Lab Code: 20ECL103
PROGRAMMING USING C LAB
Lab Manual

Department of Electronics & Communication Engineering

Bapatla Engineering College :: Bapatla

(Autonomous)

G.B.C. Road, Mahatmajipuram, Bapatla-522102, Guntur (Dist.)
Andhra Pradesh, India.
E-Mail:bec.principal@becbapatla.ac.in
Web:www.becbapatla.ac.in

Contents

Bapatla Engineering College :: Bapatla
(Autonomous)

Vision

- *To build centers of excellence, impart high quality education and instill high standards of ethics and professionalism through strategic efforts of our dedicated staff, which allows the college to effectively adapt to the ever changing aspects of education.*

- *To empower the faculty and students with the knowledge, skills and innovative thinking to facilitate discovery in numerous existing and yet to be discovered fields of engineering, technology and interdisciplinary endeavors.*

Mission

- *Our Mission is to impart the quality education at par with global standards to the students from all over India and in particular those from the local and rural areas.*

- *We continuously try to maintain high standards so as to make them technologically competent and ethically strong individuals who shall be able to improve the quality of life and economy of our country.*

Bapatla Engineering College :: Bapatla

(Autonomous)

Department of Electronics and Communication Engineering

Vision

To produce globally competitive and socially responsible Electronics and Communication Engineering graduates to cater the ever changing needs of the society.

Mission

- *To provide quality education in the domain of Electronics and Communication Engineering with advanced pedagogical methods.*

- *To provide self learning capabilities to enhance employability and entrepreneurial skills and to inculcate human values and ethics to make learners sensitive towards societal issues.*

- *To excel in the research and development activities related to Electronics and Communication Engineering.*

Bapatla Engineering College :: Bapatla

(Autonomous)

Department of Electronics and Communication Engineering

Program Educational Objectives (PEO's)

PEO-I: *Equip Graduates with a robust foundation in mathematics, science and Engineering Principles, enabling them to excel in research and higher education in Electronics and Communication Engineering and related fields.*

PEO-II: *Impart analytic and thinking skills in students to develop initiatives and innovative ideas for Start-ups, Industry and societal requirements.*

PEO-III: *Instill interpersonal skills, teamwork ability, communication skills, leadership, and a sense of social, ethical, and legal duties in order to promote lifelong learning and Professional growth of the students.*

Program Outcomes (PO's)

Engineering Graduates will be able to:

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 multidisciplinary 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.*

Bapatla Engineering College :: Bapatla

(Autonomous)

Department of Electronics and Communication Engineering

Program Specific Outcomes (PSO's)

PSO1: *Develop and implement modern Electronic Technologies using analytical methods to meet current as well as future industrial and societal needs.*

PSO2: *Analyze and develop VLSI, IoT and Embedded Systems for desired specifications to solve real world complex problems.*

PSO3: *Apply machine learning and deep learning techniques in communication and signal processing.*

PROGRAMMING USING C LAB
I B.Tech – I Semester (Code: 20ECL103)

Lectures	*0*	*Tutorial*	*0*	*Practical*	*3*	*Credits*	*1.5*
Continuous Internal Assessment			*30*	*Semester End Examination (3 Hours)*			*70*

Prerequisites: *None*

Course Objectives: *Students will*

- *Understand basic concepts of C Programming such as: C-tokens, Operators, Input/output, Arithmetic rules.*
- *Develop problem-solving skills to translate "English" described problems into programs written using C language.*
- *Apply pointers for parameter passing, referencing and dereferencing, and linking data structures.*
- *Manipulate variables and types to change the problem state, including numeric, character, array, and pointer types, as well as the use of structures and unions, file handling.*
-

Course Outcomes: After studying this course, the students will be able to

CO1	*Address the challenge, pick and analyze the appropriate data representation formats and algorithms.*
CO2	*Choose the best programming construct for the job at hand by comparing it to other structures and considering their constraints.*
CO3	*Develop the program on a computer, edit, compile, debug, correct, recompile and run it.*
CO4	*Identify tasks in which the numerical techniques learned are applicable and apply them to write programs, and hence use computers effectively to solve the practical problems of engineering and scientific computations.*

Mapping of Course Outcomes with Program Outcomes & Program Specific Outcomes

	PO's												**PSO's**		
CO	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**1**	**2**	**3**
CO1	*3*	*3*			*3*								*2*		
CO2		*2*	*3*		*3*								*2*		
CO3		*2*	*3*		*3*								*2*		
CO4		*3*	*2*		*3*								*2*		
AVG	*3*	*2.5*	*2.67*		*3*								*2*		

LIST OF LAB EXPERIMENTS

1 *Electricity Bill Calculation for Different Categories of Users*

2a *Series Evaluation* $(1 + x^2/2! + x^4/4! + \ldots)$ *up to 10 Terms*

2b *Series Evaluation* $(x + x^3/3! + x7/5! + \ldots)$ *up to 7-digit Accuracy*

3a *Prime Number Check*

3b *Perfect, Abundant, or Deficient Number Check*

4 *Statistical Parameters Calculation Using One-Dimensional Array*

5 *Operations on a List of Numbers (Print, Delete Duplicates, Reverse)*

6 *Binary Search on a List of Numbers*

7 *Matrix Addition and Multiplication*

8 *Menu Driven Program Using Array of Character Pointers*

9 *Sorting and Displaying a List of Student Names*

10a *Recursive Function to Find Factorial*

10b *Solving Tower of Hanoi Using Recursion*

11 *Bookshop Inventory Management using Structures*

12 *Processing Student Records from a Data File*

NOTE: *A minimum of 10 (Ten) experiments have to be Performed and recorded by the candidate to attain eligibility for Semester End Examination.*

Experiment 1: Electricity Bill Calculation for Different Categories

Title: Electricity Bill Calculation Based on User Category and Slabs

Aim:
To write a C program to calculate the electricity bill for different categories of users (Domestic and Commercial), with different slabs in each category.

Apparatus Required:
- *Code::Blocks Software*
- *Computer system with GCC compiler*

Algorithm:
1. *Input the category of the user (Domestic or Commercial).*
2. *Input the number of units consumed.*
3. *Based on the category, apply the corresponding rate slabs:*
   - *Domestic User:*
     - *0 – 200 units: 0.50 per unit*
     - *201 – 400 units: 100 plus 0.65 per unit for units above 200*
     - *401 – 600 units: 230 plus 0.80 per unit for units above 400*
     - *601 and above: 390 plus 1.00 per unit for units above 600*
   - *Commercial User:*
     - *0 – 100 units: 0.50 per unit*
     - *101 – 200 units: 50 plus 0.60 per unit for units above 100*
     - *201 – 300 units: 110 plus 0.70 per unit for units above 200*
     - *301 and above: 230 plus 1.00 per unit for units above 300*
4. *Calculate the bill based on the applicable rates and units consumed.*
5. *Display the total bill.*

C Code:

```
#include <stdio.h>

int main() {
    int units, category;
    double bill = 0;

    // Input category (1 for Domestic, 2 for Commercial)
```

```
    printf("Enter user category (1 for Domestic, 2 for Commercial): ");
    scanf("%d", &category);

    // Input number of units consumed printf("Enter
    the number of units consumed: "); scanf("%d",
    &units);

    if (category == 1) { // Domestic User if
        (units <= 200) {
            bill = units * 0.50;
        } else if (units <= 400) {
            bill = 100 + (units - 200) * 0.65;
        } else if (units <= 600) {
            bill = 230 + (units - 400) * 0.80;
        } else {
            bill = 390 + (units - 600) * 1.00;
        }
    } else if (category == 2) { // Commercial User if
        (units <= 100) {
            bill = units * 0.50;
        } else if (units <= 200) {
            bill = 50 + (units - 100) * 0.60;
        } else if (units <= 300) {
            bill = 110 + (units - 200) * 0.70;
        } else {
            bill = 230 + (units - 300) * 1.00;
        }
    } else {
        printf("Invalid category selected.\n"); return
        1;
    }

    // Display the total bill
    printf("Total electricity bill: Rs. %.2lf\n", bill);

    return 0;
}
```

Expected Output:

For a Domestic User consuming 450 units:

```
Enter user category (1 for Domestic, 2 for Commercial): 1
Enter the number of units consumed: 450
Total electricity bill: Rs. 270.00
```

For a Commercial User consuming 350 units:

```
Enter user category (1 for Domestic, 2 for Commercial): 2
Enter the number of units consumed: 350
Total electricity bill: Rs. 280.00
```

Result:

The C program successfully calculates the electricity bill for both Domestic and Commercial users based on different slabs and categories.

Experiment 2a: Evaluate the Series $1 + x^2/2! + x^4/4! + \ldots$ up to 10 terms

Title: Series Evaluation using Loops (Part a)

Aim:
To write a C program to evaluate the series $1 + x^2/2! + x^4/4! + \ldots$ up to 10 terms using loops.

Apparatus Required:
- *Code::Blocks Software*
- *Computer system with GCC compiler*

Algorithm:
1. *Input a value for `x`.*
2. *Initialize `sum` to 1 and `term` to 1 (representing the first term, 1).*
3. *Use a loop to calculate the next 9 terms of the series.*
4. *In each iteration, calculate the next power of `x` and the factorial of the current even number, then add it to the sum.*
5. *Continue until 10 terms are evaluated.*
6. *Display the result.*

C Code:

```
#include <stdio.h>

int main() {
    int i, n = 10, fact;
    double x, term, sum = 1;

    printf("Enter the value of x: ");
    scanf("%lf", &x);

    for (i = 1; i <= n; i++) {
        term = 1;
        fact = 1;

        for (int j = 1; j <= 2 * i; j++) {
```

```
            term *= x;
            fact *= j;
        }

        sum += term / fact;
    }

    printf("Sum of the series up to 10 terms is: %lf\n", sum);

    return 0;
}
```

Expected Output:
For `x = 2`, the output could be:

```
Enter the value of x: 2
Sum of the series up to 10 terms is: 7.389056
```

Result:
The C program successfully evaluates the series up to 10 terms.

Experiment 2b: Evaluate the Series $x + x^3/3! + x^5/5! + \ldots$ up to 7-digit accuracy

Title: Series Evaluation using Loops (Part b)

Aim:

To write a C program to evaluate the series $x + x^3/3! + x^5/5! + \ldots$ up to 7-digit accuracy using loops.

Apparatus Required:

- *Code::Blocks Software*
- *Computer system with GCC compiler*

Algorithm:

1. *Input a value for `x`.*
2. *Initialize `sum` to `x` and `term` to `x` (representing the first term, `x`).*
3. *Use a loop to calculate the subsequent terms until the difference between two successive terms is less than a tolerance value corresponding to 7-digit accuracy.*
4. *In each iteration, calculate the next power of `x` and the corresponding factorial, then add it to the sum.*
5. *Display the result.*

C Code:

```
#include <stdio.h>

int main() {
    double x, term, sum, accuracy = 0.0000001;
    int i = 3, fact = 1;

    printf("Enter the value of x: ");
    scanf("%lf", &x);

    sum = x;
    term = x;
```

```
    while (term > accuracy) { term
        *= (x * x) / (i * (i - 1)); sum
        += term;
        i += 2;
    }

    printf("Sum of the series up to 7-digit accuracy is: %lf\n", sum);

    return 0;
}
```

Expected Output:
For `x = 2`, the output could be:

```
Enter a number: 2
Sum of the series up to 7-digit accuracy is: 3.626860
```

Result:
The C program successfully evaluates the series up to 7-digit accuracy.

Experiment 3a: Check if a Number is Prime

Title: Prime Number Check

Aim:
To write a C program to check whether a given number is prime or not.

Apparatus Required:
- *Code::Blocks Software*
- *Computer system with GCC compiler*

Algorithm:
1. *Input a number from the user.*
2. *Check if the number is less than or equal to 1. If true, it is not prime.*
3. *Loop through all numbers from 2 to (number/2).*
4. *If the number is divisible by any of these, it is not prime.*
5. *Otherwise, it is a prime number.*

C Code:
```
#include <stdio.h>

int main() {
    int number, i, isPrime = 1;

    printf("Enter a number: ");
    scanf("%d", &number);

    if (number <= 1) {
        isPrime = 0;
    } else {
        for (i = 2; i <= number / 2; i++) {
            if (number % i == 0) {
                isPrime = 0;
                break;
            }
        }
    }
```

```
    if (isPrime) {
        printf("%d is a Prime number.\n", number);
    } else {
        printf("%d is not a Prime number.\n", number);
    }

    return 0;
}
```

Expected Output:
For a number input, say 5, the output should be:

```
Enter a number: 5
5 is a Prime number.
```

For a number input, say 6, the output should be:

```
Enter a number: 6
6 is not a Prime number.
```

Result:
The C program successfully checks if the given number is prime or not.

Experiment 3b: Check if a Number is Perfect, Abundant, or Deficient

Title: Perfect, Abundant, or Deficient Number Check

Aim:
To write a C program to check whether a given number is Perfect, Abundant, or Deficient.

Apparatus Required:
- *Code::Blocks Software*
- *Computer system with GCC compiler*

Algorithm:
1. *Input a number from the user.*
2. *Find the sum of the divisors of the number (excluding the number itself).*
3. *If the sum of divisors equals the number, it is a Perfect number.*
4. *If the sum of divisors is greater than the number, it is Abundant.*
5. *If the sum of divisors is less than the number, it is Deficient.*

C Code:

```
#include <stdio.h>

int main() {
    int number, i, sum = 0;

    printf("Enter a number: ");
    scanf("%d", &number);

    for (i = 1; i < number; i++) {
        if (number % i == 0) {
            sum += i;
        }
    }

    if (sum == number) {
        printf("%d is a Perfect number.\n", number);
    } else if (sum > number) {
        printf("%d is an Abundant number.\n", number);
    } else {
```

```
        printf("%d is a Deficient number.\n", number);
    }

    return 0;
}
```

Expected Output:
For a number input, say 6, the output should be:

```
Enter a number: 6
6 is a Perfect number.
```

For a number input, say 12, the output should be:

```
Enter a number: 12
12 is an Abundant number.
```

Result:
The C program successfully checks whether the given number is Perfect, Abundant, or Deficient.

Experiment 4: Statistical Parameters Calculation Using One-Dimensional Array

Title: Calculation of Mean, Mode, Median, and Variance using a One-Dimensional Array

Aim:
To write a C program to calculate and display statistical parameters (Mean, Mode, Median, and Variance) using a one-dimensional array.

Apparatus Required:
- *Code::Blocks Software*
- *Computer system with GCC compiler*

Algorithm:
1. *Input the number of elements in the array.*
2. *Input the elements of the array.*
3. *Mean: Calculate the sum of the elements and divide by the total number of elements.*
4. *Mode: Find the element that appears most frequently in the array.*
5. *Median: Sort the array and find the middle element(s).*
6. *Variance: Calculate the squared differences between each element and the mean, and then find the average of these differences.*
7. *Display the Mean, Mode, Median, and Variance.*

C Code:

```
#include <stdio.h>

int main() {
    int n, i, j, temp, mode, maxCount = 0, count;
    double sum = 0, mean, median, variance = 0;

    // Input number of elements printf("Enter
    the number of elements: "); scanf("%d",
    &n);

    int arr[n];
```

```
// Input the elements
printf("Enter the elements:\n");
for (i = 0; i < n; i++) {
    scanf("%d", &arr[i]);
    sum += arr[i]; // Sum for mean
}

// Calculate mean
mean = sum / n;

// Sort the array for calculating median and mode for
(i = 0; i < n - 1; i++) {
    for (j = i + 1; j < n; j++) {
        if (arr[i] > arr[j]) {
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
}

// Calculate median
if (n % 2 == 0) {
    median = (arr[n / 2 - 1] + arr[n / 2]) / 2.0;
} else {
    median = arr[n / 2];
}

// Calculate mode
for (i = 0; i < n; i++) {
    count = 1;
    for (j = i + 1; j < n; j++) {
        if (arr[i] == arr[j]) {
            count++;
        }
    }
    if (count > maxCount) {
        maxCount = count;
        mode = arr[i];
```

```
        }
    }

    // Calculate variance
    for (i = 0; i < n; i++) {
        variance += (arr[i] - mean) * (arr[i] - mean);
    }
    variance /= n;

    // Display results
    printf("Mean: %.2lf\n", mean);
    printf("Mode: %d\n", mode);
    printf("Median: %.2lf\n", median);
    printf("Variance: %.2lf\n", variance);

    return 0;
}
```

Expected Output:
For the input array `[1, 2, 3, 2, 4, 5, 2]`:

```
Enter the number of elements: 7
Enter the elements:
1 2 3 2 4 5 2
Mean: 2.71
Mode: 2
Median: 2.00
Variance: 1.63
```

Result:
The C program successfully calculates and displays the statistical parameters (Mean, Mode, Median, and Variance) using a one-dimensional array.

Experiment 5: Operations on a List of Numbers

Title: Performing Operations on a List of Numbers (Print, Delete Duplicates, Reverse)

Aim:

To write a C program to read a list of numbers and perform the following operations:

a) Print the list.

b) Delete duplicates from the list.

c) Reverse the list.

Apparatus Required:

- *Code::Blocks Software*
- *Computer system with GCC compiler*

Algorithm:

1. *Input the number of elements in the list and the list elements.*
2. *Print the original list.*
3. ***Delete Duplicates**:*
   - *Traverse the list.*
   - *For each element, compare it with the subsequent elements.*
   - *Remove any duplicates by shifting the array elements.*
4. ***Reverse the List**:*
   - *Use a loop to swap the first element with the last, the second with the second-last, and so on.*
5. *Print the modified list after removing duplicates and reversing.*

C Code:

```
#include <stdio.h>

int main() {
    int n, i, j, k;

    // Input number of elements printf("Enter
    the number of elements: "); scanf("%d",
    &n);
```

```
int arr[n];

// Input the list of numbers
printf("Enter the elements:\n");
for (i = 0; i < n; i++) {
    scanf("%d", &arr[i]);
}

// a) Print the list
printf("Original List: ");
for (i = 0; i < n; i++) {
    printf("%d ", arr[i]);
}
printf("\n");

// b) Delete duplicates
for (i = 0; i < n - 1; i++) {
    for (j = i + 1; j < n; j++) {
        if (arr[i] == arr[j]) {
            for (k = j; k < n - 1; k++) {
                arr[k] = arr[k + 1];
            }
            n--;
            j--;
        }
    }
}

// Print the list after deleting duplicates
printf("List after deleting duplicates: "); for
(i = 0; i < n; i++) {
    printf("%d ", arr[i]);
}
printf("\n");

// c) Reverse the list
for (i = 0; i < n / 2; i++) {
    int temp = arr[i];
    arr[i] = arr[n - i - 1];
```

```
        arr[n - i - 1] = temp;
    }

    // Print the reversed list
    printf("Reversed List: "); for
    (i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    return 0;
}
```

Expected Output:
For input list `[2, 3, 4, 3, 2, 5]`:

```
Enter the number of elements: 6
Enter the elements:
2 3 4 3 2 5
Original List: 2 3 4 3 2 5
List after deleting duplicates: 2 3 4 5
Reversed List: 5 4 3 2
```

Result:
The C program successfully reads a list of numbers and performs the following operations: printing the list, deleting duplicates, and reversing the list.

Experiment 6: Binary Search on a List of Numbers

Title: Searching for a Number using Binary Search Algorithm

Aim:
To write a C program to read a list of numbers and search for a given number using the Binary Search algorithm. If found, display its index; otherwise, display the message "Element not found in the List."

Apparatus Required:
- *Code::Blocks Software*
- *Computer system with GCC compiler*

Algorithm:
1. *Input the number of elements in the list and the list elements.*
2. *Sort the list in ascending order.*
3. *Input the number to be searched.*
4. *Set two pointers, `low` and `high`, to the start and end of the list.*
5. *Repeat the following until `low` is greater than `high`:*
   - *Calculate the middle index.*
   - *If the middle element equals the number, return the index.*
   - *If the number is smaller than the middle element, adjust `high` to `mid - 1`.*
   - *If the number is greater than the middle element, adjust `low` to `mid + 1`.*
6. *If the number is found, display its index; otherwise, display the message "Element not found in the List."*

C Code:

```
#include <stdio.h>

int main() {
    int n, i, low, high, mid, key, found = 0;

    // Input number of elements printf("Enter
    the number of elements: "); scanf("%d",
    &n);
```

```
int arr[n];

// Input the list of numbers
printf("Enter the elements:\n");
for (i = 0; i < n; i++) {
    scanf("%d", &arr[i]);
}

// Sort the array (using simple Bubble Sort for sorting) for
(i = 0; i < n - 1; i++) {
    for (int j = 0; j < n - i - 1; j++) {
        if (arr[j] > arr[j + 1]) {
            int  temp  =  arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
        }
    }
}

// Input the key to search for printf("Enter
the number to search: "); scanf("%d",
&key);

// Binary Search algorithm
low = 0;
high = n - 1;

while (low <= high) {
    mid = (low + high) / 2;

    if (arr[mid] == key) {
        printf("Element found at index %d.\n", mid); found
        = 1;
        break;
    } else if (arr[mid] < key) {
        low = mid + 1;
    } else {
        high = mid - 1;
    }
}
```

```
    if (!found) {
        printf("Element not found in the List.\n");
    }

    return 0;
}
```

Expected Output:
For the input list `[20, 10, 40, 30, 50]` and the search key `30`:

```
Enter the number of elements: 5
Enter the elements:
20 10 40 30 50
Enter the number to search: 30
Element found at index 2.
```

For the search key `60`:

```
Enter the number of elements: 5
Enter the elements:
20 10 40 30 50
Enter the number to search: 60
Element not found in the List.
```

Result:
The C program successfully searches for a given number using the Binary Search algorithm and displays its index or the message "Element not found in the List."

Experiment 7: Matrix Addition and Multiplication

Title: Matrix Addition and

Multiplication Aim:
To write a C program to read two matrices and compute their sum and product.

Apparatus Required:
- *Code::Blocks Software*
- *Computer system with GCC compiler*

Algorithm:
1. *Input the number of rows and columns for the matrices.*
2. *Input the elements of the first matrix.*
3. *Input the elements of the second matrix.*
4. *Matrix Addition:*
   - *Add corresponding elements of both matrices and store the result in the sum matrix.*
5. *Matrix Multiplication:*
   - *Multiply the elements of the rows of the first matrix by the corresponding elements of the columns of the second matrix and sum them, storing the result in the product matrix.*
6. *Display the sum and product matrices.*

C Code:

```
#include <stdio.h>

int main() {
    int rows, cols, i, j, k;

    // Input the number of rows and columns
    printf("Enter the number of rows and columns: ");
    scanf("%d %d", &rows, &cols);
```

```
    int matrix1[rows][cols], matrix2[rows][cols], sum[rows][cols], product[rows]
[cols];

    // Input the elements of the first matrix printf("Enter
    the elements of the first matrix:\n"); for (i = 0; i <
    rows; i++) {
        for (j = 0; j < cols; j++) {
            scanf("%d", &matrix1[i][j]);
        }
    }

    // Input the elements of the second matrix printf("Enter
    the elements of the second matrix:\n"); for (i = 0; i <
    rows; i++) {
        for (j = 0; j < cols; j++) {
            scanf("%d", &matrix2[i][j]);
        }
    }

    // Compute the sum of the two matrices
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            sum[i][j] = matrix1[i][j] + matrix2[i][j];
        }
    }

    // Display the sum matrix
    printf("Sum of the matrices:\n"); for
    (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            printf("%d ", sum[i][j]);
        }
        printf("\n");
    }

    // Initialize the product matrix to  0
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            product[i][j] = 0;
        }
```

```
    }

    // Compute the product of the two matrices
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            for (k = 0; k < cols; k++) {
                product[i][j] += matrix1[i][k] * matrix2[k][j];
            }
        }
    }

    // Display the product matrix
    printf("Product of the matrices:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            printf("%d ", product[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Expected Output:
For two matrices:

Matrix 1:
```
1 2
3 4
```

Matrix 2:
```
5 6
7 8
```

```
```

Enter the number of rows and columns: 2 2
Enter the elements of the first matrix:
1 2
3 4
Enter the elements of the second matrix:
5 6
7 8
Sum of the matrices:
6 8
10 12
Product of the matrices:
19 22
43 50
```

**Result:**
*The C program successfully computes the sum and product of two matrices.*
```

Experiment 8: Menu Driven Program Using Array of Character Pointers

Title: Menu Driven Program to Insert, Delete, and Print Student Names using Array of Character Pointers

Aim:

To write a menu-driven C program using an array of character pointers to:
a) Insert a student name.
b) Delete a student name.
c) Print the names of students.

Apparatus Required:

- *Code::Blocks Software*
- *Computer system with GCC compiler*

Algorithm:

1. *Initialize an array of character pointers for storing student names.*
2. *Display the menu with options to:*
   - *Insert a student name.*
   - *Delete a student name.*
   - *Print the names of students.*
3. *Use a switch-case structure to handle each option.*
4. *For inserting a student name, allocate memory and store the name.*
5. *For deleting a student name, find the name in the array and remove it by shifting the remaining names.*
6. *For printing student names, display all the names stored in the array.*
7. *Repeat the menu until the user chooses to exit.*

C Code:

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

#define MAX 100

int main() {
```

```
char *students[MAX]; // Array of character pointers for student names int
count = 0, choice, i;
char name[50];

while (1) {
    // Display menu
    printf("\nMenu:\n");
    printf("1. Insert a student name\n");
    printf("2. Delete a student name\n");
    printf("3. Print the names of students\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1: // Insert a student name if
            (count < MAX) {
                printf("Enter student name: ");
                scanf("%s", name);
                students[count] = (char *)malloc(strlen(name) + 1);
                strcpy(students[count], name);
                count++;
            } else {
                printf("List is full.\n");
            }
            break;

        case 2: // Delete a student name
            printf("Enter student name to delete: ");
            scanf("%s", name);
            for (i = 0; i < count; i++) {
                if (strcmp(students[i], name) == 0) {
                    free(students[i]);
                    for (int j = i; j < count - 1; j++) {
                        students[j] = students[j + 1];
                    }
                    count--;
                    printf("Student name deleted.\n");
                    break;
                }
```

```
            }
            if (i == count) {
                printf("Name not found.\n");
            }
            break;

        case 3: // Print student names
            printf("List of students:\n");
            for (i = 0; i < count; i++) {
                printf("%s\n", students[i]);
            }
            break;

        case 4: // Exit
            for (i = 0; i < count; i++) {
                free(students[i]); // Free allocated memory before exiting
            }
            return 0;

        default:
            printf("Invalid choice.\n");
    }
  }

  return 0;
}
```

Expected Output:

```
Menu:
1. Insert a student name
2. Delete a student name
3. Print the names of students
4. Exit
Enter your choice: 1 Enter
student name: Alice

Menu:
1. Insert a student name
```

2. Delete a student name
3. Print the names of students
4. Exit
Enter your choice: 1
Enter student name: Bob

Menu:
1. Insert a student name
2. Delete a student name
3. Print the names of students
4. Exit
Enter your choice: 3
List of students:
Alice
Bob

Menu:
1. Insert a student name
2. Delete a student name
3. Print the names of students
4. Exit
Enter your choice: 2
Enter student name to delete: Bob
Student name deleted.

Menu:
1. Insert a student name
2. Delete a student name
3. Print the names of students
4. Exit
Enter your choice: 3
List of students:
Alice
```

**Result:**
*The C program successfully implements a menu-driven system to insert, delete, and print student names using an array of character pointers.*
```

Experiment 9: Operations on a List of Student Names

Title: Sorting and Displaying a List of Student Names

Aim:

To write a C program to read a list of student names and perform the following operations:

a) Print the list of names.

b) Sort them in ascending order.

c) Print the list after sorting.

Apparatus Required:

- *Code::Blocks Software*
- *Computer system with GCC compiler*

Algorithm:

1. *Input the number of students.*
2. *Input the names of the students into an array of character pointers.*
3. *Print the List:*
   - *Display the names as they were input.*
4. *Sort in Ascending Order:*
   - *Use the Bubble Sort algorithm to sort the names alphabetically.*
5. *Print the Sorted List:*
   - *Display the names in ascending order after sorting.*

C Code:

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

#define MAX 100

int main() {
    int n, i, j;
    char temp[50];
    char students[MAX][50]; // Array of strings to store student names

    // Input the number of students
```

```
    printf("Enter the number of students: ");
    scanf("%d", &n);

    // Input the student names
    printf("Enter the names of the students:\n");
    for (i = 0; i < n; i++) {
        scanf("%s", students[i]);
    }

    // a) Print the list of names
    printf("List of student names:\n"); for
    (i = 0; i < n; i++) {
        printf("%s\n", students[i]);
    }

    // b) Sort the names in ascending order
    for (i = 0; i < n - 1; i++) {
        for (j = i + 1; j < n; j++) {
            if (strcmp(students[i], students[j]) > 0) {
                strcpy(temp, students[i]);
                strcpy(students[i], students[j]);
                strcpy(students[j], temp);
            }
        }
    }

    // c) Print the sorted list of names
    printf("List of student names after sorting:\n");
    for (i = 0; i < n; i++) {
        printf("%s\n", students[i]);
    }

    return 0;
}
```

Expected Output:
For the input list of student names `["Alice", "Bob", "Charlie", "David"]`:

```
*Enter the number of students: 4*
*Enter the names of the students:*
*Alice*
*Bob*
*Charlie*
*David*
*List of student names:*
*Alice*
*Bob*
*Charlie*
*David*
*List of student names after sorting:*
*Alice*
*Bob*
*Charlie*
*David*
```

For an unsorted list of names `["Charlie", "Alice", "David", "Bob"]`:

```
*Enter the number of students: 4*
*Enter the names of the students:*
*Charlie*
*Alice*
*David*
*Bob*
*List of student names:*
*Charlie*
*Alice*
*David*
*Bob*
*List of student names after sorting:*
*Alice*
*Bob*
*Charlie*
```

David
```

**Result:**
*The C program successfully reads the list of student names, prints the original list, sorts the names in ascending order, and prints the sorted list.*
```

Experiment 10a: Recursive Function to Find Factorial

Title: Finding Factorial Using Recursion

Aim:
To write a C program to find the factorial of a given number using a recursive function.

Apparatus Required:
- *Code::Blocks Software*
- *Computer system with GCC compiler*

Algorithm:
1. *Input a number from the user.*
2. *Define a recursive function `factorial()` that:*
   - *Returns 1 if the number is 0 or 1.*
   - *Otherwise, returns `n * factorial(n-1)`.*
3. *Call the recursive function and display the result.*

C Code:

```
#include <stdio.h>

// Recursive function to find factorial
int factorial(int n) {
    if (n == 0 || n == 1) {
        return 1;
    } else {
        return n * factorial(n - 1);
    }
}

int main() {
    int number;

    printf("Enter a number: ");
    scanf("%d", &number);
```

```
    printf("Factorial of %d is %d\n", number, factorial(number));

    return 0;
}
```

Expected Output:

```
Enter a number: 5
Factorial of 5 is 120
```

Result:

The C program successfully finds the factorial of a given number using a recursive function.

Experiment 10b: Recursive Function for Tower of Hanoi

Title: Solving Tower of Hanoi Using Recursion

Aim:
To write a C program to solve the Tower of Hanoi problem with three towers (A, B, and C) and three disks using a recursive function.

Apparatus Required:
- *Code::Blocks Software*
- *Computer system with GCC compiler*

Algorithm:
1. *Define a recursive function `towerOfHanoi()` that:*
   - *Takes the number of disks and the source, auxiliary, and destination towers as parameters.*
   - *Moves `n-1` disks from the source to the auxiliary tower using the destination as an intermediate.*
   - *Moves the remaining disk from the source to the destination tower.*
   - *Moves `n-1` disks from the auxiliary to the destination using the source as an intermediate.*
2. *Call the recursive function and display the steps for moving the disks.*

C Code:

```
#include <stdio.h>

// Recursive function for Tower of Hanoi
void towerOfHanoi(int n, char source, char auxiliary, char destination) { if
    (n == 1) {
        printf("Move disk 1 from %c to %c\n", source, destination);
        return;
    }
    towerOfHanoi(n - 1, source, destination, auxiliary); printf("Move
    disk %d from %c to %c\n", n, source, destination);
    towerOfHanoi(n - 1, auxiliary, source, destination);
}
```

```
int main() {
    int number;

    printf("Enter the number of disks: ");
    scanf("%d", &number);

    towerOfHanoi(number, 'A', 'B', 'C');

    return 0;
}
```

Expected Output:
For 3 disks:
```
Enter the number of disks: 3
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

Result:
The C program successfully solves the Tower of Hanoi problem using a recursive function.

Experiment 11: Bookshop Inventory Management using Structures

Title: Bookshop Inventory System using Structures

Aim:
To write a C program that maintains the inventory of books being sold in a bookshop using structures. The system allows searching for books, checking availability, and calculating the total cost if the required number of copies is available.

Apparatus Required:
- *Code::Blocks Software*
- *Computer system with GCC compiler*

Algorithm:
1. Define a structure `Book` to store details such as author, title, price, publisher, and stock position.
2. Input the details of multiple books into an array of structures.
3. Input the title and author of the book a customer is looking for.
4. Search the array for the book based on the title and author.
5. If the book is not found, display an appropriate message.
6. If the book is found, display its details and request the number of copies required.
7. If the requested number of copies is available, display the total cost. If not, display the message "required copies not in stock."

C Code:

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

// Structure to represent a book
struct Book {
    char title[50];
    char author[50];
    char publisher[50];
```

```
    float price;
    int stock;
};

int main() {
    int n, i, copies, found = 0;
    char searchTitle[50], searchAuthor[50];

    // Input number of books printf("Enter
    the number of books: "); scanf("%d",
    &n);

    struct Book books[n];

    // Input details of each book
    for (i = 0; i < n; i++) {
        printf("\nEnter details of book %d:\n", i + 1);
        printf("Title: ");
        scanf("%s", books[i].title);
        printf("Author: ");
        scanf("%s", books[i].author);
        printf("Publisher: ");
        scanf("%s", books[i].publisher);
        printf("Price: ");
        scanf("%f", &books[i].price);
        printf("Stock position: ");
        scanf("%d", &books[i].stock);
    }

    // Input the title and author of the book to search for
    printf("\nEnter the title of the book you want: ");
    scanf("%s", searchTitle);
    printf("Enter the author of the book: ");
    scanf("%s", searchAuthor);

    // Search for the book in the inventory
    for (i = 0; i < n; i++) {
        if (strcmp(books[i].title, searchTitle) == 0 && strcmp(books[i].author,
searchAuthor) == 0) {
            found = 1;
```

```
            printf("\nBook found!\n");
            printf("Title: %s\n", books[i].title);
            printf("Author: %s\n", books[i].author);
            printf("Publisher: %s\n", books[i].publisher);
            printf("Price: %.2f\n", books[i].price); printf("Stock:
            %d\n", books[i].stock);

            // Request number of copies
            printf("Enter the number of copies required: ");
            scanf("%d", &copies);

            // Check if enough stock is available if
            (copies <= books[i].stock) {
                printf("Total cost: %.2f\n", copies * books[i].price);
            } else {
                printf("Required copies not in stock.\n");
            }
            break;
        }
    }

    if (!found) {
        printf("Book not found in the inventory.\n");
    }

    return 0;
}
```

Expected Output:

```
Enter the number of books: 2

Enter details of book 1:
Title: CProgramming
Author: Dennis
Publisher: Pearson
Price: 450.50
Stock position: 10
```

Enter details of book 2:
Title: Algorithms
Author: Cormen
Publisher: MITPress
Price: 500.00
Stock position: 5

Enter the title of the book you want: CProgramming
Enter the author of the book: Dennis

Book found!
Title: CProgramming
Author: Dennis
Publisher: Pearson
Price: 450.50
Stock: 10
Enter the number of copies required: 3
Total cost: 1351.50
```

*If the requested number of copies exceeds the available stock:*
```
Enter the number of copies required: 12 Required
copies not in stock.
```

*If the book is not found in the inventory:*
```
Book not found in the inventory.
```

**Result:**
*The C program successfully manages the inventory of a bookshop, allowing the search for books, checking availability, and calculating the total cost if the required number of copies is available.*
```

Experiment 12: Processing Student Records from a Data File

Title: Reading and Writing Successful Students' Data Based on Percentage

Aim:
To write a C program to read a data file of students' records (Regno, Name, M1, M2, M3, M4, M5) and write the successful students' data (percentage > 40%) to a new data file.

Apparatus Required:
- *Code::Blocks Software*
- *Computer system with GCC compiler*

Algorithm:
1. *Define a structure `Student` to store each student's details: Registration number, Name, and Marks in 5 subjects (M1, M2, M3, M4, M5).*
2. *Open the input file in read mode and the output file in write mode.*
3. *Read the student records from the input file.*
4. *Calculate the percentage for each student.*
5. *If the student's percentage is greater than 40%, write their record to the output file.*
6. *Close both the input and output files.*

C Code:

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

struct Student {
    int regno;
    char name[50];
    int marks[5]; // Marks in 5 subjects
    float percentage;
};

int main() {
    FILE *inFile, *outFile;
    struct Student student;
```

```
    int totalMarks;

    // Open input and output files
    inFile = fopen("students.txt", "r");
    outFile = fopen("successful_students.txt", "w");

    if (inFile == NULL || outFile == NULL) {
        printf("Error opening file.\n"); exit(1);
    }

    // Read student records from the input file
    while (fscanf(inFile, "%d %s %d %d %d %d %d", &student.regno,
student.name, &student.marks[0], &student.marks[1], &student.marks[2],
&student.marks[3], &student.marks[4]) != EOF) {
        // Calculate the total marks and percentage
        totalMarks = student.marks[0] + student.marks[1] + student.marks[2] +
student.marks[3] + student.marks[4];
        student.percentage = (totalMarks / 5.0);

        // Write to the output file if percentage > 40% if
        (student.percentage > 40.0) {
            fprintf(outFile, "%d %s %.2f\n", student.regno, student.name,
student.percentage);
        }
    }

    // Close the files
    fclose(inFile);
    fclose(outFile);

    printf("Successful students' data written to 'successful_students.txt'.\n");

    return 0;
}
```

Expected Input (in `students.txt`):

```
101 John 45 55 65 60 70
```

102 Jane 30 35 40 38 37
103 Mark 60 70 75 80 85
104 Luke 50 55 58 60 62
```

**Expected Output (in `successful_students.txt`):**
```
101 John 59.00
103 Mark 74.00
104 Luke 57.00
```

**Result:**
*The C program successfully reads student records from the data file and writes the successful students' data (percentage > 40%) to another file.*
```

Appendix:

Sample viva questions and answers

Experiment 1: Electricity Bill Calculation for Different Categories of Users

1. Q: Why is it necessary to handle different slabs for Domestic and Commercial users separately?
- A: To ensure that each category's pricing structure is correctly applied based on their unique consumption slabs.

2. Q: How are nested `if-else` statements used in this experiment?
- A: They allow for applying different slabs based on the units consumed, ensuring that the correct billing logic is applied.

3. Q: What happens if an invalid user category is entered? How can this be prevented?
- A: The program will print an error message. Input validation can be used to prevent invalid entries.

4. Q: How would you modify the program if new categories or slabs are introduced?
- A: New conditional blocks (if-else or switch-case) could be added for the new categories and slabs.

5. Compulsory Question: What is the practical utility of this electricity bill calculation program?
- A: This program simulates a real-world electricity billing system used by power companies to calculate bills based on consumption and user category.

Experiment 2a: Series Evaluation ($1 + x^2/2! + x^4/4! + \ldots$) up to 10 Terms

1. Q: How is factorial used in this experiment?
- A: Factorials are used in the denominator of each term to divide progressively increasing powers of x.

2. Q: How would the accuracy of the series be affected if fewer terms were used?
- A: Using fewer terms would reduce the accuracy of the series approximation.

3. Q: Why is it preferred to use a loop for this calculation rather than recursion?
- A: A loop is more efficient and straightforward when calculating a fixed number of terms.

4. Q: How does the value of x affect the series evaluation?
- A: Larger values of x can cause the terms to increase rapidly, potentially reducing the accuracy.

5. Compulsory Question: How is the series evaluation practically useful in real-world applications?
- A: Series evaluations are widely used in numerical methods, solving complex functions where direct computation is not possible, especially in engineering and physics.

Experiment 2b: Series Evaluation ($x + x^3/3! + x^5/5! + \ldots$) up to 7-digit Accuracy

1. Q: How is the accuracy controlled in this program?
- A: Accuracy is controlled by evaluating the series until the change in terms becomes negligible (up to 7 digits).

2. Q: How are higher powers of x handled in the series?
- A: Each term in the series is divided by increasing factorial values, reducing its contribution.

3. Q: What is the significance of using factorial in the denominator of each term in the series?
- A: Factorials ensure that higher-order terms decrease rapidly, helping in the convergence of the series.

4. Q: Why might the precision of floating-point numbers affect the result?

- A: Floating-point numbers have limited precision, which can lead to small errors in calculations when dealing with very large or very small values.

5. Compulsory Question: Why is achieving 7-digit accuracy important in practical applications?
- A: High accuracy is essential in scientific and engineering computations where even small deviations can lead to incorrect results.

Experiment 3a: Prime Number Check

1. Q: How do you check if a number is prime?
- A: By dividing it by numbers from 2 to the square root of the number. If no divisor yields a zero remainder, the number is prime.

2. Q: What is the time complexity of the prime number check used here?
- A: The time complexity is $O(\sqrt{n})$ since we only check up to the square root of the number.

3. Q: How can the program be optimized for large numbers?
- A: Using algorithms like the Sieve of Eratosthenes or Miller-Rabin can significantly speed up prime checking for large numbers.

4. Q: Why is 2 considered the smallest prime number?
- A: 2 is the smallest prime because it is divisible only by 1 and itself.

5. Compulsory Question: What is the real-world utility of checking whether a number is prime?
- A: Prime numbers are crucial in cryptography and network security algorithms, where they are used for encryption and securing data.

Experiment 3b: Perfect, Abundant, or Deficient Number Check

1. Q: What is the mathematical definition of a perfect number?
- A: A perfect number is one whose sum of divisors (excluding itself) is equal to the number itself.

2. Q: How can you classify a number as abundant or deficient?
- A: A number is abundant if the sum of its divisors is greater than the number and deficient if it is less than the number.

3. Q: Why are perfect numbers rare?
- A: Perfect numbers follow specific mathematical patterns and are relatively sparse among integers.

4. Q: How would the program change to classify numbers in a large range more efficiently?
- A: Precomputing divisor sums for multiple numbers using a sieve-like approach could make the program more efficient.

5. Compulsory Question: How are perfect, abundant, or deficient numbers used in practical scenarios?
- A: These classifications have applications in number theory, cryptography, and mathematical puzzles.

Experiment 4: Statistical Parameters Calculation Using One-Dimensional Array

1. Q: What is the difference between mean and median?
- A: The mean is the average of the values, while the median is the middle value when the data is sorted.

2. Q: How is variance calculated?
- A: Variance is the average of the squared differences between each data point and the mean.

3. Q: What does the mode represent in a dataset?
- A: The mode is the value that appears most frequently in a dataset.

4. Q: How is median different when dealing with an even vs. odd number of elements?
- A: For odd numbers, it is the middle element; for even numbers, it is the average of the two middle elements after sorting.

5. Compulsory Question: What is the importance of calculating mean, median, mode, and variance in real life?
- A: These statistical measures are essential for summarizing data, understanding trends, and making informed decisions in fields such as finance, research, and quality control.

Experiment 5: Operations on a List of Numbers (Print, Delete Duplicates, Reverse)

1. Q: How are duplicates removed from the list?
- A: Duplicates are removed by comparing each element with subsequent elements and shifting the remaining elements if a duplicate is found.

2. Q: How is the list reversed in this program?
- A: By swapping the first element with the last, the second with the second-last, and so on until the middle of the list is reached.

3. Q: What is the time complexity of deleting duplicates from the list?
- A: The time complexity is $O(n^2)$ because each element is compared with every other element.

4. Q: How would you improve the efficiency of the duplicate removal process?
- A: Using a hash table to store and check for duplicates would reduce the time complexity to O(n).

5. Compulsory Question: What are the practical applications of deleting duplicates and reversing lists in programming?
- A: These operations are widely used in data processing, database management, and ensuring data integrity in software systems.

Experiment 6: Binary Search on a List of Numbers

1. Q: What is the time complexity of binary search?
- A: The time complexity of binary search is O(log n) because the search space is halved with each iteration.

2. Q: Why must the list be sorted before applying binary search?
- A: Binary search relies on the list being sorted to eliminate half of the search space at each step.

3. Q: How would the algorithm behave if the list were unsorted?
- A: Binary search would not work correctly on an unsorted list and might give incorrect results.

4. Q: What is the advantage of using binary search over linear search?
- A: Binary search is much faster for large datasets because it reduces the number of comparisons needed.

5. Compulsory Question: Why is binary search more efficient compared to linear search in practical applications?
- A: Binary search is more efficient because it reduces the search space exponentially, making it ideal for searching large datasets like databases and search engines.

Experiment 7: Matrix Addition and Multiplication

1. Q: What is the condition for two matrices to be added?
- A: Two matrices can be added only if they have the same dimensions (i.e., the same number of rows and columns).

2. Q: How is matrix multiplication performed?
- A: The element in the resulting matrix is the dot product of the corresponding row from the first matrix and the column from the second matrix.

3. Q: What are the dimensions of the resulting matrix after multiplication?
- A: The resulting matrix has the same number of rows as the first matrix and the same number of columns as the second matrix.

4. Q: Can you multiply two matrices where the number of columns of the first matrix does not match the number of rows of the second matrix?
- A: No, matrix multiplication is not defined if the number of columns in the first matrix does not match the number of rows in the second matrix.

5. Compulsory Question: Where are matrix operations like addition and multiplication used in the real world?

- A: Matrix operations are fundamental in fields such as computer graphics, 3D modeling, machine learning, and physics simulations.

Experiment 8: Menu Driven Program Using Array of Character Pointers

1. Q: How are character pointers used in this experiment?
- A: Character pointers are used to dynamically store and manipulate a list of student names.

2. Q: Why is dynamic memory allocation important in this experiment?
- A: Dynamic memory allocation allows flexible memory management, enabling the program to allocate memory only when needed, preventing wastage of memory.

3. Q: What are the advantages of using a menu-driven program?
- A: Menu-driven programs make it easier for users to interact with the program by providing clear options and functionality in a structured way.

4. Q: How does the program handle inserting and deleting names from the list?
- A: The program uses dynamic memory allocation for inserting names and shifts the remaining elements when deleting a name.

5. Compulsory Question: How does a menu-driven program benefit real-world applications?
- A: Menu-driven programs provide a user-friendly interface for various tasks, commonly seen in software applications like database management systems and inventory systems.

Experiment 9: Sorting and Displaying a List of Student Names

1. Q: Which sorting algorithm is used in this experiment?
- A: The Bubble Sort algorithm is used.

2. Q: How does the Bubble Sort algorithm work?

- A: Bubble Sort works by repeatedly stepping through the list, comparing adjacent elements, and swapping them if they are in the wrong order. This process is repeated until the list is sorted.

3. Q: What is the time complexity of Bubble Sort?
- A: The time complexity of Bubble Sort is $O(n^2)$, where n is the number of elements in the list.

4. Q: What happens if the list is already sorted?
- A: If the list is already sorted, Bubble Sort still compares the elements, but no swaps will occur. An optimized version can detect this and stop early.

5. Compulsory Question: Why is sorting data important in practical applications?
- A: Sorting is crucial for efficient data retrieval, searching, and organization, widely used in databases, e-commerce websites, and operating systems.

Experiment 10a: Recursive Function to Find Factorial

1. Q: How is recursion different from iteration?
- A: In recursion, a function calls itself until a base condition is met, while in iteration, a set of instructions is repeated using loops.

2. Q: What is the base case in the recursive function for factorial?
- A: The base case is when the number is 0 or 1, in which case the factorial is 1.

3. Q: What are the risks of using recursion?
- A: Recursion can lead to excessive memory usage and stack overflow if the recursion depth is too large.

4. Q: Can you calculate factorials of large numbers using recursion?
- A: Yes, but recursion can lead to stack overflow for very large numbers. Iteration or dynamic programming may be better in such cases.

5. Compulsory Question: What are the practical applications of recursion in programming?

- A: Recursion is widely used in algorithms like tree traversal, searching, sorting, and solving combinatorial problems such as finding factorials and generating Fibonacci sequences.

Experiment 10b: Solving Tower of Hanoi Using Recursion

1. Q: What is the Tower of Hanoi problem?
- A: It is a mathematical puzzle where disks must be moved from one peg to another, using an auxiliary peg, following specific rules.

2. Q: What is the recursive approach to solving the Tower of Hanoi problem?
- A: Move the top `n-1` disks from the source peg to the auxiliary peg, then move the nth disk to the destination peg, and finally move the `n-1` disks from the auxiliary peg to the destination peg.

3. Q: How many moves are required to solve the Tower of Hanoi for `n` disks?
- A: The minimum number of moves required is `2^n - 1`, where `n` is the number of disks.

4. Q: Can the Tower of Hanoi problem be solved iteratively?
- A: While the problem is typically solved recursively, it is possible to solve it iteratively, but the recursive solution is more intuitive.

5. Compulsory Question: How does the Tower of Hanoi problem apply to real-world scenarios?
- A: The Tower of Hanoi problem illustrates recursive problem-solving techniques used in algorithm design, artificial intelligence, and multi-step computational processes.

Experiment 11: Bookshop Inventory Management using Structures

1. Q: What is the purpose of using structures in this program?
- A: Structures are used to group different attributes (title, author, price, stock) of a book under one name, allowing easy access and management of related data.

2. Q: How does the program check if a book is available?
- A: The program searches for the book by title and author in the array of structures and checks the stock value to see if it is available.

3. Q: How are multiple attributes of a book managed using structures?
- A: All the attributes (title, author, price, stock) are encapsulated within a structure, making it easy to manage multiple books using an array of structures.

4. Q: What is the advantage of using structures over individual variables for each book attribute?
- A: Structures allow for better organization and scalability, especially when dealing with large datasets where each item has multiple attributes.

5. Compulsory Question: How does the concept of structures in C help in managing bookshop inventory?
- A: Structures allow efficient storage and manipulation of related data (like book details) in a program, essential for real-world inventory management systems.

Experiment 12: Processing Student Records from a Data File

1. Q: How are files used to store and retrieve student records in this program?
- A: Files are used to store the records persistently, and functions like `fscanf()` and `fprintf()` are used to read from and write to these files.

2. Q: What are the benefits of processing student records using files rather than using arrays in memory?
- A: Files allow for storing large amounts of data that would not fit in memory, and they provide persistence, meaning the data remains available even after the program terminates.

3. Q: How is the percentage calculated for each student?
- A: The percentage is calculated by summing the marks in all subjects and dividing by the total possible marks.

4. Q: What happens if a student's percentage is below 40%?

- A: If a student's percentage is below 40%, their record is not written to the output file for successful students.

5. Compulsory Question: What is the practical utility of processing student records using files in C programming?

- A: File handling is essential for managing large datasets in real-world applications, such as student records in educational systems, where persistent data storage and efficient processing are required.