Parul University

FACULTY OF ENGINEERING & TECHNOLOGY BACHELOR OF TECHNOLOGY

COMPUTATIONAL THINKING FOR STRUCTURED DESIGN - 2

(303105151)

2nd SEMESTER

COMPUTER SCIENCE & ENGINEERING DEPARTMENT

Laboratory Manual

Faculty of Engineering & Technology
303105151 - Computational thinking for structured design

CERTIFICATE

This is to certify that

Mr./Ms *with*

enrolment no *has successfully completed his/her laboratory*

experiments in the Computational Thinking for Structured Design-2(303105151)

from the department of *during the*

academic year .....

Date of Submission:..... Staff In charge:.....

Head of Department:.....

TABLE OF CONTENT

iii)	Write a program to demonstrate accessing array elements using pointers.						
4.	Practical:4						
i)	Write a program to demonstrate an array of pointers to strings.						
ii)	Write a program to demonstrate the use of a pointer to a pointer.						
iii)	Write a program to demonstrate the call by value and call by reference.						
5.	Practical:5						
i)	Write a program to demonstrate file inclusion using #include user's own header file.						
ii)	Define a macro for a constant value and use it to calculate the perimeter of a rectangle.						
iii)	Write a program to calculate the square of a number using a macro.						
iv)	Write a program to include different code sections based on a macro value.						
6.	Practical:6						
i)	Define an enumeration for the days of the week and display the name of the day based on its value.						
ii)	Define a structure for employee details (ID, name, salary) and use an array of structures to store and display details of 5						

	employees.						
iii)	Define a structure for a student's academic record that includes a nested structure for personal details (name, age, address).						
7.	Practical:7						
i)	Pass a structure containing two integers to a function to calculate their sum.						
ii)	Define a union to store data of different types (integer, float, and character).						
iii)	Use typedef to create an alias for a structure representing a complex number and perform addition of two complex numbers.						
8.	Practical:8						
i)	Write a program to create a file and write user input into it.						
ii)	Write a program to read and display the contents of a file.						
iii)	Write a program to append text to an existing file.						
9.	Practical:9						
i)	Write a program to search for a specific word in a file and display its occurrences.						
ii)	Write a program to reverse the contents of a file and save it in another file.						
iii)	Write a program to read a specific line from a file.						
10	Practical:10						
i)	Write a program to sort an						

	array using the Bubble Sort algorithm.						
ii)	Write a program to sort an array using the Insertion Sort algorithm.						
iii)	Write a program to perform a linear search on an array.						
iv)	Write a program to perform a binary search on a sorted array.						
v)	Write a program to sort an array using the Selection Sort algorithm.						

Practical-1

1. Write a c program to increase or decrease the existing size of an 1D array.

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

void resizeArray(int **arr, int *oldSize, int newSize) {
  int *newArray = (int *)malloc(newSize * sizeof(int)); // Allocate memory
  for new array if (newArray == NULL) {
    printf("Memory
    allocation failed!\n");
    return;
  }

  for (int i = 0; i < (*oldSize < newSize ? *oldSize : newSize);
    i++) { newArray[i] = (*arr)[i];
  }


  if (*oldSize >
    newSize) {
    free(*arr);
  }


  *arr = newArray;
  *oldSize = newSize;

  printf("Array resized to size %d\n", newSize);
}

int main() {
  int *arr = (int *)malloc(5 * sizeof(int)); // Initial
  size of 5 for (int i = 0; i < 5; i++) {
    arr[i] = i + 1; // Initialize elements (optional)
  }
```

```
int oldSize =
5;
printf("Origina
l array: ");

for (int i = 0; i < oldSize;
 i++) { printf("%d ",
 arr[i]);

}
```

```
    printf("\n");

    //
    Increase
    size int
    newSize
    = 8;
    resizeArray(&arr, &oldSize, newSize);

    printf("Resized array: ");
    for (int i = 0; i <
    newSize; i++) {
    printf("%d ", arr[i]);
    }
    printf("\n");

    // Decrease size
    (optional) newSize
    = 3;
    resizeArray(&arr, &oldSize, newSize);

    printf("Resized array: ");
    for (int i = 0; i <
    newSize; i++) {
    printf("%d ", arr[i]);
    }
    printf("\n");

    free
    (
    arr);
    retu
    r   n
    0;
}
```

Output:

```
garlicbread@pop-os:~/Coding/C$ cd "/home/garlicbread/Coding/C/" && gcc relesize
m && "/home/garlicbread/Coding/C/"relesize
Original array: 1 2 3 4 5
Array resized to size 8
Resized array: 1 2 3 4 5 0 0 0
Array resized to size 3
Resized array: 1 2 3
```

2. Write a c program on 2D array to Increase & Decrease

i) No of subarrays

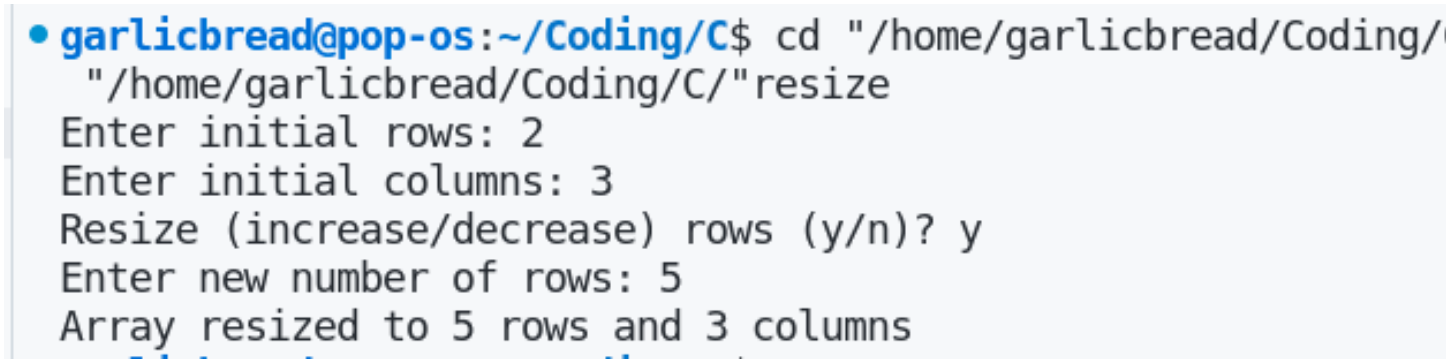
ii) elements in the subarrays

```
#include<stdio.h>

#include<stdlib.h>


void resizeArray(int ***arr, int *rows, int *cols, int newRows, int
 newCols) { int **newArray = (int **)malloc(newRows *
 sizeof(int *)); if (newArray == NULL) {

  printf("Memory
  allocation failed!\n");
  return;

 }

 for (int i = 0; i < newRows; i++) {
  newArray[i] = (int *)malloc(newCols
  * sizeof(int)); if (newArray[i] ==
  NULL) { printf("Memory allocation

   failed!\n"); return;

  }

 }

 // Copy elements based on size changes

 for (int i = 0; i < (*rows < newRows ? *rows :
  newRows); i++) { for (int j = 0; j < (*cols < newCols
  ? *cols : newCols); j++) { newArray[i][j] = (*arr)[i][j];

  }

 }
```

```
  // Free old memory
  for (int i = 0; i <
  *rows; i++) {
  free((*arr)[i]);
  }
  free(*arr);
  *arr = newArray;
  *rows = newRows;
  *cols = newCols;

  printf("Array resized to %d rows and %d columns\n", newRows, newCols);
}
int main() {
  int **arr, rows, cols;
  printf("Enter initial
  rows: "); scanf("%d",
  &rows);
  printf("Enter initial columns: ");
  scanf("%d", &cols);
  arr = (int **)malloc(rows * sizeof(int
  *)); for (int i = 0; i < rows; i++) {
    arr[i] = (int *)malloc(cols * sizeof(int));
  }
  printf("Resize (increase/decrease) rows
  (y/n)? "); char choice;
  scanf(" %c",
  &choice); if
  (choice == 'y') {
  int newRows;
    printf("Enter new number
    of rows: "); scanf("%d",
    &newRows);
    resizeArray(&arr, &rows, &cols, newRows, cols);
  }
```

```
  for (int i = 0; i <
    rows; i++) {
    free(arr[i]);
  }
  free(arr);
  return 0;
}
```

Output:



PRACTICAL:2

1. Write a program to compare malloc and calloc by allocating memory for an array and printing the uninitialized values.

```
#include <stdio.h>
#include <stdlib.h>
void compare_malloc_calloc(int n) {
    int *arr_malloc, *arr_calloc;
    // Allocating memory using malloc
    arr_malloc = (int *)malloc(n * sizeof(int));
    if (arr_malloc == NULL) {
        printf("Memory allocation failed using malloc.\n");
        return;
    }

    // Allocating memory using calloc
    arr_calloc = (int *)calloc(n, sizeof(int));
    if (arr_calloc == NULL) {
        printf("Memory allocation failed using calloc.\n");
        free(arr_malloc);
        return;
    }
    printf("Initial values in the array allocated by malloc (uninitialized):\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr_malloc[i]);
    }
    printf("\n");
    printf("Initial values in the array allocated by calloc (initialized to 0):\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr_calloc[i]);
    }
```

```
    printf("\n");
    // Assigning values to both arrays
    for (int i = 0; i < n; i++) {
        arr_malloc[i] = i + 1;
        arr_calloc[i] = i + 1;
    }
    printf("Values in the array after assigning values (malloc):\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr_malloc[i]);
    }
    printf("\n");
    printf("Values in the array after assigning values (calloc):\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr_calloc[i]);
    }
    printf("\n");
    // Free allocated memory
    free(arr_malloc);
    free(arr_calloc);
}
int main() {
int n;
printf("Enter the size of the array: ");
scanf("%d", &n);
compare_malloc_calloc(n);
return 0;
}
```

OUTPUT

```
Enter the size of the array: 5
Initial values in the array allocated by malloc (uninitialized):
1406211352  0  1406211352  0  1406211352
Initial values in the array allocated by calloc (initialized to 0):
0 0 0 0 0
Values in the array after assigning values (malloc):
1 2 3 4 5
Values in the array after assigning values (calloc):
1 2 3 4 5
```

2. Write a program to dynamically allocate memory for a string and store a user-entered string

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
char *str;
int size;
// Ask user for the size of the string
printf("Enter the length of the string (number of characters): ");
scanf("%d", &size);
// Allocate memory dynamically
str = (char *)malloc((size + 1) * sizeof(char)); // +1 for null terminator
if (str == NULL) {
printf("Memory allocation failed.\n");
return 1;
}
```

```
// Clear input buffer (if needed)
while (getchar() != '\n');
// Take string input from user
printf("Enter the string: ");
fgets(str, size + 1, stdin);
// Remove newline character if fgets captures it
str[strcspn(str, "\n")] = '\0';
// Print the entered string
printf("You entered: %s\n", str);
// Free the allocated memory
free(str);
return 0;
}
```

OUTPUT

```
Enter the length of the string (number of characters): 10
Enter the string: HelloWorld
You entered: HelloWorld
```

3. Write a program to find the largest element in an array dynamically allocated using malloc

```
#include <stdio.h>
#include <stdlib.h> // For malloc and free
int main() {
int n, i;
int *arr;
// Prompt the user for the size of the array
printf("Enter the number of elements: ");
scanf("%d", &n);
```

```
// Dynamically allocate memory for the array
arr = (int *)malloc(n * sizeof(int));
if (arr == NULL) {
printf("Memory allocation failed!\n");
return 1; // Exit the program if memory allocation fails
}
// Input elements into the array
printf("Enter %d elements:\n", n);
for (i = 0; i < n; i++) {
scanf("%d", &arr[i]);
}
// Find the largest element in the array
int max = arr[0];
for (i = 1; i < n; i++) {
if (arr[i] > max) {
max = arr[i];
}
}
// Output the largest element
printf("The largest element is: %d\n", max);
// Free the dynamically allocated memory
free(arr);
return 0;
}
```

OUTPUT

```
Enter the number of elements: 5

Enter 5 elements:

10 20 5 15 25

The largest element is: 25
```

PRACTICAL-03

1. Write a program to demonstrate how to access a variable using its pointer.

```
#include <stdio.h>

int main() {
    int num = 42;       // Declare an integer variable and initialize it
    int *ptr = &num;    // Declare a pointer and store the address of 'num'

    // Print the value of 'num' directly
    printf("Value of num: %d\n", num);

    // Print the address of 'num'
    printf("Address of num: %p\n", (void *)&num);

    // Print the value stored in the pointer (address of 'num')
    printf("Value of ptr (address of num): %p\n", (void *)ptr);

    // Print the value of 'num' using the pointer
    printf("Value of num using pointer: %d\n", *ptr);

    return 0;
}
```

OUTPUT

```
Value of num: 42
Address of num: 0x7fff52b2ff04
Value of ptr (address of num): 0x7fff52b2ff04
Value of num using pointer: 42
```

2. Write a program to swap two numbers using pointers.

```
#include <stdio.h>

// Function to swap two numbers using pointers
void swap(int *a, int *b) {
    int temp;   // Temporary variable for swapping
    temp = *a;  // Store the value of *a in temp
    *a = *b;    // Assign the value of *b to *a
    *b = temp;  // Assign the value of temp to *b
}

int main() {
    int num1, num2;

    // Input two numbers
    printf("Enter the first number: ");
    scanf("%d", &num1);
    printf("Enter the second number: ");
    scanf("%d", &num2);

    // Display numbers before swapping
    printf("\nBefore swapping: num1 = %d, num2 = %d\n", num1, num2);

    // Call the swap function
    swap(&num1, &num2);

    // Display numbers after swapping
    printf("After swapping: num1 = %d, num2 = %d\n", num1, num2);

    return 0;
}
```

OUTPUT

Enter the first number: 5

Enter the second number: 10

Before swapping: num1 = 5, num2 = 10

After swapping: num1 = 10, num2 = 5

3. Write a program to demonstrate accessing array elements using pointers

```
#include <stdio.h>

int main() {
    // Define and initialize an array
    int arr[] = {10, 20, 30, 40, 50};
    int *ptr; // Pointer to integer
    int size = sizeof(arr) / sizeof(arr[0]);

    // Point to the start of the array
    ptr = arr;

    printf("Array elements accessed using pointers:\n");
    for (int i = 0; i < size; i++) {
        printf("Element %d: %d (Pointer Address: %p)\n", i, *(ptr + i), (ptr + i));
    }

    return 0;
}
```

OUTPUT

```
Array elements accessed using pointers:
Element 0: 10 (Pointer Address: 0x7ffeeb2dbcc0)
Element 1: 20 (Pointer Address: 0x7ffeeb2dbcc4)
Element 2: 30 (Pointer Address: 0x7ffeeb2dbcc8)
Element 3: 40 (Pointer Address: 0x7ffeeb2dbccc)
Element 4: 50 (Pointer Address: 0x7ffeeb2dbcd0)
```

PRACTICAL:4

1. Write a program to demonstrate an array of pointers to strings.

```
#include <stdio.h>

int main() {
    // Array of pointers to strings
    const char *fruits[] = {"Apple", "Banana", "Cherry", "Date", "Elderberry"};

    // Calculate the number of strings in the array
    int n = sizeof(fruits) / sizeof(fruits[0]);

    // Print each string using the pointers
    printf("List of fruits:\n");
    for (int i = 0; i < n; i++) {
        printf("%s\n", fruits[i]);
    }

    return 0;
}
```

OUTPUT:

```
Output

List of fruits:
Apple
Banana
Cherry
Date
Elderberry


=== Code Execution Successful ===
```

2. Write a program to demonstrate the use of a pointer to a pointer.

```
#include <stdio.h>

int main() {
    // A normal variable
    int number = 42;

    // Pointer to the variable
    int *ptr = &number;

    // Pointer to the pointer
    int **ptr_to_ptr = &ptr;

    // Display values and addresses
    printf("Value of number: %d\n", number);
    printf("Address of number: %p\n", (void*)&number);

    printf("\nValue stored in ptr (address of number): %p\n", (void*)ptr);
    printf("Value pointed to by ptr: %d\n", *ptr);

    printf("\nValue stored in ptr_to_ptr (address of ptr): %p\n", (void*)ptr_to_ptr);
    printf("Value pointed to by ptr_to_ptr (value of ptr): %p\n", (void*)*ptr_to_ptr);
    printf("Value pointed to by the value pointed to by ptr_to_ptr: %d\n", **ptr_to_ptr);

    return 0;
}
```

OUTPUT:

```
Output

Value of number: 42
Address of number: 0x7ffe40198f24

Value stored in ptr (address of number): 0x7ffe40198f24
Value pointed to by ptr: 42

Value stored in ptr_to_ptr (address of ptr): 0x7ffe40198f18
Value pointed to by ptr_to_ptr (value of ptr): 0x7ffe40198f24
Value pointed to by the value pointed to by ptr_to_ptr: 42


=== Code Execution Successful ===
```

3. Write a program to demonstrate the call by value and call by reference.

```
#include <stdio.h>

// Function for call by value
void callByValue(int x) {
    x = x + 10; // Modify the value of x
    printf("Inside callByValue: x = %d\n", x);
}

// Function for call by reference
void callByReference(int *x) {
    *x = *x + 10; // Modify the value pointed to by x
    printf("Inside callByReference: *x = %d\n", *x);
}

int main() {
    int value = 20;

    // Call by value
```

```
    printf("Before callByValue: value = %d\n", value);
    callByValue(value);
    printf("After callByValue: value = %d\n", value);

    printf("\n");

    // Call by reference
    printf("Before callByReference: value = %d\n", value);
    callByReference(&value);
    printf("After callByReference: value = %d\n", value);

    return 0;
}
```

OUTPUT:

```
Output

Before callByValue: value = 20
Inside callByValue: x = 30
After callByValue: value = 20

Before callByReference: value = 20
Inside callByReference: *x = 30
After callByReference: value = 30


=== Code Execution Successful ===
```

PRACTICAL:5

1. Write a program to demonstrate file inclusion using #include user's own header file.

1. **myheader.h** (Custom Header File)

```
#ifndef MYHEADER_H
#define MYHEADER_H

// Function prototype
int add(int a, int b);

#endif
```

2. **myheader.c** (Header File Implementation)

```
#include "myheader.h"

// Function definition
int add(int a, int b) {
    return a + b;
}
```

3. **main.c** (Main Program)

```
#include <stdio.h>
#include "myheader.h" // Including the custom header file

int main() {
    int num1, num2, sum;

    printf("Enter two numbers: ");
    scanf("%d %d", &num1, &num2);

    // Calling the function from the custom header file
    sum = add(num1, num2);

    printf("The sum of %d and %d is %d\n", num1, num2, sum);
```

```
    return 0;
}
```

OUTPUT:

```
Enter two numbers: 5 10
The sum of 5 and 10 is 15
```

2. Define a macro for a constant value and use it to calculate the perimeter of a rectangle.

CODE:

```
#include <stdio.h>
// Define a macro for a constant value (2 for the perimeter formula)
#define MULTIPLIER 2
int main() {
    int length, width, perimeter;
    // Input the length and width of the rectangle
    printf("Enter the length of the rectangle: ");
    scanf("%d", &length);
    printf("Enter the width of the rectangle: ");
    scanf("%d", &width);
    // Calculate the perimeter using the macro
    perimeter = MULTIPLIER * (length + width);
    // Output the result
    printf("The perimeter of the rectangle is: %d\n", perimeter);
    return 0;
}
```

OUTPUT:

```
Output

Enter the length of the rectangle: 3
Enter the width of the rectangle: 2
The perimeter of the rectangle is: 10


=== Code Execution Successful ===
```

3. Write a program to calculate the square of a number using a macro.

CODE:

```
#include <stdio.h>
// Define a macro to calculate the square of a number
#define SQUARE(x) ((x) * (x))

int main() {
    int number, result;

    // Input the number
    printf("Enter a number: ");
    scanf("%d", &number);
    // Calculate the square using the macro
    result = SQUARE(number);
    // Output the result
    printf("The square of %d is: %d\n", number, result);
    return 0;
}
```

OUTPUT:

```
Output

Enter a number: 2
The square of 2 is: 4


=== Code Execution Successful ===
```

4. Write a program to include different code sections based on a macro value.

CODE:

```
#include <stdio.h>
// Define a macro to control the code sections
#define MODE 1  // Change this value to 0 or 1 to include different code sections
int main() {
    #if MODE == 1
        // Code section for MODE 1
        printf("MODE is set to 1: Executing code for MODE 1.\n");
        printf("This section performs Task A.\n");
    #elif MODE == 0
        // Code section for MODE 0
        printf("MODE is set to 0: Executing code for MODE 0.\n");
        printf("This section performs Task B.\n");
    #else
        // Code section for invalid MODE
        printf("Invalid MODE value. Please set MODE to 0 or 1.\n");
    #endif
    return 0;
}
```

OUTPUT:

Output

```
MODE is set to 1: Executing code for MODE 1.
This section performs Task A.


=== Code Execution Successful ===
```

PRACTICAL:6

1. Define an enumeration for the days of the week and display the name of the day based on its value.

CODE:

```
#include <stdio.h>

// Define an enumeration for days of the week
typedef enum {
    SUNDAY,
    MONDAY,
    TUESDAY,
    WEDNESDAY,
    THURSDAY,
    FRIDAY,
    SATURDAY
} DayOfWeek;

int main() {
    DayOfWeek day;
    int dayValue;

    // Prompt user for a day value (0-6)
    printf("Enter a number (0 for Sunday, 1 for Monday, ..., 6 for Saturday): ");
    scanf("%d", &dayValue);

    // Validate the input
    if (dayValue < 0 || dayValue > 6) {
        printf("Invalid day value. Please enter a number between 0 and 6.\n");
        return 1;
    }

    // Assign the day value to the enumeration variable
```

```
    day = (DayOfWeek)dayValue;

    // Display the name of the day based on its value
    printf("The day is: ");
    switch (day) {
        case SUNDAY:    printf("Sunday\n"); break;
        case MONDAY:    printf("Monday\n"); break;
        case TUESDAY:   printf("Tuesday\n"); break;
        case WEDNESDAY: printf("Wednesday\n"); break;
        case THURSDAY:  printf("Thursday\n"); break;
        case FRIDAY:    printf("Friday\n"); break;
        case SATURDAY:  printf("Saturday\n"); break;
        default:        printf("Invalid day\n"); break;
    }

    return 0;
}
```

OUTPUT:

```
Output                                                    Clear
Enter a number (0 for Sunday, 1 for Monday, ..., 6 for Saturday): 4
The day is: Thursday


=== Code Execution Successful ===
```

2. Define a structure for employee details (ID, name, salary) and use an array of structures to store and display details of 5 employees.

CODE:

```
#include <stdio.h>

// Define the structure for employee details
```

```
struct Employee {
    int id;
    char name[50];
    float salary;
};

int main() {
    struct Employee employees[5]; // Array to store details of 5 employees
    int i;

    // Input details for 5 employees
    for (i = 0; i < 5; i++) {
        printf("Enter details for employee %d:\n", i + 1);

        printf("ID: ");
        scanf("%d", &employees[i].id);

        printf("Name: ");
        scanf(" %[^\n]s", employees[i].name); // Read string with spaces

        printf("Salary: ");
        scanf("%f", &employees[i].salary);

        printf("\n");
    }

    // Display the details of 5 employees
    printf("Employee Details:\n");
    printf("-----------------------------------------------\n");
    printf("ID\tName\t\tSalary\n");
    printf("-----------------------------------------------\n");
    for (i = 0; i < 5; i++) {
        printf("%d\t%-15s%.2f\n", employees[i].id, employees[i].name, employees[i].salary);
```

```
    return 0;
}
```

OUTPUT:

Output

```
Enter details for employee 1:
ID: 1
Name: WEE
Salary: 2

Enter details for employee 2:
ID: 2
Name: KJJ
Salary: 6

Enter details for employee 3:
ID: 3
Name: RFRDG
Salary: 9

Enter details for employee 4:
ID: 4
Name: RFRE
Salary: 6

Enter details for employee 5:
ID: 5
Name: RTR
Salary: 7

Employee Details:
--------------------------------------
ID  Name        Salary
--------------------------------------
1   WEE           2.00
2   KJJ           6.00
3   RFRDG         9.00
4   RFRE          6.00
5   RTR           7.00
```

3. Define a structure for a student's academic record that includes a nested structure for personal details (name, age, address).

CODE:

```
#include <stdio.h>


// Define the nested structure for personal details
struct PersonalDetails {
    char name[50];
    int age;
    char address[100];
};


// Define the main structure for the student's academic record
```

```
struct AcademicRecord {
    struct PersonalDetails personal; // Nested structure for personal details
    int rollNumber;
    float marks;
};

int main() {
    struct AcademicRecord student; // Declare a variable for the student's record

    // Input personal details
    printf("Enter student's personal details:\n");

    printf("Name: ");
    scanf(" %[^\n]s", student.personal.name); // Read string with spaces

    printf("Age: ");
    scanf("%d", &student.personal.age);

    printf("Address: ");
    scanf(" %[^\n]s", student.personal.address);

    // Input academic details
    printf("\nEnter student's academic details:\n");

    printf("Roll Number: ");
    scanf("%d", &student.rollNumber);

    printf("Marks: ");
    scanf("%f", &student.marks);

    // Display the student's complete record
    printf("\nStudent's Complete Record:\n");
```

```
    printf("----------------------------------\n");
    printf("Name       : %s\n", student.personal.name);
    printf("Age        : %d\n", student.personal.age);
    printf("Address    : %s\n", student.personal.address);
    printf("Roll Number: %d\n", student.rollNumber);
    printf("Marks      : %.2f\n", student.marks);

    return 0;
}
```

OUTPUT:

Output

```
Enter student's personal details:
Name: abc
Age: 23
Address: vadodara

Enter student's academic details:
Roll Number: 12
Marks: 90

Student's Complete Record:
----------------------------------
Name       : abc
Age        : 23
Address    : vadodara
Roll Number: 12
Marks      : 90.00


=== Code Execution Successful ===
```

PRACTICAL:7

1. Pass a structure containing two integers to a function to calculate their sum.

CODE:

```
#include <stdio.h>

// Define a structure to hold two integers
struct Numbers {
    int num1;
    int num2;
};

// Function to calculate the sum of two integers in the structure
int calculateSum(struct Numbers nums) {
    return nums.num1 + nums.num2;
}

int main() {
    struct Numbers numbers; // Declare a structure variable
    int sum;

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

    printf("Enter the second number: ");
    scanf("%d", &numbers.num2);

    // Pass the structure to the function and calculate the sum
    sum = calculateSum(numbers);

    // Display the sum
    printf("The sum of %d and %d is: %d\n", numbers.num1, numbers.num2, sum);
```

return 0;

}

OUTPUT:

```
Output

Enter the first number: 1
Enter the second number: 2
The sum of 1 and 2 is: 3


=== Code Execution Successful ===
```

2. Define a union to store data of different types (integer, float, and character).

CODE:

```
#include <stdio.h>

// Define a union to store different types of data
union Data {
    int intValue;
    float floatValue;
    char charValue;
};

int main() {
    union Data data; // Declare a union variable

    // Store and display an integer
    data.intValue = 42;
    printf("Integer value: %d\n", data.intValue);

    // Store and display a float
    data.floatValue = 3.14f;
    printf("Float value: %.2f\n", data.floatValue);
```

```
    // Store and display a character
    data.charValue = 'A';
    printf("Character value: %c\n", data.charValue);

    // Note: Only one value can be stored at a time in a union
    printf("\nAfter storing a character, previous values are overwritten.\n");
    printf("Integer value: %d\n", data.intValue);  // May show garbage value
    printf("Float value: %.2f\n", data.floatValue); // May show garbage value

    return 0;
}
```

OUTPUT:

```
Output

Integer value: 42
Float value: 3.14
Character value: A

After storing a character, previous values are overwritten.
Integer value: 1078523201
Float value: 3.14


=== Code Execution Successful ===
```

3. Use typedef to create an alias for a structure representing a complex number and perform addition of two complex numbers.

CODE:

```
#include <stdio.h>

// Define a structure for complex numbers using typedef
typedef struct {
    float real;    // Real part
    float imag;    // Imaginary part
} Complex;
```

```
// Function to add two complex numbers

Complex addComplex(Complex c1, Complex c2) {
    Complex result;
    result.real = c1.real + c2.real;
    result.imag = c1.imag + c2.imag;
    return result;
}

int main() {
    Complex num1, num2, sum;

    // Input the first complex number
    printf("Enter the real and imaginary parts of the first complex number:\n");
    scanf("%f %f", &num1.real, &num1.imag);

    // Input the second complex number
    printf("Enter the real and imaginary parts of the second complex number:\n");
    scanf("%f %f", &num2.real, &num2.imag);

    // Perform addition
    sum = addComplex(num1, num2);

    // Display the result
    printf("The sum of the complex numbers is: %.2f + %.2fi\n", sum.real, sum.imag);

    return 0;
}
```

OUTPUT:

```
Output

Enter the real and imaginary parts of the first complex number:
3
2
Enter the real and imaginary parts of the second complex number
4
3
The sum of the complex numbers is: 7.00 + 5.00i


=== Code Execution Successful ===
```

PRACTICAL:8

1. Write a program to create a file and write user input into it.

CODE:

```
#include <stdio.h>

int main() {
   FILE *file;  // Declare a pointer to FILE
   char filename[100], userInput[500];

   // Prompt the user to enter the filename
   printf("Enter the filename to create: ");
   scanf("%s", filename);

   // Open the file for writing (creates the file if it doesn't exist)
   file = fopen(filename, "w");

   // Check if the file was successfully created/opened
   if (file == NULL) {
      printf("Error opening file!\n");
      return 1;  // Exit the program with an error code
   }

   // Prompt the user to enter some text to write into the file
   printf("Enter the text to write into the file (max 500 characters):\n");
   getchar();  // Clear the newline left by previous scanf
   fgets(userInput, sizeof(userInput), stdin);  // Read the user's input (including spaces)

   // Write the user's input into the file
   fprintf(file, "%s", userInput);

   // Close the file after writing
   fclose(file);
```

```
    printf("Data successfully written to the file: %s\n", filename);

    return 0;
}
```

OUTPUT:

```
Enter the filename to create: output.txt
Enter the text to write into the file (max 500 characters):
Hello, this is a test file.
```

```
Data successfully written to the file: output.txt
```

2. Write a program to read and display the contents of a file.

CODE:

```
#include <stdio.h>

int main() {
    FILE *file;          // Declare a pointer to FILE
    char filename[100];  // To store the filename
    char ch;             // To store each character read from the file

    // Prompt the user to enter the filename
    printf("Enter the filename to read: ");
    scanf("%s", filename);

    // Open the file in read mode
    file = fopen(filename, "r");

    // Check if the file was successfully opened
    if (file == NULL) {
        printf("Error opening the file %s\n", filename);
        return 1;  // Exit the program with an error code
    }
```

```
    // Display the contents of the file character by character
    printf("\nContents of the file %s:\n", filename);
    while ((ch = fgetc(file)) != EOF) {
        putchar(ch);  // Display each character from the file
    }

    // Close the file after reading
    fclose(file);

    return 0;
}
```

OUTPUT:

```
Enter the filename to read: example.txt
```

```
Contents of the file example.txt:
Hello, World!
```

3. Write a program to append text to an existing file.

CODE:

```
#include <stdio.h>

int main() {
    FILE *file;          // Declare a pointer to FILE
    char filename[100];   // To store the filename
    char userInput[500];  // To store the text input by the user

    // Prompt the user to enter the filename
    printf("Enter the filename to append text to: ");
    scanf("%s", filename);

    // Open the file in append mode (creates the file if it doesn't exist)
    file = fopen(filename, "a");
```

```
    // Check if the file was successfully opened
    if (file == NULL) {
        printf("Error opening the file %s\n", filename);
        return 1;  // Exit the program with an error code
    }

    // Prompt the user to enter the text to append
    printf("Enter the text to append to the file (max 500 characters):\n");
    getchar();  // Clear the newline left by previous scanf
    fgets(userInput, sizeof(userInput), stdin);  // Read user input, including spaces

    // Append the user's input to the file
    fprintf(file, "%s", userInput);

    // Close the file after appending
    fclose(file);

    printf("Text successfully appended to the file: %s\n", filename);

    return 0;
}
```

OUTPUT:

```
Enter the filename to append text to: example.txt
Enter the text to append to the file (max 500 characters):
This is the appended text.
```

```
Text successfully appended to the file: example.txt
```

PRACTICAL: 9

1. Write a program to search for a specific word in a file and display its occurrences.

CODE:

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

#define MAX_LINE_LENGTH 1000

int main() {
    FILE *file;            // Declare a pointer to FILE
    char filename[100];    // To store the filename
    char searchWord[100];  // To store the word to search for
    char line[MAX_LINE_LENGTH];  // To store each line read from the file
    int count = 0;         // To count occurrences of the word

    // Prompt the user to enter the filename
    printf("Enter the filename to search in: ");
    scanf("%s", filename);

    // Open the file in read mode
    file = fopen(filename, "r");

    // Check if the file was successfully opened
    if (file == NULL) {
        printf("Error opening the file %s\n", filename);
        return 1;  // Exit the program with an error code
    }

    // Prompt the user to enter the word to search for
    printf("Enter the word to search for: ");
    scanf("%s", searchWord);
```

```
    // Read the file line by line
    while (fgets(line, sizeof(line), file)) {
        // Search for the word in the current line
        char *pos = line;
        while ((pos = strstr(pos, searchWord)) != NULL) {
            count++;  // Increment count if the word is found
            pos++;    // Move to the next character after the found word
        }
    }
    // Close the file after reading
    fclose(file);
    // Display the result
    if (count > 0) {
        printf("The word '%s' occurred %d times in the file '%s'.\n", searchWord, count, filename);
    } else {
        printf("The word '%s' was not found in the file '%s'.\n", searchWord, filename);
    }
    return 0;
}
```

OUTPUT:

```
Enter the filename to search in: example.txt
Enter the word to search for: hello
```

```
The word 'hello' occurred 2 times in the file 'example.txt'.
```

2. Write a program to reverse the contents of a file and save it in another file.

CODE:

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

```
#define MAX_FILE_SIZE 10000

int main() {
    FILE *inputFile, *outputFile;  // Declare file pointers
    char inputFilename[100], outputFilename[100];  // Store filenames
    char fileContent[MAX_FILE_SIZE];  // To store the file contents

    // Prompt the user to enter the input filename
    printf("Enter the input filename: ");
    scanf("%s", inputFilename);

    // Open the input file in read mode
    inputFile = fopen(inputFilename, "r");

    // Check if the input file was successfully opened
    if (inputFile == NULL) {
        printf("Error opening the file %s\n", inputFilename);
        return 1;  // Exit the program with an error code
    }

    // Read the contents of the input file into a buffer
    size_t fileSize = fread(fileContent, sizeof(char), MAX_FILE_SIZE - 1, inputFile);
    fileContent[fileSize] = '\0';  // Null-terminate the string

    // Close the input file after reading
    fclose(inputFile);

    // Prompt the user to enter the output filename
    printf("Enter the output filename: ");
    scanf("%s", outputFilename);

    // Open the output file in write mode
    outputFile = fopen(outputFilename, "w");
```

```
    // Check if the output file was successfully opened
    if (outputFile == NULL) {
        printf("Error opening the file %s\n", outputFilename);
        return 1;  // Exit the program with an error code
    }

    // Reverse the contents of the file and write to the output file
    for (int i = fileSize - 1; i >= 0; i--) {
        fputc(fileContent[i], outputFile);
    }

    // Close the output file after writing
    fclose(outputFile);

    printf("The content has been reversed and saved to '%s'.\n", outputFilename);

    return 0;
}
```

OUTPUT:

```
Enter the input filename: input.txt
Enter the output filename: output.txt
```

```
The content has been reversed and saved to 'output.txt'.
```

```
 !dlroW olleH
```

3. Write a program to read a specific line from a file.

CODE:

```
#include <stdio.h>

#define MAX_LINE_LENGTH 1000
```

```
int main() {
    FILE *file;           // Declare a pointer to FILE
    char filename[100];    // To store the filename
    char line[MAX_LINE_LENGTH];  // To store each line from the file
    int lineNumber, currentLine = 1;

    // Prompt the user to enter the filename
    printf("Enter the filename to read from: ");
    scanf("%s", filename);

    // Open the file in read mode
    file = fopen(filename, "r");

    // Check if the file was successfully opened
    if (file == NULL) {
        printf("Error opening the file %s\n", filename);
        return 1;  // Exit the program with an error code
    }

    // Prompt the user to enter the line number to read
    printf("Enter the line number to read: ");
    scanf("%d", &lineNumber);

    // Read the file line by line until we reach the desired line
    while (fgets(line, sizeof(line), file)) {
        if (currentLine == lineNumber) {
            // If the current line number matches the desired line, display it
            printf("Line %d: %s", lineNumber, line);
            break;  // Exit the loop after printing the specific line
        }
        currentLine++;  // Increment the line counter
    }
```

```
    // If the loop ends without finding the line, it means the line number is invalid
    if (currentLine <= lineNumber) {
        printf("The file does not have %d lines.\n", lineNumber);
    }

    // Close the file after reading
    fclose(file);

    return 0;
}
```

OUTPUT:

```
Enter the filename to read from: example.txt
Enter the line number to read: 3
```

```
Line 3: This is the third line of the file.
```

```
The file does not have 3 lines.
```

PRACTICAL:10

1. Write a program to sort an array using the Bubble Sort algorithm.

CODE:

```
#include <stdio.h>

void bubbleSort(int arr[], int n) {
    int temp;
    // Traverse through all array elements
    for (int i = 0; i < n - 1; i++) {
        // Last i elements are already sorted, so we don't need to compare them
        for (int j = 0; j < n - i - 1; j++) {
            // Swap if the element found is greater than the next element
            if (arr[j] > arr[j + 1]) {
                // Swap elements
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

void printArray(int arr[], int size) {
    // Function to print the elements of the array
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {64, 25, 12, 22, 11};  // Example array to sort
```

```
int n = sizeof(arr) / sizeof(arr[0]);  // Calculate the number of elements

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

    // Calling the bubbleSort function to sort the array
    bubbleSort(arr, n);

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

    return 0;
}
```

OUTPUT:

```
Output

Original array:
64 25 12 22 11
Sorted array:
11 12 22 25 64


=== Code Execution Successful ===
```

2. Write a program to sort an array using the Insertion Sort algorithm.

CODE:

```
#include <stdio.h>

void insertionSort(int arr[], int n) {
    int key, j;
    // Traverse the array from the second element (index 1)
    for (int i = 1; i < n; i++) {
        key = arr[i];  // The element to be inserted
        j = i - 1;
```

```
        // Shift elements of arr[0..i-1], that are greater than key, to one position ahead
        // of their current position
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];  // Shift element to the right
            j = j - 1;
        }

        arr[j + 1] = key;  // Place the key in its correct position
    }
}

void printArray(int arr[], int size) {
    // Function to print the elements of the array
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {12, 11, 13, 5, 6};  // Example array to sort
    int n = sizeof(arr) / sizeof(arr[0]);  // Calculate the number of elements

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

    // Calling the insertionSort function to sort the array
    insertionSort(arr, n);

    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

OUTPUT:

```
Output

Original array:
12 11 13 5 6
Sorted array:
5 6 11 12 13


=== Code Execution Successful ===
```

3. Write a program to perform a linear search on an array.

CODE:

```
#include <stdio.h>

int linearSearch(int arr[], int n, int key) {
    // Traverse the array to search for the key
    for (int i = 0; i < n; i++) {
        if (arr[i] == key) {
            return i;  // Return the index if the element is found
        }
    }
    return -1;  // Return -1 if the element is not found
}

int main() {
    int arr[] = {64, 25, 12, 22, 11};  // Example array
    int n = sizeof(arr) / sizeof(arr[0]);  // Calculate number of elements
    int key;

    // Prompt user to enter the element to search for
    printf("Enter the element to search for: ");
    scanf("%d", &key);
```

```
    // Perform linear search
    int result = linearSearch(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;
}
```

OUTPUT:

```
Output

Enter the element to search for: 2
Element 2 not found in the array.


=== Code Execution Successful ===
```

4. Write a program to perform a binary search on a sorted array.

CODE:

```
#include <stdio.h>

int binarySearch(int arr[], int n, int key) {
    int left = 0, right = n - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;  // Find the middle index

        // Check if key is present at mid
        if (arr[mid] == key) {
            return mid;  // Return the index if key is found
```

```
        }

        // If key is greater, ignore the left half
        if (arr[mid] < key) {
            left = mid + 1;
        }
        // If key is smaller, ignore the right half
        else {
            right = mid - 1;
        }
    }

    return -1;  // Return -1 if the key is not found
}

int main() {
    int arr[] = {11, 12, 22, 25, 64};  // Sorted array
    int n = sizeof(arr) / sizeof(arr[0]);  // Calculate number of elements
    int key;

    // Prompt user to enter the element to search for
    printf("Enter the element to search for: ");
    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;
}
```

OUTPUT:

```
Output

Enter the element to search for: 11
Element 11 found at index 0.


=== Code Execution Successful ===
```

5. Write a program to sort an array using the Selection Sort algorithm.

CODE:

```
#include <stdio.h>

void selectionSort(int arr[], int n) {
    int minIndex, temp;

    // Traverse the array
    for (int i = 0; i < n - 1; i++) {
        minIndex = i;  // Assume the current index as the minimum

        // Find the minimum element in the remaining unsorted part of the array
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;  // Update the minimum element's index
            }
        }

        // Swap the found minimum element with the first element of the unsorted part
        if (minIndex != i) {
            temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
```

```
    }
}

void printArray(int arr[], int size) {
    // Function to print the elements of the array
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int arr[] = {64, 25, 12, 22, 11};  // Example array to sort
    int n = sizeof(arr) / sizeof(arr[0]);  // Calculate number of elements

    printf("Original array: \n");
    printArray(arr, n);
    // Calling the selectionSort function to sort the array
    selectionSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);

    return 0;
}
```

OUTPUT:

Output

```
Original array:
64 25 12 22 11
Sorted array:
11 12 22 25 64


=== Code Execution Successful ===
```