

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

Lab Manual

Course: Programming in C

SEM-II

**A.Y. 2023-2024**

Vision

To evolve as a centre of excellence in Computer Science and Engineering to produce skilled and proficient global professionals to build the society.

Mission

* To provide the conducive environment for establishing students in the global platform of research and innovation.
* To educate students on cutting edge technologies with problem solving capabilities, leadership and teamwork skills.
* To inculcate the professional values with lifelong learning through curricular and co-curricular activities and create globally-aware disciplined citizens.
* To commence various initiatives for motivating students to work for the betterment of society.

Program Outcomes

# Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **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.
3. **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.
4. **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.
5. **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.
6. **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.
7. **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.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **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.
11. **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.
12. **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.

Course Outcome

1. Understand the programming in IDE (Integrated Development Environment) and write, execute and debug simple programs.
2. Interpret the programming tasks logically and understand making the pseudo-code and flowchart.
3. Design and implement basic programming solutions including statements, macros, control structures and methods.
4. Understand and apply the concept of Array and Strings to solve problem statement.
5. Understand and apply the concepts of structures and unions: declaration, initialization and implementation.
6. Understand the concepts of Function modules, its usage and memory allocation using Pointers.

List of Experiments

|  |  |  |
| --- | --- | --- |
| **Sr.**  **No.** | **Title of Experiment** | **CO** |
| 1 | Design and develop a flowchart or an algorithm that takes three coefficients a,b and c of a quadratic equation (ax2+bx+c=0) as input and compute all possible roots. | CO1 |
| 2 | Write a C program for the developed flowchart/algorithm and execute the same to output the possible roots for a given set of coefficients with appropriate messages. | CO1 |
| 3 | Write a C Program to relate two integers using =, > or < using nested ifs &  multiple ifs. | CO2 |
| 4 | Write a C program to find whether (using switch case)   1. A given number is prime or composite. 2. A given number is even or odd. | CO3 |
| 5 | Write a C program to generate the Fibonacci Series for a given number ‘n’. | CO3 |
| 6 | Write a C program of binary search for a user-given element in a set of ‘n’ numbers. | CO2 |
| 7 | Write a C program to perform matrix operations like addition, subtraction and transpose | CO4 |
| 8 | Write a C program to accept a string from console and to display the following on console (without using built-in functions):   1. Length of the string 2. Total number of characters in the string 3. Total number of vowels in the string 4. Copy one string into the other. | CO4 |
| 9 | Write a C program using functions to accept two strings from the console and perform the following operations (without using built-in functions):   1. Compare the strings: equal/not equal 2. Find the longer string. 3. Concatenate the two strings. 4. Find occurrence of substring in the main string | CO4 |
| 10 | Write a C program using functions to accept a one-dimensional array of integers and sort them in ascending order. | CO6 |

|  |  |  |
| --- | --- | --- |
| 11 | Write a C program of selection sorting using functions. | CO6 |
| 12 | Write a C program to find the factorial of a number using recursion | CO6 |
| 13 | Write a C program using structures to print the pay slip of an employee after accepting details like id. no, name, designation, department and basic salary | CO5 |
| 14 | 1. Write a program to demonstrate compile time memory allocation 2. Write a program to demonstrate runtime memory allocation | CO6 |
| 15 | Write a C Program to demonstrate Structure pointer. | CO5 |

Prepared By:

Faculty Name(s): Mr. Mahesh Arse, Dr. Kanhaiya Sharma, Mr. Sachin Gaikwad

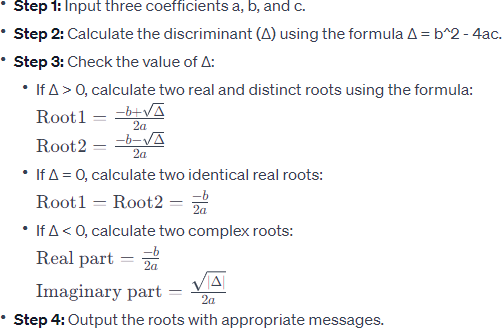
**Experiment/Practical 1**

**Title:** Design and develop a flowchart or an algorithm that takes three coefficients a,b and c of a quadratic equation (ax2+bx+c=0) as input and compute all possible roots.

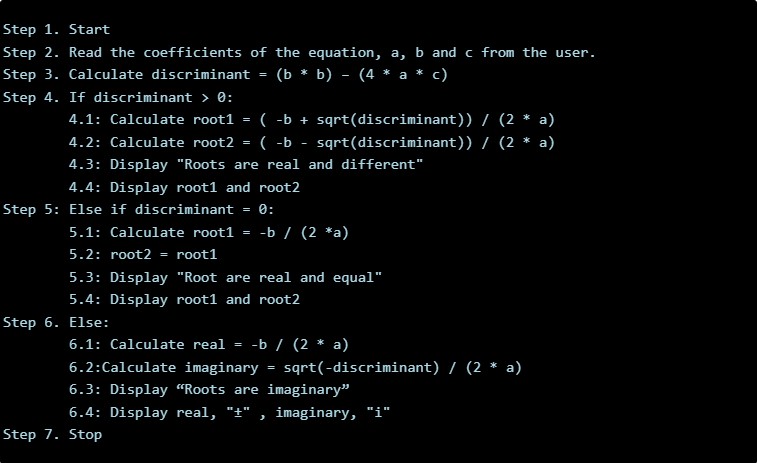
**Objective:** Students will learn and implement

* + Design and develop a flowchart or algorithm for computing roots of a quadratic equation.
  + Implement a C program based on the developed flowchart/algorithm.
  + Execute the program to output the possible roots for a given set of coefficients.
  + Provide appropriate messages for the output.

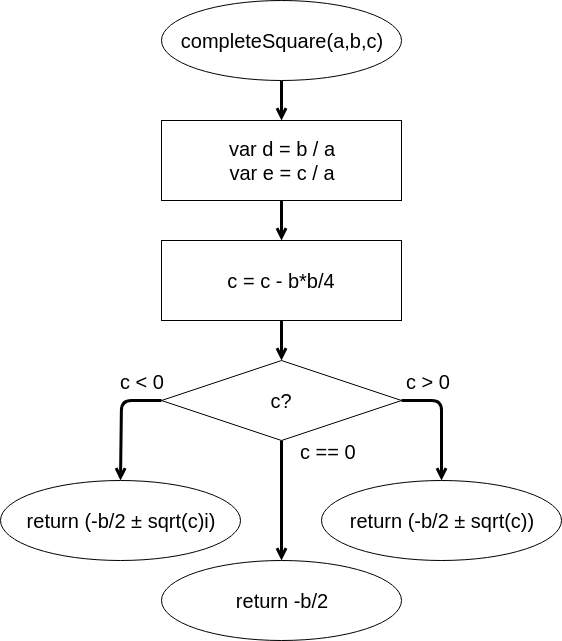
**Explanation/ Stepwise-Procedure/ Algorithm:**



# Code



**Input & Output:**



# Conclusion:

* + Summarize the outcomes of the program execution.
  + Highlight the nature of the roots (real, identical, or complex) based on the discriminant.

# Post Lab Questions:

* + Discuss the importance of the discriminant in determining the nature of roots.
  + How would you modify the program to handle cases where the coefficients may not form a valid quadratic equation?
  + Can you optimize the program for better performance?



# Practice Problem:

* + Create additional quadratic equations and manually compute their roots to verify the program's accuracy.
  + Modify the program to handle input validation (e.g., ensuring the user enters numerical values for coefficients).
  + Extend the program to handle cubic equations or higher-degree polynomials.

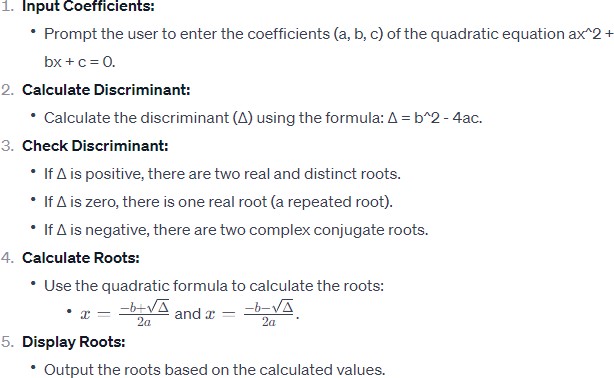
# Experiment/Practical 2

**Title:** Write a C program for the developed flowchart/algorithm and execute the same to output the possible roots for a given set of coefficients with appropriate messages.

**Objective:** Students will learn and implement

* + To understand the concept of quadratic equations.
  + To implement a C program to find the roots of a quadratic equation.
  + To practice the usage of mathematical formulas in programming.
  + To enhance problem-solving skills using programming.

**Explanation/ Stepwise-Procedure/ Algorithm:**



**Code** #include<stdio.h> #include<math.h> main()

{

float d,b,a,c,r1,r2,t;

printf("Enter a,b,c : "); scanf("%f,%f,%f",&a,&b,&c); d=(b\*b)-(4\*a\*c); printf("d= ..f\n",d); if (d…0)

{

printf("Both the roots are real and equal\n"); r1=(-b+sqrt(d))/2\*a; printf("%f",r1);

}

else if (d…..0)

{

printf("Both the roots are real and distinct\n"); r1=(-b+. (d))/2\*a;

r2=(-b-sqrt(d))/2\*a;

printf("%f",r1);

}

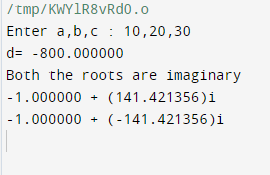
else

{

printf("Both the roots are imaginary\n"); t=(sqrt(-d))/2\*a; printf("%f + ..f)i\n",-b/(2\*a),t); printf("%f + (. f)i\n",-b/(2\*a),-t); }

}

# Input & Output:



**Conclusion:**

* + The implementation of the C program successfully finds the roots of a quadratic equation. The program takes user input for coefficients, calculates the discriminant, and determines the type and value of roots. It demonstrates the application of mathematical concepts in programming and enhances problem-solving skills.

# Post Lab Questions:

* + Explain the significance of the discriminant in solving quadratic equations.
  + How does the program handle complex roots? Provide an example.
  + Can the program handle quadratic equations with non-integer coefficients? Discuss.



# Practice Problem:

* + Modify the program to handle cases where coefficients can be non-integer numbers.
  + Implement error handling to validate user input for coefficients.
  + Extend the program to solve cubic equations.

# Experiment/Practical 3

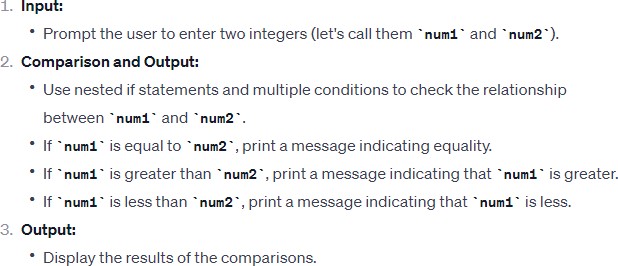


**Title:** Write a C Program to relate two integers using =, > or < using nested ifs & multiple ifs.

**Objective:** Students will learn and implement

* + To understand the usage of relational operators (=, >, <) in C programming.
  + To practice implementing nested if statements and multiple if conditions.
  + To relate and compare two integers and determine their relationship using the specified operators.

**Explanation/ Stepwise-Procedure/ Algorithm:**



# Code

#include <stdio.h>

int main() {

int num1, num2;

// Input two integers printf("Enter the first integer: "); scanf("%d", &num1);

printf("Enter the second integer: "); scanf("%d", &num2);

// Compare the two integers using nested if statements if (num1 ….. num2) {

printf("%d is equal to….d\n", num1, …..);

} else {

if (num1 > num2) {

printf(“. is greater than %d\n", num1, num2);

} else {

printf("%d is less than …..\n", num1, num2);

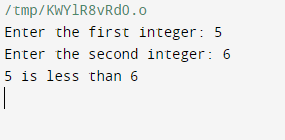
}

}

return 0….

}

# Input & Output:



**Conclusion:**

* + The C program successfully compares two integers using the relational operators =, >, and <, employing nested if statements and multiple conditions. The program effectively determines the relationship between the two input integers and provides the corresponding output.

# Post Lab Questions:

* + Explain the purpose of using nested if statements in this C program.
  + How would you modify the program to handle cases where the user inputs non-integer values?
  + Discuss the significance of using relational operators (=, >, <) in programming.



# Practice Problem:

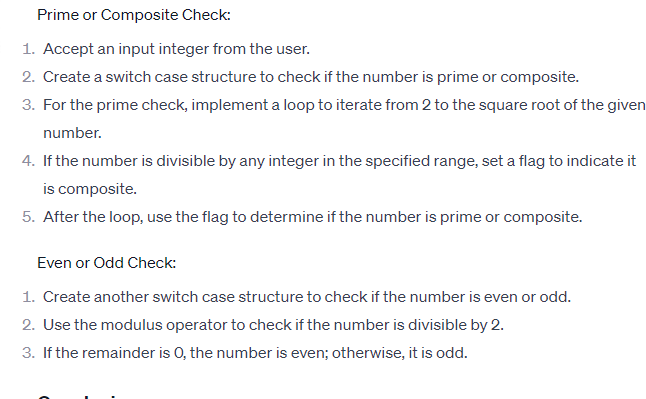
* + Write a C program to find the maximum of three integers using nested if statements.
  + Develop a C program to check if a given year is a leap year or not using conditional statements.
  + Create a C program to calculate the factorial of a given number using a loop and if statements.

# Experiment/Practical 4

**Title:** Write a C program to find whether (using switch case)

* + A given number is prime or composite.
  + A given number is even or odd.

**Objective:** Students will learn and implement

* + Determine whether a given number is prime or composite using a switch case in a C program.
  + Identify whether the given number is even or odd using a switch case in a C program.

**Explanation/ Stepwise-Procedure/ Algorithm:**

# Code

**For Prime or Composite**

#include <stdio.h> int main() {

int num;

// Input a number from the user printf("Enter a number: "); scanf("%d", &num)

switch (num) { case 0

case 1:

printf("d is neither prime nor composite.\n", num); break;

default:

int isPrime 1; // Assume the number is prime

// Check for factors from 2 to the square root of the number for (int i = 2; i \* i < num; ++i) {

if (num % i == 0) {

isPrime 0; // Set isPrime to 0 if a factor is found break;

}

}

if (isPrime)

printf("d is a prime number.\n", num); else

printf("%d is a composite number.\n", ); break;

}

return 0;

}

# For Even or Odd

#include <stdio.h> int main() {

int num;

// Input a number from the user printf("Enter a number: ") scanf("d", &num);

switch (num 2) { case 0:

printf("%d is an even number.\n", num); break;

case 1

case -1:

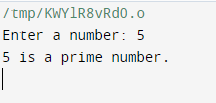
printf("%d is an odd number.\n", num); break

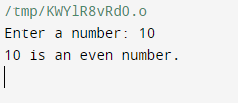
}

return 0

}

# Input & Output:





**Conclusion:**

* + Summarize the results obtained from the prime or composite check and the even or odd check. Conclude whether the program successfully determines these properties for the given number.

# Post Lab Questions:

* + Discuss the significance of using a switch case in this program compared to other control structures.
  + How would you modify the program to handle negative numbers or zero?
  + Explain the role of the square root in optimizing the prime or composite check algorithm.
  + Suggest improvements or additional features that could be added to the program.



# Practice Problem:

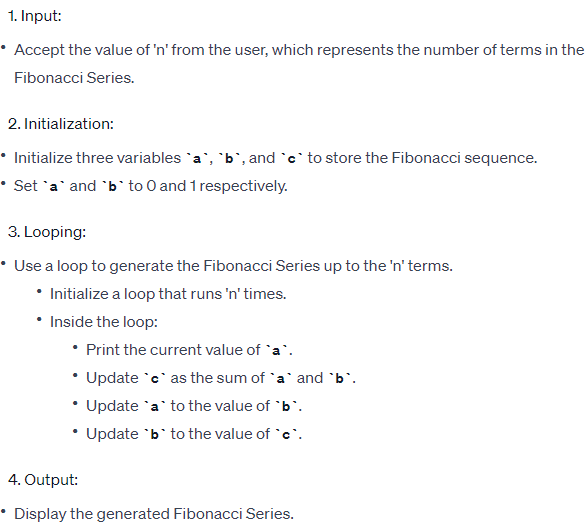
* + Write a C program to find the sum of digits of a given number.
  + Implement a C program to check whether a given year is a leap year or not.
  + Develop a C program to calculate the factorial of a given number.
  + Create a C program to find the largest among three numbers.

# Experiment/Practical 5

**Title:** Write a C program to generate the Fibonacci Series for a given number ‘n’.

**Objective:** Students will learn and implement

* + To understand the concept of the Fibonacci Series.
  + To implement a C program that generates the Fibonacci Series for a given number 'n'.
  + To practice looping structures in C programming.



**Explanation/ Stepwise-Procedure/ Algorithm:**

# Code

#include <stdio.h>

void generateFibonacci(int n) { int first = , second = 1, next;

printf("Fibonacci Series for n=%d: ", n)

for (int i = 0; i .. n; i++) { if (i <= 1) {

next = i;

} else {

next = first + second; first = second

second = next;

}

printf("%d ", next)

}

}

int main() { int n;

// Input the number of terms

printf("Enter the number of terms for Fibonacci series: "); scanf("%d", &n)

// Check for valid input if (n < 0) {

printf("Please enter a non-negative number.\n"); return 1 // Exit with an error code

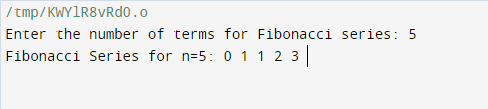
}

// Generate and print the Fibonacci series generateFibonacci(n);

return 0

}

# Input & Output:



**Conclusion:**

* + Summarize the purpose and outcome of the experiment.
  + Discuss the importance of understanding loops and sequence generation in programming.

# Post Lab Questions:

* + Explain the significance of initializing variables a, b, and c in the Fibonacci Series program.
  + How does the loop structure contribute to the generation of the Fibonacci Series?
  + Can you suggest modifications to the program to make it more efficient or handle larger values of 'n'?



# Practice Problem:

* + Modify the program to print the Fibonacci Series in reverse order.
  + Implement a recursive version of the Fibonacci Series program.
  + Write a program to find the sum of the Fibonacci Series up to the 'n' terms.

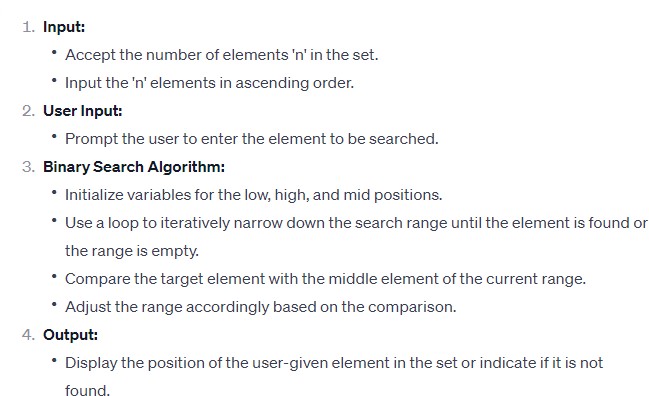
**Title:** Write a C program of binary search for a user-given element in a set of ‘n’ numbers.

**Experiment/Practical 6**

**Objective:** Students will learn and implement

* + To understand the binary search algorithm.
  + To implement binary search in the C programming language.
  + To create a program that allows a user to input a target element and find its position in a set of 'n' numbers using binary search.

**Explanation/ Stepwise-Procedure/ Algorithm:**



# Code

#include <stdio.h>

// Function to perform binary search

int binarySearch(int arr[], int size, int key) { int low = 0, high = size - 1

while (low <= high) {

int mid = low (high - low) / 2;

// If the key is present at the middle if (arr[mid] = key)

return mid

// If the key is smaller, ignore the right half else if (arr[mid] > key)

high = mid 1;

// If the key is larger, ignore the left half else

low = mid + 1

}

// If the key is not present in the array return -1;

}

int main() { int n, key;

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

int arr[n];

// Input the elements of the array printf("Enter %d sorted numbers:\n", n); for (int i = 0; i < n; i++) {

scanf("%d", &arr[i])

}

// Input the key to search

printf("Enter the element to search: "); scanf("%d", &key)

// Perform binary search

int result = binarySearch(arr n, key);

// Display the result if (result != -1) {

printf("Element %d found at index %d.\n", key, result);

} else {

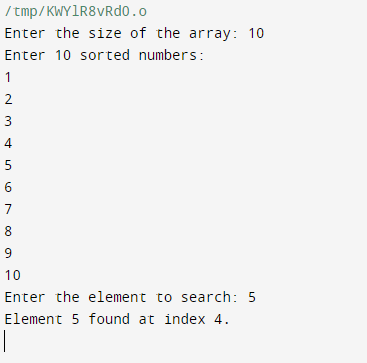
printf("Element d not found in the array.\n", key);

}

return 0;

}

# Input & Output:



**Conclusion:**

* + In conclusion, the implementation of the binary search algorithm in the C program provides an efficient way to locate a user-given element within a sorted set of 'n' numbers. The algorithm demonstrates the importance of a sorted input array for optimal search performance.

# Post Lab Questions:

* + What is the time complexity of the binary search algorithm?
  + How does binary search differ from linear search in terms of efficiency?
  + Can binary search be applied to an unsorted array? Why or why not?



# Practice Problem:

* + Implement a modified version of the binary search that returns the count of occurrences of the target element in the array.
  + Extend the program to handle a rotated sorted array and perform a binary search in such cases.
  + Explore and implement the recursive version of the binary search algorithm in C.

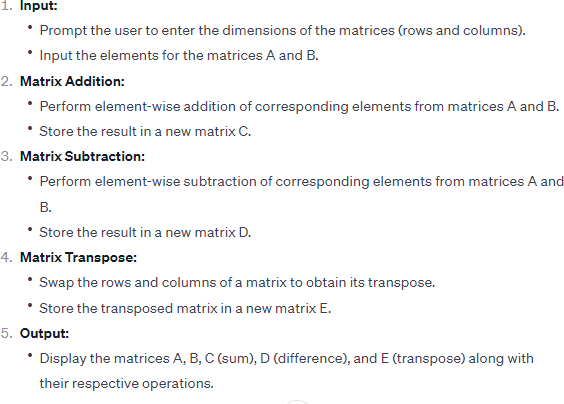
# Experiment/Practical 7

**Title:** Write a C program to perform matrix operations like addition, subtraction and transpose

**Objective:** Students will learn and implement

* + To understand the concept of matrix operations in C programming.
  + To implement matrix addition, subtraction, and transpose operations using C language.
  + To enhance programming skills in handling multi-dimensional arrays.

**Explanation/ Stepwise-Procedure/ Algorithm:**



# Code

#include <stdio.h>

// Function to input a matrix

void inputMatrix(int mat[0][10], int rows, int cols) { printf("Enter matrix elements:\n")

for (int i = 0; i < rows; i++) { for (int j = 0 j < cols; j++) {

printf("Enter element at position (%d, %d): ", i + 1, j + 1); scanf("%d", &mat[][j])

}

}

}

// Function to display a matrix

void displayMatrix(int mat[10][10], int rows, int cols) { printf("Matrix:\n");

for (int i = 0 i < rows; i++) { for (int j = 0; j < cols; j++) {

printf("d\t", mat[j]);

}

printf("\n");

}

}

// Function to add two matrices

void addMatrices(int mat1[10][10], int mat2[10][10], int result[10][10], int rows, int cols) { for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) { result[i][j] = mat1[i][j] + mat2[i]

}

}

}

// Function to subtract two matrices

void subtractMatrices(int mat1[10][10], int mat2[10][10], int result[10][10], int rows, int cols) {

for (int i = 0; i < rows; i+) { for (int j = 0; j < cols; j+) {

result[i][] = mat1[][j] - mat2[i][j];

}

}

}

// Function to transpose a matrix

void transposeMatrix(int mat[10][10], int result[10][10], int rows, int cols) { for (int i = 0; i < rows; i++) {

for (int j = ; j < cols; j++) { result[j][i] = mat[i][j];

}

}

}

int main() {

int mat1[1][10], mat2[10][10], result[10][10]; int rows, cols

// Input matrices

printf("Enter the number of rows and columns for the matrices: "); scanf("%d %d", &rows, &cols);

inputMatrix(mat1, rows, cols)

inputMatrix(mat2, rows, cols);

// Display matrices printf("\nMatrix 1:\n"); displayMatrix(mat1, rows, cols); printf("\nMatrix 2:\n"); displayMatrix(mat2, rows, cols)

// Add matrices

addMatrices(mat1, mat2, result, rows, cols); printf("\nMatrix Addition:\n"); displayMatrix(result, rows, cols)

// Subtract matrices

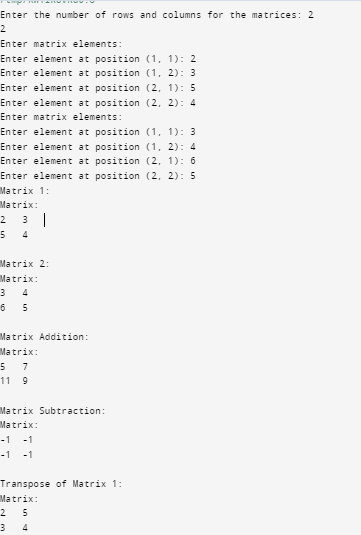
subtractMatrices(mat1, mat2, result, rows, cols); printf("\nMatrix Subtraction:\n"); displayMatrix(result, rows, cols);

// Transpose matrix 1 transposeMatrix(mat1, result, rows, cols); printf("\nTranspose of Matrix 1:\n"); displayMatrix(result, cols, rows)

return 0;

}

# Input & Output:



**Conclusion:**

* + Summarize the key learnings from the experiment.
  + Reflect on the challenges faced and how they were overcome.
  + Discuss the significance of matrix operations in various applications.

# Post Lab Questions:

* + How does the program handle matrices of different dimensions during addition and subtraction?
  + Explain the importance of the transpose operation in real-world scenarios.
  + What modifications would you make to the program to handle square matrices efficiently?



# Practice Problem:

* + Implement a C program to multiply two matrices.
  + Develop a C program to find the determinant of a square matrix.
  + Extend the program to handle matrix inversion.

**Title:** Write a C program to accept a string from console and to display the following on console (without using built-in functions):

**Experiment/Practical 8**

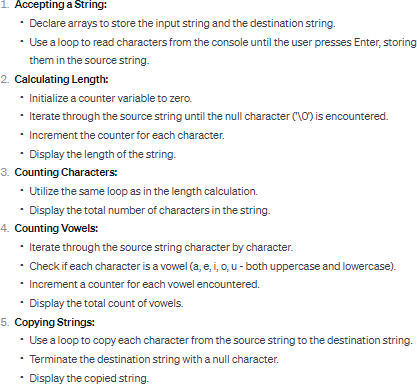
1. Length of the string
2. Total number of characters in the string
3. Total number of vowels in the string

(d) Copy one string into the other.

**Objective:** Students will learn and implement

* + To create a C program that accepts a string from the console.
  + To calculate and display the length of the entered string.
  + To determine and display the total number of characters in the string.
  + To count and display the total number of vowels in the string.
  + To copy the entered string into another string without using built-in functions.

**Explanation/ Stepwise-Procedure/ Algorithm:**



# Code

#include <stdio.h>

void stringInfo(char \*str, int \*length, int \*numChars, int \*numVowels) {

\*length = 0;

\*numChars = 0;

\*numVowels = 0

while (\*str != '\0') { (\*length)++; (\*numChars)++

// Check for vowels (assuming only lowercase for simplicity)

if (\*str == 'a' || \*str == 'e' || \*str == 'i' || \*str == 'o' || \*str == 'u') { (\*numVowels)++

}

str++;

}

}

void stringCopy(char \*source, char \*destination) { while (\*source != '\0') {

\*destination = \*source; source+;

destination++

}

\*destination = '\0'; // Null-terminate the destination string

}

int main() {

char inputString[100]; // Assuming a maximum string length of 100 characters int length, numChars, numVowels;

// Accept a string from the console printf("Enter a string: ");

fgets(inputString, sizeof(inputString), stdin);

// Remove the newline character if present int i = 0;

while (inputString[i] != '\0') { if (inputString[i] = '\n') { inputString[i] = '\0';

break;

} i++;

}

// Calculate and display string information stringInfo(inputString, &length, &numChars, &numVowels);

printf("Length of the string: %d\n", length)

printf("Total number of characters in the string: %d\n", numChars); printf("Total number of vowels in the string: d\n", numVowels);

// Copy one string into the other

char copiedString[100]; // Assuming a maximum string length of 100 characters stringCopy(inputString, copiedString);

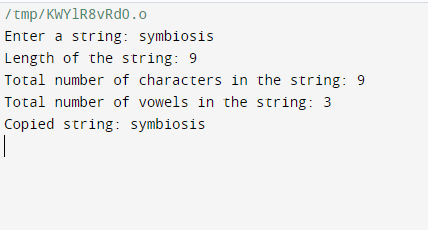
// Display the copied string

printf("Copied string: %s\n", copiedString);

return 0

}

# Input & Output:



**Conclusion:**

* + In conclusion, the C program successfully accomplishes the objectives outlined. It allows users to input a string, calculates its length, counts the total number of characters and vowels, and copies the string into another without using built-in functions. The program demonstrates fundamental string manipulation concepts in the C programming language.

# Post Lab Questions:

* + How can you modify the program to handle both uppercase and lowercase vowels?
  + Explain the significance of the null character ('\0') in C strings.
  + Can you optimize the program to handle strings of varying lengths without fixed-size arrays?



# Practice Problem:

* + Write a C program to reverse a given string without using built-in functions.
  + Develop a program to check if a given string is a palindrome.
  + Create a program to concatenate two strings without using the strcat() function.

# Experiment/Practical 9

**Title:** Write a C program using functions to accept two strings from the console and perform the following operations (without using built-in functions):

1. Compare the strings: equal/not equal
2. Find the longer string.
3. Concatenate the two strings.

(d) Find occurrence of substring in the main string

**Objective:** Students will learn and implement

* + To compare two strings without using built-in functions.
  + To find the longer string among the two.
  + To concatenate two strings without using built-in functions.
  + To find the occurrence of a substring in the main string without using built-in functions.

**Explanation/ Stepwise-Procedure/ Algorithm:**

**Step 1: Include Header Files** #include <stdio.h> #include <string.h>

**Step 2: Declare Functions**

void compareStrings(char str1[], char str2[]); void findLongerString(char str1[], char str2[]); void concatenateStrings(char str1[], char str2[]); void findSubstring(char mainStr, char subStr[]);

**Step 3: Main Function**

int main() {

char str1[100], str2[100]

// Input two strings from the console printf("Enter the first string: "); gets(str1);

printf("Enter the second string: "); gets(str2)

// Call functions compareStrings(str1, str2); findLongerString(str1, str2); concatenateStrings(str1, str2); findSubstring(str, str2);

return 0;

}

**Step 4: Implement Functions**

* + Implement compareStrings, findLongerString, concatenateStrings, and

findSubstring functions.

# Code

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

// Function to compare two strings

int compareStrings(const char \*str1, const char \*str2) { while (\*str1 && \*str2 && \*str1 == \*str2) {

str1++ str2++

}

if (\*str1 == \*str2) {

return 0; // Strings are equal

} else {

return 1 // Strings are not equal

}

}

// Function to find the longer string

void findLongerString(const char \*str1, const char \*str2) { int len1 = 0, len2 = 0;

while (\*str1) { len1++; str1+; }

while (\*str2) { len2+; str2++;

}

if (len1 > len2) {

printf("The first string is longer.\n");

} else if (len1 len2) {

printf("The second string is longer.\n");

} else {

printf("Both strings are of equal length.\n");

}

}

// Function to concatenate two strings

void concatenateStrings(const char \*str1, const char \*str2, char \*result) {

while (str1) {

\*result = \*str1; result+; str1++;

}

while (\*str2) {

\*result = \*str2; result++; str2++;

}

\*result = '\'; // Add null terminator to the end

}

// Function to find the occurrence of substring in the main string int findSubstring(const char \*mainStr, const char \*subStr) {

while (\*mainStr) {

const char \*mainTemp = mainStr; const char \*subTemp = subStr;

while (\*mainTemp && \*subTemp && \*mainTemp == \*subTemp) { mainTemp++;

subTemp++;

}

if (\*subTemp == '\0') {

return 1; // Substring found

}

mainStr++;

}

return 0; // Substring not found

}

int main() {

char str1[100], str2[100], result[200];

// Accept two strings from the console printf("Enter the first string: "); gets(str1);

printf("Enter the second string: "); gets(str2)

// Compare the strings

if (compareStrings(str1, str2) == 0) { printf("The strings are equal.\n");

} else {

printf("The strings are not equal.\n");

}

// Find the longer string findLongerString(str1, str2)

// Concatenate the two strings concatenateStrings(str1, str2, result); printf("Concatenated string: %s\n", result);

// Find occurrence of substring in the main string char subStr[50];

printf("Enter a substring to search in the main string: "); gets(subStr);

if (findSubstring(result, subStr)) {

printf(Substring found in the concatenated string.\n");

} else {

printf("Substring not found in the concatenated string.\n");

}

return 0

}

# Input & Output:

**Conclusion:**

* + In this experiment, we successfully implemented a C program to perform string operations without using built-in functions. The program can compare strings, find the longer string, concatenate two strings, and find the occurrence of a substring in the main string.

# Post Lab Questions:

* + Discuss the significance of avoiding built-in functions in string manipulation.
  + How would you modify the program to handle case-sensitive string comparison?
  + Explain the limitations of the current program and suggest improvements.



# Practice Problem:

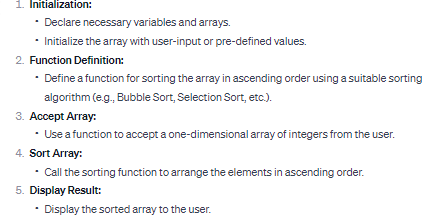
* + Modify the program to reverse each of the input strings.
  + Implement a function to count the number of vowels in a given string.
  + Extend the program to handle multiple substrings in the main string and count their occurrences.

# Experiment/Practical 10

**Title:** Write a C program using functions to accept a one-dimensional array of integers and sort them in ascending order.

**Objective:** Students will learn and implement

* + To understand the concept of arrays in C programming.
  + To learn the implementation of functions in C.
  + To grasp the sorting algorithm for arranging integers in ascending order.
  + To enhance problem-solving skills through practical coding.



**Explanation/ Stepwise-Procedure/ Algorithm:**

# Code

#include <stdio.h>

// Function to accept an array of integers void acceptArray(int arr[], int size) {

printf("Enter %d integers:\n", size); for (int i = 0; i < size; i++) {

scanf("%d", &arr[i])

}

}

// Function to swap two integers void swap(int \*a, int \*b) {

int temp = a;

\*a = b;

\*b = temp;

}

// Function to perform bubble sort on an array void bubbleSort(int arr[], int size) {

for (int i = 0 i < size - 1; i++) {

for (int j = 0; j < size - i - 1; j++) { if (arr[j] > arr[ + 1]) {

// Swap if the element found is greater than the next element swap(&arr[j], &arr[j + 1]);

}

}

}

}

// Function to display the sorted array void displayArray(int arr[], int size) {

printf("Sorted array in ascending order:\n"); for (int i = 0 i < size; i++) {

printf("%d ", arr[i]);

}

printf("\n");

}

int main() { int size;

// Input the size of the array printf("Enter the size of the array: "); scanf("%d" &size);

int array[size];

// Function call to accept array elements acceptArray(array, size);

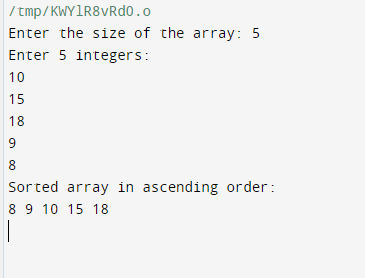
// Function call to perform bubble sort bubbleSort(array, size)

// Function call to display the sorted array displayArray(array, size);

return 0

}

# Input & Output:



**Conclusion:**

* + In conclusion, this C program successfully accepts a one-dimensional array of integers, sorts them in ascending order using a chosen sorting algorithm, and then displays the sorted array. The exercise reinforces the understanding of arrays, functions, and basic sorting techniques in C programming.

# Post Lab Questions:

* + What is the purpose of using a function for sorting the array?
  + Explain the chosen sorting algorithm and its time complexity.
  + How would you modify the program to sort the array in descending order?



# Practice Problem:

* + Implement a C program to find the sum and average of elements in the sorted array.
  + Extend the program to handle arrays of floating-point numbers.
  + Modify the sorting function to use a different sorting algorithm (e.g., Selection Sort) and compare the results.

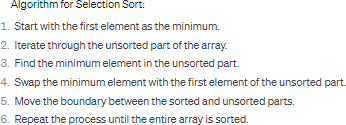
# Experiment/Practical 11

**Title:** Write a C program of selection sorting using functions.

**Objective:** Students will learn and implement

* + To understand the concept of selection sort algorithm.
  + To implement selection sort using functions in C programming.
  + To practice modular programming by using functions to break down the sorting process.
  + To analyze the time complexity of the selection sort algorithm.

**Explanation/ Stepwise-Procedure/ Algorithm:**



# Code

#include <stdio.h>

// Function to perform selection sort void selectionSort(int arr[], int n) {

int i, j, minIndex, temp;

for (i = 0; i < n-1; i+) {

// Assume the current index is the minimum minIndex = i;

// Find the minimum element in the unsorted part of the array for (j = i+1; j < n; j+) {

if (arr[j] < arr[minIndex]) { minIndex = j

}

}

// Swap the found minimum element with the element at index i temp = arr[i];

arr[] = arr[minIndex]; arr[minIndex] = temp;

}

}

// Function to print an array

void printArray(int arr[], int size) { for (int i=0; i < size; i++) {

printf("%d ", arr[i])

}

printf("\n");

}

int main() {

int arr[] = {, 25, 12, 22, 11};

int n = sizeof(arr) / sizeof(arr[0]);

printf("Original array: "); printArray(arr, n);

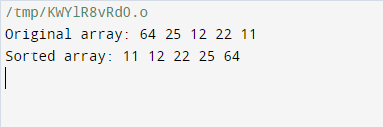
// Perform selection sort selection(arr, n);

printf("Sorted array: "); printArray(arr, n)

return 0;

}

# Input & Output:



**Conclusion:**

* + In conclusion, the experiment successfully demonstrated the implementation of the selection sort algorithm using modular programming in C. The use of functions allowed for a clear and organized code structure. The selection sort algorithm was effective in sorting the input array in ascending order. The analysis of the time complexity provides insights into the efficiency of the sorting process.

# Post Lab Questions:

* + Explain the role of functions in modular programming and how they enhance code readability.
  + Discuss the time complexity of the selection sort algorithm. How does it compare to other sorting algorithms?
  + Can you suggest any modifications to the program to make it more efficient?



# Practice Problem:

* + Implement the bubble sort algorithm using functions in C.
  + Modify the program to sort the array in descending order.
  + Explore and implement a different sorting algorithm, such as insertion sort or merge sort, using functions.
  + Investigate and implement a variation of the selection sort algorithm with improved performance.

# Experiment/Practical 12

**Title:** Write a C program to find the factorial of a number using recursion.

**Objective:** Students will learn and implement

* + To understand the concept of recursion in programming.
  + To implement a recursive function for calculating the factorial of a given number in C.
  + To analyze the efficiency and simplicity of a recursive approach for factorial calculation.

**Explanation/ Stepwise-Procedure/ Algorithm:**

* + Include Header Files:
  + Function Prototype:
  + Main Function:
  + Recursive Factorial Function:

# Code

#include<stdio.h>

// Function to calculate the factorial using recursion int factorial(int n) {

// Base case: factorial of 0 or 1 is 1 if (n = 0 || n == 1) {

return 1;

} else {

// Recursive case: n! = n \* (n-1)! return n \* factorial(n - 1)

}

}

int main() { int num;

// Input the number from the user printf("Enter a non-negative integer: "); scanf("%d", &num);

// Check if the input is non-negative if (num 0) {

printf("Factorial is not defined for negative numbers.\n");

} else {

// Call the factorial function and display the result

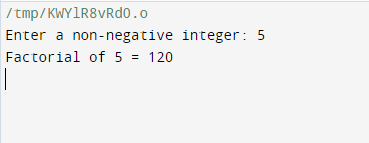
printf("Factorial of %d = %d\n", num, factorial(num));

}

return 0

}

# Input & Output:



**Conclusion:**

* + In this experiment, we successfully implemented a C program to find the factorial of a given number using recursion. The recursive approach provided a clear and concise solution, showcasing the power of recursion in solving mathematical problems. The program handles negative numbers by indicating that factorial is not defined for such cases.

# Post Lab Questions:

* + How does the recursive approach differ from an iterative one in calculating the factorial?
  + Discuss the scenarios where recursion might be advantageous or disadvantageous for factorial calculation.
  + Modify the program to handle larger values of 'num' and discuss any potential issues with this modification.



# Practice Problem:

* + Implement an iterative version of the factorial calculation in C.
  + Extend the program to calculate the factorial of a floating-point number.
  + Write a C program to find the factorial using a loop structure other than recursion.

# Experiment/Practical 13

**Title:** Write a C program using structures to print the pay slip of an employee after accepting details like id. no, name, designation, department and basic salary

**Objective:** Students will learn and implement

* + To design and implement a C program using structures for creating a pay slip for an employee.
  + To accept details such as employee ID, name, designation, department, and basic salary.
  + To use structures to organize and store the employee information efficiently.
  + To calculate and display the pay slip, including details such as gross salary, deductions, and net salary.
  + To enhance understanding of C programming concepts, including structures and basic arithmetic operations.

**Explanation/ Stepwise-Procedure/ Algorithm:**

* + Include Necessary Header Files:
  + Define the Structure:
  + Accept Employee Details:
  + Calculate Gross Salary, Deductions, and Net Salary:
  + Display Pay Slip:

# Code

#include <stdio.h>

// Define a structure for employee details struct Employee {

int id;

char name[50];

char designation[50]; char department[50]; float basicSalary;

};

// Function to calculate gross salary

float calculateGrossSalary(float basicSalary) {

// Consider some standard allowances and deductions for simplicity float allowances = 0.2 \* basicSalary; // 20% of basic salary

float deductions = 0.1 \* basicSalary; // 10% of basic salary

return basicSalary + allowances - deductions;

}

// Function to print pay slip

void printPaySlip(struct Employee employee) { printf("\nPay Slip\n");

printf(" \n")

printf("ID: %d\n", employee.id); printf("Name: %s\n", employee.name);

printf("Designation: %s\n", employee.designation); printf("Department: %s\n", employee.department); printf("Basic Salary: $%.2f\n", employee.basicSalary);

// Calculate and display gross salary

float grossSalary = calculateGrossSalary(employee.basicSalary); printf("Gross Salary: %.2f\n", grossSalary);

}

int main() {

// Declare and initialize an employee structure struct Employee emp1;

// Input employee details printf("Enter Employee ID: ") scanf("%d", &emp1.id);

printf("Enter Employee Name: "); scanf("%s", emp1.name);

printf("Enter Designation: "); scanf("s", emp1.designation);

printf("Enter Department: "); scanf("%s", emp1.department);

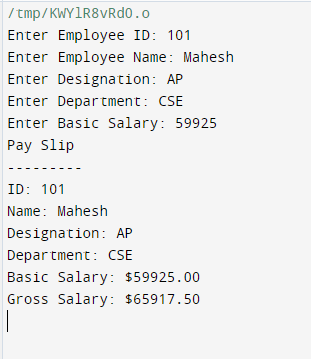
printf("Enter Basic Salary: "); scanf("%f", &emp1.basicSalary);

// Print the pay slip printPaySlip(emp1)

return 0;

}

# Input & Output:



**Conclusion:**

* + The C program successfully utilizes structures to create a pay slip for an employee. It takes user input for employee details and calculates gross salary, deductions, and net salary based on predefined allowances and deductions. The program then displays the comprehensive pay slip.

# Post Lab Questions:

* + How does the program utilize structures to organize employee information?
  + Explain the calculation of gross salary, allowances, deductions, and net salary.
  + Can you suggest additional features or modifications to improve the pay slip generation program?



# Practice Problem:

* + Modify the program to accept overtime hours and calculate overtime pay.
  + Implement a feature to handle different tax rates based on salary ranges.
  + Extend the program to handle multiple employees and generate pay slips for each.

# Experiment/Practical 14

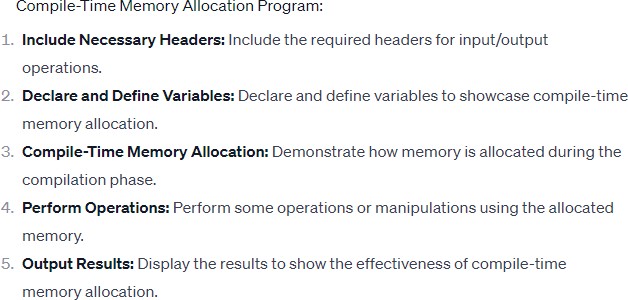
**Title:** Write a program to demonstrate compile time memory allocation

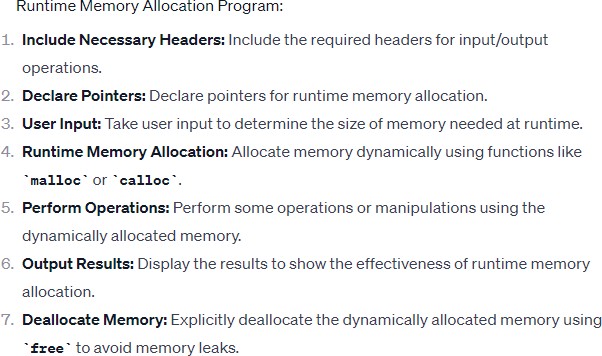
Write a program to demonstrate runtime memory allocation

**Objective:** Students will learn and implement

* + Understand the concept of compile-time memory allocation.
  + Gain insights into runtime memory allocation.
  + Learn how to write programs to demonstrate both compile-time and runtime memory allocation.
  + Compare and contrast compile-time and runtime memory allocation.

**Explanation/ Stepwise-Procedure/ Algorithm:**





# Code 14(a)

#include <stdio.h>

// Will occupy memory at compile time

// Global scope allocation int a;

void foo()

{

// Will occupy memory at compile time

// static allocation static int c = 10

c;

printf("c = %d\n", c);

}

int main()

{

// Will occupy the memory at compile time

// Automatic allocation int b;

// assigning value to test/print a=10

b=20;

// Print values printf("a = %d\n", a); printf("b = %d\n", b)

foo(); foo()

return 0;

}

# 14(b)

#include <stdio.h>

#include <stdlib.h> // To use dynamic memory allocation functions

int main()

{

// int pointer to store address of dynamic declared memory int \*parr;

// n - To get the limit

// i - Loop counter int n, i

printf("Enter limit: "); scanf("%d", &n);

// Allocate memory for n elements at runtime parr = malloc(n \* sizeof(int));

// check memory allocation error like overflow if (parr == NULL)

{

printf("Insufficient Memory!!!\n"); return 0

}

// Read n elements

printf("Input %d elements:\n", n); for (i = 0; i < n; i++)

{

printf("Enter element %d: ", i + 1); scanf("%d", (parr + i));

}

// Print array elements printf("Input elements: "); for (i = 0; i n; i++)

{

printf("%d " \*(parr + i));

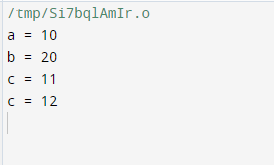
}

// Release memory occupied by dynamically allocated array free(parr);

return 0;

}

# Input & Output:

**14(a)**

# 14(b)

**Conclusion:**

* + Highlight the differences between compile-time and runtime memory allocation.
  + Discuss the advantages and disadvantages of each approach.
  + Emphasize the importance of proper memory management to prevent memory leaks.

# Post Lab Questions:

* + Compare and contrast compile-time and runtime memory allocation.
  + What are the advantages of dynamic memory allocation at runtime?
  + Explain the potential drawbacks of compile-time memory allocation.
  + How does memory deallocation help in preventing memory leaks?
  + Discuss situations where dynamic memory allocation is preferable over static memory allocation.



# Practice Problem:

* + Write a program to demonstrate the use of arrays with compile-time memory allocation.
  + Modify the runtime memory allocation program to handle error conditions gracefully.
  + Create a program that dynamically allocates memory for a linked list and performs basic operations.
  + Investigate and implement memory allocation strategies like realloc in a practical scenario.
  + Write a program that combines both compile-time and runtime memory allocation for a comprehensive understanding.

# Experiment/Practical 15

**Title:** Write a C Program to demonstrate Structure pointer.

**Objective:** Students will learn and implement

* + To understand the concept of structures in C programming.
  + To explore the use of pointers with structures.
  + To demonstrate the manipulation of structure members using pointers.
  + To enhance knowledge of memory allocation and data access through structure pointers.

**Explanation/ Stepwise-Procedure/ Algorithm:**

# Define a Structure:

Start by defining a structure that represents a specific entity. For example:

# Create Structure Variable:

Declare a structure variable and initialize it with sample data.

# Declare a Structure Pointer:

Declare a pointer of the structure type and assign the address of the structure variable.

# Access Structure Members using Pointers:

Use the arrow operator (->) to access structure members through pointers.

# Modify Structure Members using Pointers:

Illustrate how to modify structure members through pointers.

# Print Modified Structure:

Display the structure after modification.

# Code 15(a)

// C Program to demonstrate Structure pointer #include <stdio.h>

#include <string.>

// Creating Structure Student struct Student {

};

// variable of structure with pointer defined struct Student s, ptr;

int main()

{

int roll\_no; char name[30];

char branch[40]; int batch;

ptr = &s;

// Taking inputs

printf("Enter the Roll Number of Student\n"); scanf("%d", &ptr->roll\_no)

printf("Enter Name of Student\n"); scanf("%s", &ptr->name); printf("Enter Branch of Student\n"); scanf("%s", &ptr->branch); printf("Enter batch of Student\n"); scanf("d", &ptr->batch);

// Displaying details of the student printf("\nStudent details are: \n");

printf("Roll No: d\n", ptr->roll\_no); printf("Name: %s\n", ptr->name); printf("Branch: %s\n", ptr->branch); printf("Batch: %d\n", ptr->batch);

return 0;

}

# Input & Output:

**Conclusion:**

* Understanding the role of structures in organizing related data.
* Demonstrating the use of pointers with structures for efficient data manipulation.
* Observing how structure members can be accessed and modified using pointers.

# Post Lab Questions:

* What is the purpose of using pointers with structures in C?
* Explain the significance of the arrow operator (->) when working with structure pointers.
* How does the concept of structure pointers contribute to efficient memory management?



# Practice Problem:

* Write a C program to create an array of structures and use pointers to find the highest and lowest values.
* Extend the current program to handle multiple students by using an array of structures and dynamic memory allocation.