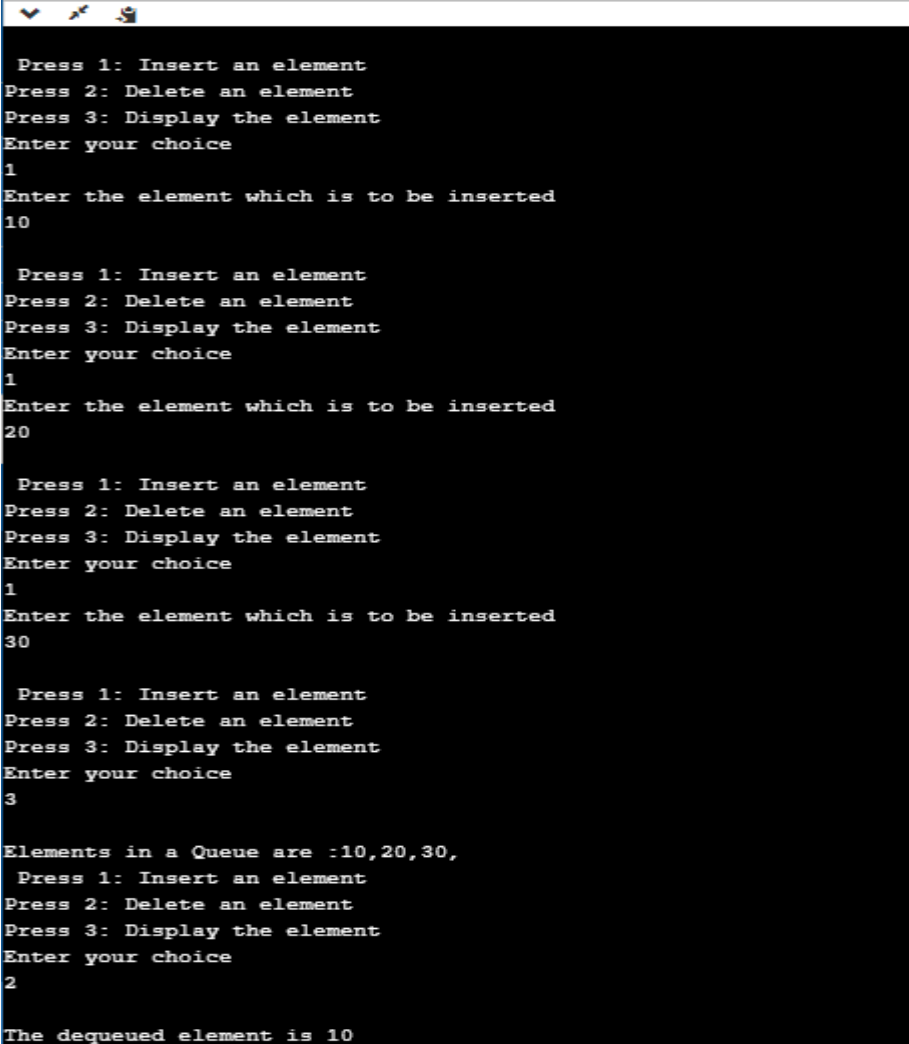
    rear=-1;
}
else
{
    printf("\nThe dequeued element is %d", queue[front]);
    front=(front+1)%max;
} }
// function to display the elements of a queue
void display()
{
    int i=front;
    if(front== -1 && rear== -1)
    {
        printf("\n Queue is empty...");
    }
    else
    {
        printf("\nElements in a Queue are :");
        while(i<=rear)
        {
            printf("%d,", queue[i]);
            i=(i+1)%max;
        } }
}

int main()
{
    int choice=1, x; // variables declaration
    while(choice<4 && choice!=0) // while loop
    { printf("\n Press 1: Insert an element");
      printf("\nPress 2: Delete an element");
      printf("\nPress 3: Display the element");
      printf("\nEnter your choice");
      scanf("%d", &choice);
      switch(choice)
      { case 1:
        printf("Enter the element which is to be inserted");

```

```
scanf("%d", &x);
enqueue(x);
break;
case 2:
dequeue();
break;
case 3:
display();
}}
return 0;
}
```

Output:



```
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
1
Enter the element which is to be inserted
10

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
1
Enter the element which is to be inserted
20

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
1
Enter the element which is to be inserted
30

Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
3

Elements in a Queue are :10,20,30,
Press 1: Insert an element
Press 2: Delete an element
Press 3: Display the element
Enter your choice
2

The dequeued element is 10
```

Part 4

```
#include <stdio.h>
#define size 5
int deque[size];
int f = -1, r = -1;
// insert_front function will insert the value from the front
void insert_front(int x)
{
    if((f==0 && r==size-1) || (f==r+1))
    {
        printf("Overflow");
    }
    else if((f== -1) && (r== -1))
    {
        f=r=0;
        deque[f]=x;
    }
    else if(f==0)
    {
        f=size-1;
        deque[f]=x;
    }
    else
    {
        f=f-1;
        deque[f]=x;
    }
}
// insert_rear function will insert the value from the rear
void insert_rear(int x)
{
    if((f==0 && r==size-1) || (f==r+1))
    {
        printf("Overflow");
    }
}
```

```

else if((f==-1) && (r==-1))
{
    r=0;
    deque[r]=x;
}
else if(r==size-1)
{
    r=0;
    deque[r]=x;
}
else
{
    r++;
    deque[r]=x;
}

}

// display function prints all the value of deque.
void display()
{
    int i=f;
    printf("\nElements in a deque are: ");

    while(i!=r)
    {
        printf("%d ",deque[i]);
        i=(i+1)%size;
    }
    printf("%d",deque[r]);
}

// getfront function retrieves the first value of the deque.
void getfront()
{
    if((f==-1) && (r==-1))
    {

```

```
    printf("Deque is empty");
}
else
{
    printf("\nThe value of the element at front is: %d", deque[f]);
} }
```

// getrear function retrieves the last value of the deque.

```
void getrear()
{
    if((f== -1) && (r== -1))
    {
        printf("Deque is empty");
    }
    else
    {
        printf("\nThe value of the element at rear is %d", deque[r]);
    } }
```

// delete_front() function deletes the element from the front

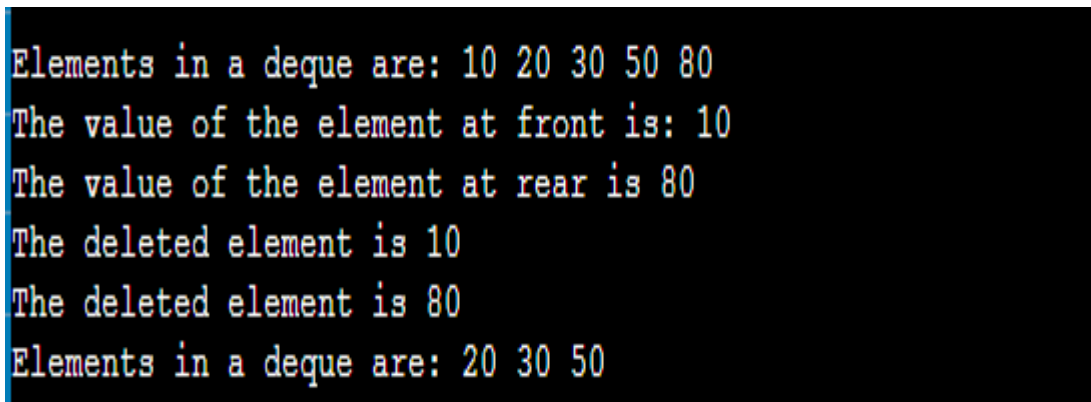
```
void delete_front()
{
    if((f== -1) && (r== -1))
    {
        printf("Deque is empty");
    }
    else if(f==r)
    {
        printf("\nThe deleted element is %d", deque[f]);
        f=-1;
        r=-1;
    }
    else if(f==(size-1))
    {
        printf("\nThe deleted element is %d", deque[f]);
        f=0;
    }
}
```

```

else
{
    printf("\nThe deleted element is %d", deque[f]);
    f=f+1;
} }
// delete_rear() function deletes the element from the rear
void delete_rear()
{
    if((f== -1) && (r== -1))
    {
        printf("Deque is empty");
    }
    else if(f==r)
    {
        printf("\nThe deleted element is %d", deque[r]);
        f=-1;
        r=-1;
    }
    else if(r==0)
    {
        printf("\nThe deleted element is %d", deque[r]);
        r=size-1;
    }
    else
    {
        printf("\nThe deleted element is %d", deque[r]);
        r=r-1;
    }
}
int main()
{
    insert_front(20);
    insert_front(10);
    insert_rear(30);
    insert_rear(50);

```

```
insert_rear(80);  
display(); // Calling the display function to retrieve the values of deque  
getfront(); // Retrieve the value at front-end  
getrear(); // Retrieve the value at rear-end  
delete_front();  
delete_rear();  
display(); // calling display function to retrieve values after deletion  
return 0;  
}
```

Output:

```
Elements in a deque are: 10 20 30 50 80  
The value of the element at front is: 10  
The value of the element at rear is 80  
The deleted element is 10  
The deleted element is 80  
Elements in a deque are: 20 30 50
```


EXPERIMENT-4

OBJECTIVE

Represent a 2-variable polynomial using array. Use this representation to implement addition of polynomials.

PROGRAM

Part 1

```
#include <stdio.h>
#include<stdlib.h>
struct Term
{
    int coeff;
    int exp;
};

struct Poly
{
    int n;
    struct Term *terms;
};

void create (struct Poly *p)
{
    int i;

    printf ("Enter Number of terms: ");
    scanf ("%d", &p->n);

    p->terms = (struct Term *) malloc (p->n * sizeof (struct Term));

    printf ("Enter terms:\n");
    for (i = 0; i < p->n; i++)
        scanf ("%d%d", &p->terms[i].coeff, &p->terms[i].exp);
    printf ("\n");
}

void display (struct Poly p)
{
    int i;
    for (i = 0; i < p.n; i++)
    {
        printf ("%dx%d", p.terms[i].coeff, p.terms[i].exp);
        if (i + 1 < p.n)
            printf (" + ");
    }
    printf ("\n");
}

struct Poly *add (struct Poly *p1, struct Poly *p2)
```

```

{
    int i, j, k;
    struct Poly *sum;

    sum = (struct Poly *) malloc (sizeof (struct Poly));
    sum->terms = (struct Term *) malloc ((p1->n + p2->n) * sizeof (struct Term));

    i = j = k = 0;

    while (i < p1->n && j < p2->n)
    {
        if (p1->terms[i].exp > p2->terms[j].exp)
            sum->terms[k++] = p1->terms[i++];
        else if (p1->terms[i].exp < p2->terms[j].exp)
            sum->terms[k++] = p2->terms[j++];
        else
        {
            sum->terms[k].exp = p1->terms[i].exp;
            sum->terms[k++].coeff = p1->terms[i++].coeff + p2->terms[j++].coeff;
        }
    }

    for (; i < p1->n; i++)
        sum->terms[k++] = p1->terms[i];
    for (; j < p2->n; j++)
        sum->terms[k++] = p2->terms[j];

    sum->n = k;
    return sum;
}

int main()
{
    struct Poly p1, p2, *p3;

    printf ("Enter Polynomial 1:\n");
    create (&p1);
    printf ("Enter Polynomial 2:\n");
    create (&p2);

    p3 = add (&p1, &p2);
    printf ("\n");

    printf ("Polynomial 1 is: ");
    display (p1);
    printf ("\n");

    printf ("Polynomial 2 is: ");
    display (p2);
    printf ("\n");

    printf ("Polynomial 3 is: ");
    display (*p3);

    return 0;
}

```

Output:

```

M:\##DS\#6_Sparse Matrix & Polynomial Representation\8_Polynomial Code\Program.exe
Enter Polynomial 1:
Enter Number of terms: 3
Enter terms:
4 4
3 2
2 0

Enter Polynomial 2:
Enter Number of terms: 3
Enter terms:
5 3
9 1
1 0

Polynomial 1 is: 4x4 + 3x2 + 2x0
Polynomial 2 is: 5x3 + 9x1 + 1x0
Polynomial 3 is: 4x4 + 5x3 + 3x2 + 9x1 + 3x0

-----
Process exited after 16.74 seconds with return value 0
Press any key to continue . . .

```

Part 2

```

#include<stdio.h>

/* declare structure for polynomial */
struct poly
{
    int coeff;
    int expo;
};
/* declare three arrays p1, p2, p3 of type structure poly.
 * each polynomial can have maximum of ten terms
 * addition result of p1 and p2 is stored in p3 */

struct poly p1[10],p2[10],p3[10];

/* function prototypes */
int readPoly(struct poly []);

```

```
int addPoly(struct poly [],struct poly [],int ,int ,struct poly []);
void displayPoly( struct poly [],int terms);
```

```
int main()
{
    int t1,t2,t3;

    /* read and display first polynomial */
    t1=readPoly(p1);
    printf(" \n First polynomial : ");
    displayPoly(p1,t1);
    /* read and display second polynomial */
    t2=readPoly(p2);
    printf(" \n Second polynomial : ");
    displayPoly(p2,t2);

    /* add two polynomials and display resultant polynomial */
    t3=addPoly(p1,p2,t1,t2,p3);
    printf(" \n\n Resultant polynomial after addition : ");
    displayPoly(p3,t3);
    printf("\n");

    return 0;
}
```

```
int readPoly(struct poly p[10])
{
    int t1,i;

    printf("\n\n Enter the total number of terms in the polynomial:");
    scanf("%d",&t1);

    printf("\n Enter the COEFFICIENT and EXPONENT in DESCENDING ORDER\n");
    for(i=0;i<t1;i++)
    {
        printf("  Enter the Coefficient(%d): ",i+1);
        scanf("%d",&p[i].coeff);
        printf("  Enter the exponent(%d): ",i+1);
        scanf("%d",&p[i].expo);    /* only statement in loop */
    }
    return(t1);
}
```

```
int addPoly(struct poly p1[10],struct poly p2[10],int t1,int t2,struct poly p3[10])
{
    int i,j,k;
    i=0;
```

```
j=0;
k=0;

while(i<t1 && j<t2)
{
    if(p1[i].expo==p2[j].expo)
    {
        p3[k].coeff=p1[i].coeff + p2[j].coeff;
        p3[k].expo=p1[i].expo;

        i++;
        j++;
        k++;
    }
    else if(p1[i].expo>p2[j].expo)
    {
        p3[k].coeff=p1[i].coeff;
        p3[k].expo=p1[i].expo;
        i++;
        k++;
    }
    else
    {
        p3[k].coeff=p2[j].coeff;
        p3[k].expo=p2[j].expo;
        j++;
        k++;
    }
}

/* for rest over terms of polynomial 1 */
while(i<t1)
{
    p3[k].coeff=p1[i].coeff;
    p3[k].expo=p1[i].expo;
    i++;
    k++;
}

/* for rest over terms of polynomial 2 */
while(j<t2)
{
    p3[k].coeff=p2[j].coeff;
    p3[k].expo=p2[j].expo;
    j++;
    k++;
}
```

```

        return(k); /* k is number of terms in resultant polynomial*/
    }

    void displayPoly(struct poly p[10],int term)
    {
        int k;

        for(k=0;k<term-1;k++)
            printf("%d(x^%d)+",p[k].coeff,p[k].expo);
            printf("%d(x^%d)",p[term-1].coeff,p[term-1].expo);
    }

```

Output:

```

Enter the total number of terms in the polynomial:4
Enter the COEFFICIENT and EXPONENT in DESCENDING ORDER
Enter the Coefficient(1): 3
Enter the exponent(1): 4
Enter the Coefficient(2): 7
Enter the exponent(2): 3
Enter the Coefficient(3): 5
Enter the exponent(3): 1
Enter the Coefficient(4): 8
Enter the exponent(4): 0

First polynomial : 3(x^4)+7(x^3)+5(x^1)+8(x^0)

Enter the total number of terms in the polynomial:5
Enter the COEFFICIENT and EXPONENT in DESCENDING ORDER
Enter the Coefficient(1): 7
Enter the exponent(1): 5
Enter the Coefficient(2): 6
Enter the exponent(2): 4
Enter the Coefficient(3): 8
Enter the exponent(3): 2
Enter the Coefficient(4): 9
Enter the exponent(4): 1
Enter the Coefficient(5): 2
Enter the exponent(5): 0
Second polynomial : 7(x^5)+6(x^4)+8(x^2)+9(x^1)+2(x^0)

Resultant polynomial after addition :
7(x^5)+9(x^4)+7(x^3)+8(x^2)+14(x^1)+10(x^0)

```

EXPERIMENT-5**OBJECTIVE**

Represent a sparse matrix using array. Implement addition and transposition operations using the representation.

PROGRAM

```
#include<stdio.h>
#define MAX 20
void printsparse(int[][3]);
void readsparse(int[][3]);
void transpose(int[][3],int[][3]);
int main()
{
int b1[MAX][3],b2[MAX][3],m,n;
printf("Enter the size of matrix (rows,columns):");
scanf("%d%d",&m,&n);
b1[0][0]=m;
b1[0][1]=n;
readsparse(b1);
transpose(b1,b2);
printsparse(b2);
}
void readsparse(int b[MAX][3])
{
int i,t;
printf("\nEnter no. of non-zero elements:");
scanf("%d",&t);
b[0][2]=t;
for(i=1;i<=t;i++)
{
printf("\nEnter the next triple(row,column,value):");
scanf("%d%d%d",&b[i][0],&b[i][1],&b[i][2]);
}
}
```

```

void printspase(int b[MAX][3])
{
    int i,n;
    n=b[0][2]; //no of 3-triples
    printf("\nAfter Transpose:\n");
    printf("\nrow\t\tcolumn\t\tvalue\n");
    for(i=0;i<=n;i++)
        printf("%d\t\t%d\t\t%d\n",b[i][0],b[i][1],b[i][2]);
}

void transpose(int b1[][3],int b2[][3])
{
    int i,j,k,n;
    b2[0][0]=b1[0][1];
    b2[0][1]=b1[0][0];
    b2[0][2]=b1[0][2];
    k=1;
    n=b1[0][2];
    for(i=0;i<b1[0][1];i++)
        for(j=1;j<=n;j++)
            //if a column number of current triple==i then insert the current triple in b2
            if(i==b1[j][1])
            {
                b2[k][0]=i;
                b2[k][1]=b1[j][0];
                b2[k][2]=b1[j][2];
                k++;
            }
}

```

Output:

```

Enter the size of matrix (rows,columns):3 4
Enter no. of non-zero elements:4
Enter the next triple(row,column,value):1 0 5
Enter the next triple(row,column,value):1 2 3

```


Enter the next triple(row,column,value):2 1 1

Enter the next triple(row,column,value):2 3 2

After Transpose:

row column value

4 3 4

0 1 5

1 2 1

2 1 3

3 2 2

EXPERIMENT-6**OBJECTIVE**

Implement singly, doubly and circularly connected linked lists illustrating operations like addition at different locations, deletion from specified locations and traversal.

PROGRAM**Part 1**

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int Data;
    Struct Node *next;
};
void insertStart (struct Node **head, int data)
{
    struct Node *newNode = (struct Node *) malloc (sizeof (struct Node));
    newNode -> data = data;
    newNode -> next = *head;

    //changing the new head to this freshly entered node
    *head = newNode;
}
void deleteStart(struct Node **head)
{
    struct Node *temp = *head;
    // if there are no nodes in Linked List can't delete
    if (*head == NULL)
    {
        printf ("Linked List Empty, nothing to delete");
        return;
    }
    // move head to next node
    *head = (*head)->next;
```

```
    free (temp);
}
void display(struct Node* node)
{
    printf("Linked List: ");

    // as linked list will end when Node is Null
    while(node!=NULL){
        printf("%d ",node->data);
        node = node->next;
    }
    printf("\n");
}
int main ()
{
    struct Node *head = NULL;
    // Need '&' i.e. address as we need to change head
    insertStart (&head, 100);
    insertStart (&head, 80);
    insertStart (&head, 60);
    insertStart (&head, 40);
    insertStart (&head, 20);
    // No Need for '&' as not changing head in display operation
    display (head);
    deleteStart (&head);
    deleteStart (&head);
    display (head);
    return 0;
}
```

Output:

```
100 Inserted
80 Inserted
60 Inserted
40 Inserted
20 Inserted
Linked List: 20 40 60 80 100
20 deleted
40 deleted
Linked List: 60 80 100
```

Part 2

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
struct node {
    int data;
    int key;
    struct node *next;
    struct node *prev;
};
//this link always point to first Link
struct node *head = NULL;
//this link always point to last Link
struct node *last = NULL;
struct node *current = NULL;
//is list empty
bool isEmpty() {
    return head == NULL;
}
```

```

int length() {
    int length = 0;
    struct node *current;
    for(current = head; current != NULL; current = current->next){
        length++;
    }
    return length;
}

//display the list in from first to last
void displayForward() {
    //start from the beginning
    struct node *ptr = head;
    //navigate till the end of the list
    printf("\n[ ");
    while(ptr != NULL) {
        printf("(%d,%d) ",ptr->key,ptr->data);
        ptr = ptr->next;
    }
    printf("]");
}

//display the list from last to first
void displayBackward() {
    //start from the last
    struct node *ptr = last;
    //navigate till the start of the list
    printf("\n[ ");
    while(ptr != NULL) {
        //print data
        printf("(%d,%d) ",ptr->key,ptr->data);
        //move to next item
        ptr = ptr ->prev;
    }
}

//insert link at the first location
void insertFirst(int key, int data) {

```

```
//create a link
struct node *link = (struct node*) malloc(sizeof(struct node));
link->key = key;
link->data = data;
if(isEmpty()) {
    //make it the last link
    last = link;
} else {
    //update first prev link
    head->prev = link;
}
//point it to old first link
link->next = head;
//point first to new first link
head = link;
}
//insert link at the last location
void insertLast(int key, int data) {
    //create a link
    struct node *link = (struct node*) malloc(sizeof(struct node));
    link->key = key;
    link->data = data;
    if(isEmpty()) {
        //make it the last link
        last = link;
    } else {
        //make link a new last link
        last->next = link;
        //mark old last node as prev of new link
        link->prev = last;
    }
    //point last to new last node
    last = link;
}
```

```
//delete first item
struct node* deleteFirst() {
    //save reference to first link
    struct node *tempLink = head;
    //if only one link
    if(head->next == NULL){
        last = NULL;
    } else {
        head->next->prev = NULL;
    }
    head = head->next;
    //return the deleted link
    return tempLink;
}

//delete link at the last location
struct node* deleteLast() {
    //save reference to last link
    struct node *tempLink = last;
    //if only one link
    if(head->next == NULL) {
        head = NULL;
    } else {
        last->prev->next = NULL;
    }
    last = last->prev;
    //return the deleted link
    return tempLink;
}

//delete a link with given key

struct node* delete(int key) {
    //start from the first link
    struct node* current = head;
    struct node* previous = NULL;
```

```

//if list is empty
if(head == NULL) {
    return NULL;
}
//navigate through list
while(current->key != key) {
    //if it is last node
    if(current->next == NULL) {
        return NULL;
    } else {
        //store reference to current link
        previous = current;
        //move to next link
        current = current->next;
    }
}
//found a match, update the link
if(current == head) {
    //change first to point to next link
    head = head->next;
} else {
    //bypass the current link
    current->prev->next = current->next;
}
if(current == last) {
    //change last to point to prev link
    last = current->prev;
} else {
    current->next->prev = current->prev;
}
return current;
}

```

```

bool insertAfter(int key, int newKey, int data) {
    //start from the first link

```



```
struct node *current = head;
//if list is empty
if(head == NULL) {
    return false;
}
//navigate through list
while(current->key != key) {
    //if it is last node
    if(current->next == NULL) {
        return false;
    } else {
        //move to next link
        current = current->next;
    }
}
//create a link
struct node *newLink = (struct node*) malloc(sizeof(struct node));
newLink->key = newKey;
newLink->data = data;
if(current == last) {
    newLink->next = NULL;
    last = newLink;
} else {
    newLink->next = current->next;
    current->next->prev = newLink;
}
newLink->prev = current;
current->next = newLink;
return true;
}
void main() {
    insertFirst(1,10);
    insertFirst(2,20);
    insertFirst(3,30);
    insertFirst(4,1);
```

```

insertFirst(5,40);
insertFirst(6,56);
printf("\nList (First to Last): ");
displayForward();
printf("\n");
printf("\nList (Last to first): ");
displayBackward();
printf("\nList , after deleting first record: ");
deleteFirst();
displayForward();
printf("\nList , after deleting last record: ");
deleteLast();
displayForward();
printf("\nList , insert after key(4) : ");
insertAfter(4,7, 13);
displayForward();
printf("\nList , after delete key(4) : ");
delete(4);
displayForward();
}

```

Output:

List (First to Last):

[(6,56) (5,40) (4,1) (3,30) (2,20) (1,10)]

List (Last to first):

[(1,10) (2,20) (3,30) (4,1) (5,40) (6,56)]

List , after deleting first record:

[(5,40) (4,1) (3,30) (2,20) (1,10)]

List , after deleting last record:

[(5,40) (4,1) (3,30) (2,20)]

List , insert after key(4) :

[(5,40) (4,1) (7,13) (3,30) (2,20)]

List , after delete key(4) :

[(5,40) (4,13) (3,30) (2,20)]

Part 3

```

#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *head;
void beginsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
void main ()
{
    int choice =0;
    while(choice != 7)
    {
        printf("\n*****Main Menu*****\n");
        printf("\nChoose one option from the following list ...\n");
        printf("\n===== \n");
        printf("\n1.Insert in beginning\n2.Insert at last\n3.Delete from Beginning\n4.Delete from
last\n5.Search for an element\n6.Show\n7.Exit\n");
        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);
        switch(choice)
        {
            case 1:
                beginsert();
                break;
            case 2:

```

```
        lastinsert();
        break;
    case 3:
        begin_delete();
        break;
    case 4:
        last_delete();
        break;
    case 5:
        search();
        break;
    case 6:
        display();
        break;
    case 7:
        exit(0);
        break;
    default:
        printf("Please enter valid choice..");
    }
}
}
void beginsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter the node data?");
        scanf("%d",&item);
```

```

    ptr -> data = item;
    if(head == NULL)
    {
        head = ptr;
        ptr -> next = head;
    }
    else
    {
        temp = head;
        while(temp->next != head)
            temp = temp->next;
        ptr->next = head;
        temp -> next = ptr;
        head = ptr;
    }
    printf("\nnode inserted\n");
}

void lastinsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW\n");
    }
    else
    {
        printf("\nEnter Data?");
        scanf("%d",&item);
        ptr->data = item;
        if(head == NULL)
        {
            head = ptr;

```

```

        ptr -> next = head;
    }
    else
    {
        temp = head;
        while(temp -> next != head)
        {
            temp = temp -> next;
        }
        temp -> next = ptr;
        ptr -> next = head;
    }
    printf("\nnode inserted\n");
}

}

void begin_delete()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nUNDERFLOW");
    }
    else if(head->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        while(ptr -> next != head)
            ptr = ptr -> next;
        ptr->next = head->next;
        free(head);
    }
}

```

```

        head = ptr->next;
        printf("\nnode deleted\n");
    }
}

void last_delete()
{
    struct node *ptr, *preptr;
    if(head==NULL)
    {
        printf("\nUNDERFLOW");
    }
    else if (head ->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        while(ptr ->next != head)
        {
            preptr=ptr;
            ptr = ptr->next;
        }
        preptr->next = ptr -> next;
        free(ptr);
        printf("\nnode deleted\n");
    }
}

void search()
{
    struct node *ptr;
    int item,i=0,flag=1;

```

```
ptr = head;
if(ptr == NULL)
{
    printf("\nEmpty List\n");
}
else
{
    printf("\nEnter item which you want to search?\n");
    scanf("%d",&item);
    if(head ->data == item)
    {
        printf("item found at location %d",i+1);
        flag=0;
    }
    else
    {
        while (ptr->next != head)
        {
            if(ptr->data == item)
            {
                printf("item found at location %d ",i+1);
                flag=0;
                break;
            }
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
    }
    if(flag != 0)
    {
        printf("Item not found\n");
    }
}
```



```
    } } }
```

```
void display()
{
    struct node *ptr;
    ptr=head;
    if(head == NULL)
    {
        printf("\nnothing to print");
    }
    else
    { printf("\n printing values ... \n");
      while(ptr -> next != head)
      {
          printf("%d\n", ptr -> data);
          ptr = ptr -> next;
      }
      printf("%d\n", ptr -> data);
    } }
```

Output:

Choose one option from the following list ...

```
=====
```

- 1.Insert in begining
- 2.Insert at last
- 3.Delete from Beginning
- 4.Delete from last
- 5.Search for an element
- 6.Show
- 7.Exit

Enter your choice?

1

Enter the node data?10

node inserted

EXPERIMENT-7

OBJECTIVE

Repeat experiment 3, 4, 5 with linked structure.

PROGRAM**Part 1**

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

// Structure to create a node with data and the next pointer
struct node {
    int info;
    struct node *ptr;
}*top,*top1,*temp;
int count = 0;
// Push() operation on a stack
void push(int data) {
    if (top == NULL)
    {
        top =(struct node *)malloc(1*sizeof(struct node));
        top->ptr = NULL;
        top->info = data;
    }
    else
    {
        temp =(struct node *)malloc(1*sizeof(struct node));
        temp->ptr = top;
        temp->info = data;
        top = temp;
    }
    count++;
    printf("Node is Inserted\n\n");
}

int pop() {
```

```
    top1 = top;

    if (top1 == NULL)
    {
        printf("\nStack Underflow\n");
        return -1;
    }
    else
        top1 = top1->ptr;
    int popped = top->info;
    free(top);
    top = top1;
    count--;
    return popped;
}

void display() {
    // Display the elements of the stack
    top1 = top;

    if (top1 == NULL)
    {
        printf("\nStack Underflow\n");
        return;
    }

    printf("The stack is \n");
    while (top1 != NULL)
    {
        printf("%d--->", top1->info);
        top1 = top1->ptr;
    }
    printf("NULL\n\n");
}
```

```
int main() {
    int choice, value;
    printf("\nImplementation of Stack using Linked List\n");
    while (1) {
        printf("\n1. Push\n2. Pop\n3. Display\n4. Exit\n");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("\nEnter the value to insert: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                printf("Popped element is :%d\n", pop());
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
                break;
            default:
                printf("\nWrong Choice\n");
        }
    }
}
```

Output:

Implementation of Stack using Linked List

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 1

Enter the value to insert: 12

Node is Inserted

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 1

Enter the value to insert: 45

Node is Inserted

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 1

Enter the value to insert: 56

Node is Inserted

1. Push
2. Pop
3. Display
4. Exit

Enter your choice : 3

The stack is

56--->45--->12--->NULL

Part 2

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

// Structure to create a node with data and the next pointer
struct node {
    int data;
    struct node * next;
};

struct node * front = NULL;
struct node * rear = NULL;

// Enqueue() operation on a queue
void enqueue(int value) {
    struct node * ptr;
    ptr = (struct node * ) malloc(sizeof(struct node));
    ptr -> data = value;
    ptr -> next = NULL;
    if ((front == NULL) && (rear == NULL)) {
        front = rear = ptr;
    } else {
        rear -> next = ptr;
        rear = ptr;
    }
    printf("Node is Inserted\n\n");
}

// Dequeue() operation on a queue
int dequeue() {
    if (front == NULL) {
        printf("\nUnderflow\n");
        return -1;
    } else {
        struct node * temp = front;
        int temp_data = front -> data;
        front = front -> next;
        free(temp);
        return temp_data;
    }
}
```

```

    } // Display all elements of the queue
void display() {
    struct node * temp;
    if ((front == NULL) && (rear == NULL)) {
        printf("\nQueue is Empty\n");
    } else {
        printf("The queue is \n");
        temp = front;
        while (temp) {
            printf("%d--->", temp->data);
            temp = temp->next;
        }
        printf("NULL\n\n");
    }
}

int main() {
    int choice, value;
    printf("\nImplementation of Queue using Linked List\n");
    while (choice != 4) {
        printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("\nEnter the value to insert: ");
                scanf("%d", &value);
                enqueue(value);
                break;
            case 2:
                printf("Popped element is :%d\n", dequeue());
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
        }
    }
}

```

```

        break;
    default:
        printf("\nWrong Choice\n");
    } }
    return 0;}

```

Output:

Implementation of Queue using Linked List

1. Enqueue

2. Dequeue

3. Display

4. Exit

Enter your choice: 1

Enter the value to insert: 12

Node is Inserted

1.Enqueue

2.Dequeue

3.Display

4.Exit

Enter your choice: 1

Enter the value to insert: 45

Node is Inserted

1.Enqueue

2.Dequeue

3.Display

4.Exit

Enter your choice: 1

Enter the value to insert: 56

Node is Inserted

1.Enqueue

2.Dequeue

3.Display

4.Exit

Enter your choice : 3

The queue is

12--->45--->56--->NULL

EXPERIMENT-8**OBJECTIVE**

Implementation of binary tree with operations like addition, deletion, traversal.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
struct bst{
    int data;
    struct bst* left;
    struct bst* right;
}*root;
struct bst* insert(struct bst* root, int num)
{
    if(root==NULL)
    {
        root=(struct bst*)malloc(sizeof(struct bst));
        root->data=num;
        root->left=NULL;
        root->right=NULL;
    }
    else
    {
        if(num<root->data)
        {
            root->left=insert(root->left,num);
        }
        else
        {
            root->right=insert(root->right,num);
        }
    }
}
```

```
        return root;
    }
void in_order(struct bst* root)
{
    if(root!=NULL)
    {
        in_order(root->left);
        printf("%d\t",root->data);
        in_order(root->right);
    }
}
void search(struct bst* root,int num)
{
    if(root==NULL)
    {
        printf("\nnumber not found.");
    }
    else if(root->data==num)
    {
        printf("\ndata is found %d",num);
    }
    else if(root->data>num)
    {
        search(root->left,num);
    }
    else
    {
        search(root->right,num);
    }
}
```

```
struct bst* Delete(struct bst* root,int num)
{
    if(root==NULL)
    {
        printf("\nnumber not found.");
    }
    else if(root->data==num)
    {
        if(root->left==NULL&&root->right==NULL)
        {
            struct bst * temp = root;
            free(temp);
            printf("\nNo. is deleted.");
            return NULL;
        }
        else if(root->left==NULL && root->right!=NULL)
        {
            struct bst*temp=root;
            root=root->right;
            free(temp);
            printf("\nNo. is deleted.");
            return root;
        }
        else if(root->left!=NULL && root->right==NULL)
        {
            struct bst*temp=root;
            root=root->left;
            free(temp);
            printf("\nNo. is deleted.");
            return root;
        }
    }
}
```

```

        else
        {
            struct bst *temp;temp=root; root=root->right;
            while(root->left!=NULL)
            {
                root=root->left;
            }
            temp->data=root->data;
            temp->right=Delete(temp->right, root-
>data);return temp;
        }
    }
    else if(root->data>num)
    {
        root->left=Delete(root->left,num);
    }
    else
    {
        root->right=Delete(root->right,num);
    }
    return root;
}

void main()
{
    int arr[] = {20, 17, 6, 8,9, 25, 5, 27,7};
    int i;
    for(i=0;i<9;i++)
        root=insert(root,arr[i]);

    printf("\nIN order treversd list is : ");
    in_order(root);
    search(root,20);
}

```

```
    root = Delete(root,20);  
    printf("\nIN order treversd list is : ");  
    in_order(root);  
}
```

OUTPUT:

```
IN order treversd list is : 5  6    7    8    9    17    20    25    27  
data is found 20  
No. is deleted.  
IN order treversd list is : 5  6    7    8    9    17    25    27  
-----  
Process exited after 0.05983 seconds with return value 0  
Press any key to continue . . .
```

EXPERIMENT-9**OBJECTIVE**

Depth first and breadth first traversal of graphs represented using adjacency matrix and list.

PROGRAM

```
// BFS in C
#include <stdio.h>
#include <stdlib.h>
#define SIZE 40
struct queue {
    int items[SIZE];
    int front;
    int rear;
};
struct queue* createQueue();
void enqueue(struct queue* q, int);
int dequeue(struct queue* q);
void display(struct queue* q);
int isEmpty(struct queue* q);
void printQueue(struct queue* q);
struct node {
    int vertex;
    struct node* next;
};
struct node* createNode(int);
struct Graph {
    int numVertices;
    struct node** adjLists;
    int* visited;
};
// BFS algorithm
```

```

void bfs(struct Graph* graph, int startVertex) {
    struct queue* q = createQueue();
    graph->visited[startVertex] = 1; enqueue(q, startVertex);
    while (!isEmpty(q)) {
        printQueue(q);
        int currentVertex = dequeue(q);
        printf("Visited %d\n", currentVertex);
        struct node* temp = graph->adjLists[currentVertex];
        while (temp) {
            int adjVertex = temp->vertex;
            if (graph->visited[adjVertex] == 0) {
                graph->visited[adjVertex] = 1;
                enqueue(q, adjVertex);
            }
            temp = temp->next;
        }
    }
}

// Creating a node
struct node* createNode(int v) {
    struct node* newNode = malloc(sizeof(struct node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode; }

// Creating a graph
struct Graph* createGraph(int vertices) {
    struct Graph* graph = malloc(sizeof(struct Graph));
    graph->numVertices = vertices;
    graph->adjLists = malloc(vertices * sizeof(struct node*));
    graph->visited = malloc(vertices * sizeof(int));
    int i;
    for (i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
    }
    graph->visited[i] = 0; } return graph; }

```

```
// Add edge
void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to dest
    struct node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;
    // Add edge from dest to src
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;}

// Create a queue
struct queue* createQueue() {
    struct queue* q = malloc(sizeof(struct queue));
    q->front = -1;
    q->rear = -1;
    return q;}

// Check if the queue is empty
int isEmpty(struct queue* q) {
    if (q->rear == -1)
        return 1;
    else
        return 0;}

// Adding elements into queue
void enqueue(struct queue* q, int value) {
    if (q->rear == SIZE - 1)
        printf("\nQueue is Full!!");
    else {
        if (q->front == -1)

            q->front = 0;
        q->rear++;
        q->items[q->rear] = value; }}
```



```
// Removing elements from queue
int dequeue(struct queue* q) {
    int item;
    if (isEmpty(q)) {
        printf("Queue is empty");
        item = -1;
    } else {
        item = q->items[q->front];
        q->front++;
        if (q->front > q->rear) {
            printf("Resetting queue ");
            q->front = q->rear = -1;
        }
    }
    return item;}

// Print the queue
void printQueue(struct queue* q) {
    int i = q->front;
    if (isEmpty(q)) {
        printf("Queue is empty");
    } else {
        printf("\nQueue contains \n");
        for (i = q->front; i < q->rear + 1; i++) {
            printf("%d ", q->items[i]);
        }
    }
}

int main() {
    struct Graph* graph = createGraph(6);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 4);
    addEdge(graph, 1, 3);
    addEdge(graph, 2, 4);
```

```

addEdge(graph, 3, 4);
bfs(graph, 0);
return 0; }

```

Output:

```

Queue contains
0 Resetting queue Visited 0

Queue contains
2 1 Visited 2

Queue contains
1 4 Visited 1

Queue contains
4 3 Visited 4

Queue contains
3 Resetting queue Visited 3

-----
Process exited after 0.04121 seconds with return value 0
Press any key to continue . . .

```

DFS in C**PROGRAM**

```

#include <stdio.h>
#include <stdlib.h>
struct node {
    int vertex;
    struct node* next;
};
struct node* createNode(int v);
struct Graph {
    int numVertices;
    int* visited;
    // We need int** to store a two dimensional array.

```

```
// Similar, we need struct node** to store an array of Linked lists
struct node** adjLists;
};
// DFS algo
void DFS(struct Graph* graph, int vertex) {
    struct node* adjList = graph->adjLists[vertex];
    struct node* temp = adjList;
    graph->visited[vertex] = 1;
    printf("Visited %d \n", vertex);
    while (temp != NULL) {
        int connectedVertex = temp->vertex;
        if (graph->visited[connectedVertex] == 0) {
            DFS(graph, connectedVertex);
        }
        temp = temp->next;
    }
}

// Create a node
struct node* createNode(int v) {
    struct node* newNode = malloc(sizeof(struct node));
    newNode->vertex = v;
    newNode->next = NULL;
    return newNode;
}

// Create graph
struct Graph* createGraph(int vertices) {
    struct Graph* graph = malloc(sizeof(struct Graph));
    graph->numVertices = vertices;
    graph->adjLists = malloc(vertices * sizeof(struct node*));
    graph->visited = malloc(vertices * sizeof(int));
    int i;
```

```
    for (i = 0; i < vertices; i++) {
        graph->adjLists[i] = NULL;
        graph->visited[i] = 0;
    }
    return graph;
}

// Add edge
void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to dest
    struct node* newNode = createNode(dest);
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;
    // Add edge from dest to src
    newNode = createNode(src);
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// Print the graph
void printGraph(struct Graph* graph) {
    int v;
    for (v = 0; v < graph->numVertices; v++) {
        struct node* temp = graph->adjLists[v];
        printf("\n Adjacency list of vertex %d\n ", v);
        while (temp) {
            printf("%d -> ", temp->vertex);
            temp = temp->next;
        }
        printf("\n");
    }
}

int main() {
    struct Graph* graph = createGraph(4);
```

```
addEdge(graph, 0, 1);
addEdge(graph, 0, 2);
addEdge(graph, 1, 2);
addEdge(graph, 2, 3);
printGraph(graph);
DFS(graph, 2);
return 0; }
```

Output:

```
Adjacency list of vertex 0
2 -> 1 ->

Adjacency list of vertex 1
2 -> 0 ->

Adjacency list of vertex 2
3 -> 1 -> 0 ->

Adjacency list of vertex 3
2 ->
Visited 2
Visited 3
Visited 1
Visited 0

-----
Process exited after 0.04728 seconds with return value 0
Press any key to continue . . .
```

EXPERIMENT-10

OBJECTIVE

Implementation of binary search in arrays and on linked Binary Search Tree.

PROGRAM

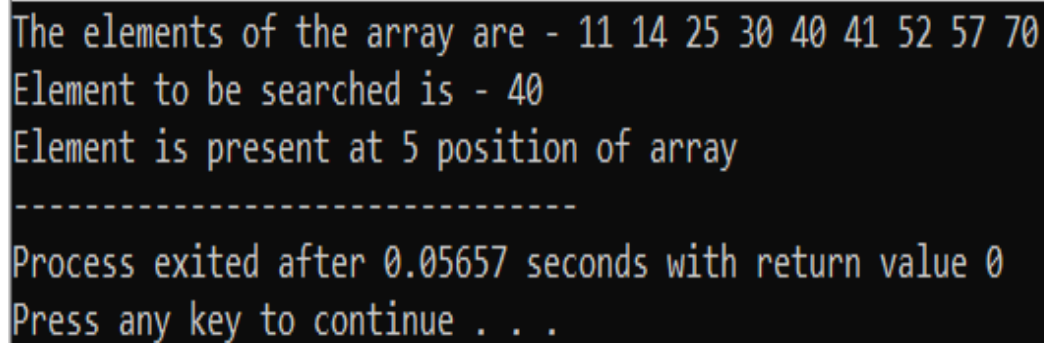
```
#include <stdio.h>

int binarySearch(int a[], int beg, int end, int val)
{
    int mid;
    if(end >= beg)
    {
        mid = (beg + end)/2;
        /* if the item to be searched is present at middle */
        if(a[mid] == val)
        {
            return mid+1;
        }
        /* if the item to be searched is smaller than middle, then it can only be in left subarray */

        else if(a[mid] < val)
        {
            return binarySearch(a, mid+1, end, val);
        }
        /* if the item to be searched is greater than middle, then it can only be in right subarray */
        else
        {
            return binarySearch(a, beg, mid-1, val);
        }
    }
    return -1;
}

int main() {
    int a[] = { 11, 14, 25, 30, 40, 41, 52, 57, 70}; // given array
```

```
int val = 40; // value to be searched
int i;
int n = sizeof(a) / sizeof(a[0]); // size of array
int res = binarySearch(a, 0, n-1, val); // Store result
printf("The elements of the array are - ");
for (i = 0; i < n; i++)
printf("%d ", a[i]);
printf("\nElement to be searched is - %d", val);
if (res == -1)
printf("\nElement is not present in the array");
else
printf("\nElement is present at %d position of array", res);
return 0;
}
```

OUTPUT:A screenshot of a terminal window with a black background and light blue/green text. The output shows the array elements, the search value, the result, and a separator line. It also displays the execution time and a prompt to press a key to continue.

```
The elements of the array are - 11 14 25 30 40 41 52 57 70
Element to be searched is - 40
Element is present at 5 position of array
-----
Process exited after 0.05657 seconds with return value 0
Press any key to continue . . .
```

EXPERIMENT-11**OBJECTIVE**

Implementation of different sorting algorithm like insertion, quick, heap, bubble and many more sorting algorithms.

Bubble sort**PROGRAM**

```
#include<stdio.h>

void print(int a[], int n) //function to print array elements
{
    int i;
    for(i = 0; i < n; i++)
    {
        printf("%d ",a[i]);
    }
}

void bubble(int a[], int n) // function to implement bubble sort
{
    int i, j, temp;
    for(i = 0; i < n; i++)
    {
        for(j = i+1; j < n; j++)
        {
            if(a[j] < a[i])
            {
                temp = a[i];a[i] = a[j]; a[j] = temp;
            }
        }
    }
}

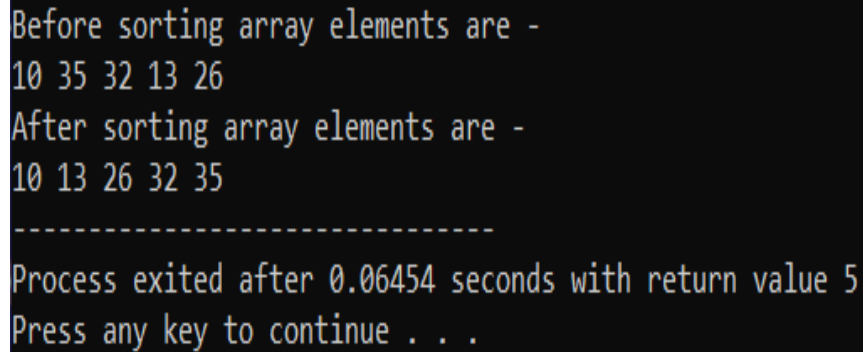
void main ()
{
    int i, j,temp;

    int a[5] = { 10, 35, 32, 13, 26};
    int n = sizeof(a)/sizeof(a[0]);
    printf("Before sorting array
elements are - \n");print(a, n);
    bubble(a, n);
    printf("\nAfter sorting array
```



```
elements are - \n");print(a, n);  
}
```

OUTPUT



```
Before sorting array elements are -  
10 35 32 13 26  
After sorting array elements are -  
10 13 26 32 35  
-----  
Process exited after 0.06454 seconds with return value 5  
Press any key to continue . . .
```

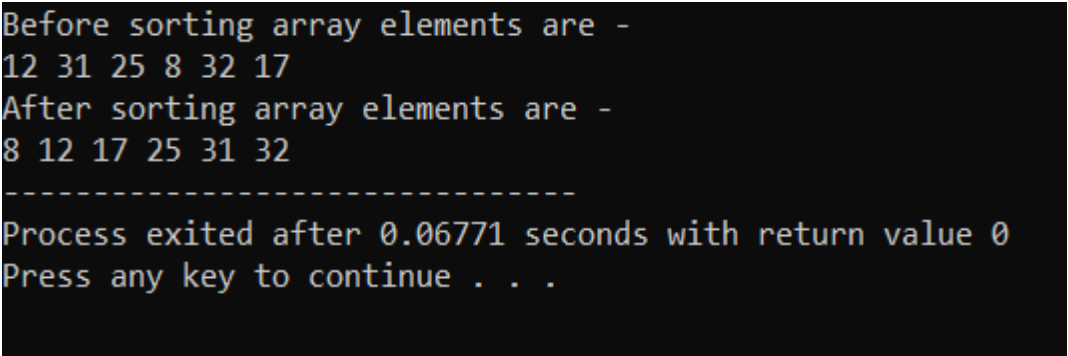
SELECTION SORT

PROGRAM

```
#include <stdio.h>  
void selection(int arr[], int n)  
{  
    int i, j, small;  
    for (i = 0; i < n-1; i++)    // One by one move boundary of unsorted subarray  
    {  
        small = i; //minimum element in unsorted array  
        for (j = i+1; j < n; j++)  
            if (arr[j] < arr[small])  
                small = j;  
        // Swap the minimum element with the first element  
        int temp = arr[small];  
        arr[small] = arr[i]; arr[i] = temp;  
    }  
}  
void printArr(int a[], int n) /* function to print the array */  
{
```

```
int i;
for (i = 0; i < n; i++)printf("%d ", a[i]);
}
int main()
{
    int a[] = { 12, 31, 25, 8, 32, 17 };
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");

    printArr(a, n); selection(a, n);
    printf("\nAfter sorting array elements are - \n");printArr(a, n);
    return 0;
}
```

Output:

```
Before sorting array elements are -
12 31 25 8 32 17
After sorting array elements are -
8 12 17 25 31 32
-----
Process exited after 0.06771 seconds with return value 0
Press any key to continue . . .
```

INSERTION SORT**PROGRAM**

```
#include <stdio.h>

void insert(int a[], int n) /* function to sort an aay with insertion sort */
{
    int i, j, temp;
    for (i = 1; i < n; i++) {
        temp = a[i];
        j = i - 1;
        while(j >= 0 && temp <= a[j]) /* Move the elements greater than temp to one
```

```
position ahead from their current position*/
{
    a[j+1] = a[j]; j = j-1;
}
a[j+1] = temp;
}
}

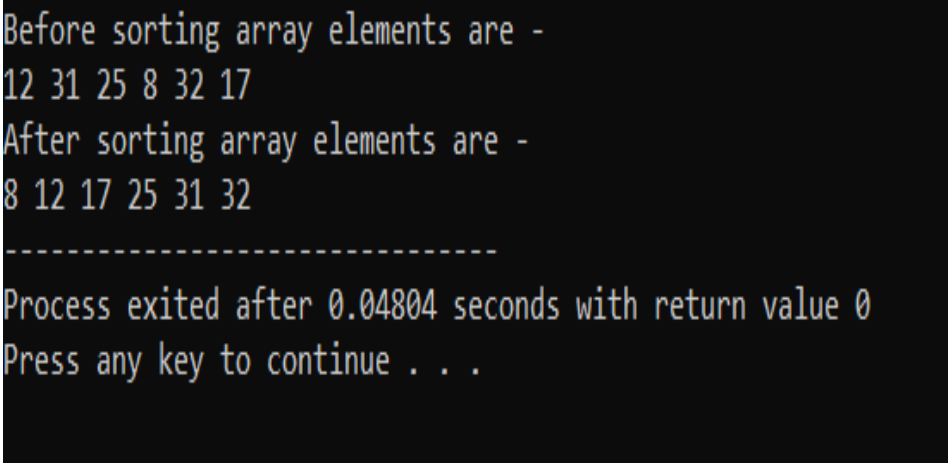
void printArr(int a[], int n) /* function to print the array */
{ int i;
  for (i = 0; i < n; i++)

    printf("%d ", a[i]);
}

int main()
{int a[] = { 12, 31, 25, 8, 32, 17 };
  int n = sizeof(a) / sizeof(a[0]);
  printf("Before sorting array elements are - \n");

  printArr(a, n);
  insert(a, n);
  printf("\nAfter sorting array elements are - \n");

  printArr(a, n);
  return 0;
}
```

Output:

```
Before sorting array elements are -
12 31 25 8 32 17
After sorting array elements are -
8 12 17 25 31 32
-----
Process exited after 0.04804 seconds with return value 0
Press any key to continue . . .
```

QUICK SORT**PROGRAM**

```
#include <stdio.h>

int partition (int a[], int start, int end)
{
    int pivot = a[end]; // pivot element

    int j, i = (start - 1);
    for (j = start; j <= end - 1; j++)
    {
        if (a[j] < pivot)
        {
            i++;
            int t = a[i];

            a[i] = a[j];

            a[j] = t;
        }
    }
    int t = a[i+1];

    a[i+1] = a[end];

    a[end] = t;
    return (i + 1);
}

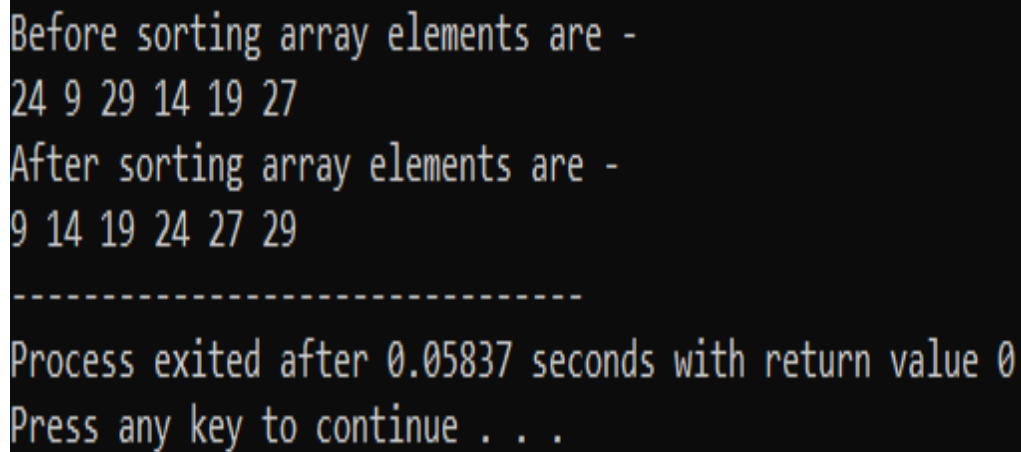
void quick(int a[], int start, int end)
{
    if (start < end)
    {
        int p = partition(a, start, end); //p is the partitioning index

        quick(a, start, p - 1);
        quick(a, p + 1, end);
    }
}

void printArr(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)

        printf("%d ", a[i]);
}
```

```
int main()
{ int a[] = { 24, 9, 29, 14, 19, 27 };
  int n = sizeof(a) / sizeof(a[0]);
  printf("Before sorting array elements are - \n");printArr(a, n);
  quick(a, 0, n - 1);
  printf("\nAfter sorting array elements are - \n");printArr(a, n);
  return 0; }
```

Output:

```
Before sorting array elements are -
24 9 29 14 19 27
After sorting array elements are -
9 14 19 24 27 29
-----
Process exited after 0.05837 seconds with return value 0
Press any key to continue . . .
```

BEYOND THE SYLLABUS EXPERIMENT-1**OBJECTIVE**

WAP to implement Radix Sort.

PROGRAM

```
#include <stdio.h>

int getMax(int a[], int n)

{ int max = a[0];
  int i;
  for(i = 1; i<n; i++)

  { if(a[i] > max)
    max = a[i];
  }

  return max; //maximum element from the array
}

void countingSort(int a[], int n, int place) // function to implement counting sort
{
  int Output[n + 1];
  int count[10] = {0};
  int i;
  // Calculate count of elements
  for (i = 0; i < n; i++)
    count[(a[i] / place) % 10]++;
  // Calculate cumulative frequency
  for (i = 1; i < 10; i++)
    count[i] += count[i - 1];
  // Place the elements in sorted order

  for (i = n - 1; i >= 0; i--) {
    Output[count[(a[i] / place) % 10] - 1] = a[i];
```

```
        count[(a[i] / place) % 10]--;
    }
    for (i = 0; i < n; i++)a[i] = Output[i];
}

// function to implement radix sort

void radixsort(int a[], int n)
{
    int place ;
    // get maximum element from array

    int max = getMax(a, n);

    // Apply counting sort to sort elements based on place value
    for (place = 1; max / place > 0; place *= 10)
        countingSort(a, n, place);
}

// function to print array elements

void printArray(int a[], int n)
{
    int i;
    for (i = 0; i < n; ++i)
    {
        printf("%d ", a[i]);
    } printf("\n");
}

int main() {
    int a[] = { 181, 289, 390, 121, 145, 736, 514, 888, 122};
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");

    printArray(a,n);
    radixsort(a, n);
    printf("After applying Radix sort, the array elements are - \n");
    printArray(a, n);
}
```

Output:

```
Before sorting array elements are -  
181 289 390 121 145 736 514 888 122  
After applying Radix sort, the array elements are -  
121 122 145 181 289 390 514 736 888  
  
-----  
Process exited after 0.06697 seconds with return value 10  
Press any key to continue . . .
```


BEYOND THE SYLLABUS EXPERIMENT-2**OBJECTIVE**

WAP to implement Count Sort.

PROGRAM

```
#include<stdio.h>

int getMax(int a[], int n)

{ int i, max = a[0];
  for(i = 1; i<n; i++)
  {   if(a[i] > max)
      max = a[i];
  }
  return max; //maximum element from the array
}

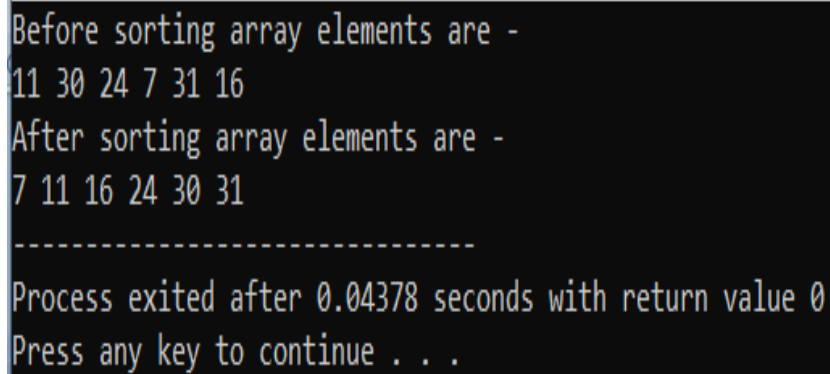
void countSort(int a[], int n) // function to perform counting sort
{ int Output[n+1];
  int max = getMax(a, n);
  int count[max+1]; //create count array with size [max+1]

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

    count[i] = 0; // Initialize count array with all zeros
  for (i = 0; i < n; i++) // Store the count of each element
    count[a[i]]++;
  for(i = 1; i<=max; i++)
    count[i] += count[i-1]; //find cumulative frequency

  for (i = n - 1; i >= 0; i--)
  {
    Output[count[a[i]] - 1] = a[i];
    count[a[i]]--; // decrease count for same numbers
  }
  for(i = 0; i<n; i++)
    a[i] = Output[i]; //store the sorted elements into main array
```

```
}  
void printArr(int a[], int n) /* function to print the array */  
{ int i;  
  for (i = 0; i < n; i++) printf("%d ", a[i]);  
}  
int main()  
{  
  int a[] = { 11, 30, 24, 7, 31, 16 };  
  int n = sizeof(a)/sizeof(a[0]);  
  printf("Before sorting array elements are - \n");  
  
  printArr(a, n);  
  countSort(a, n);  
  printf("\nAfter sorting array elements are - \n");  
  
  printArr(a, n);  
  return 0;  
}
```

Output:

```
Before sorting array elements are -  
11 30 24 7 31 16  
After sorting array elements are -  
7 11 16 24 30 31  
-----  
Process exited after 0.04378 seconds with return value 0  
Press any key to continue . . .
```