PROGRAMMING in C

Subject code : 3132

Department of Computer Engineering

UNIT-4

SYLLABUS

Program **: Diploma in Computer Engineering**	
Course Code **: 4152**	Course Title **: Programming in C**
Semester **: 3**	Credits : 3
Course Category **: Program Core**	
Periods per week **: 3 (L:3 T:0 P:0)**	Periods per semester : **45**

Course Objectives:

- Provide a thorough knowledge in Programming using C Language.
- Develop programming skills using arrays, pointers, structures and files to solve real world problems.
- Build the necessary foundation for system programming and other advanced programming courses.

Course Prerequisites:

Topic	Course Code	Course Name	Semester
Basic knowledge on problem solving and programming concepts.	3134	Problem Solving and Programming	2

COURSE OUTCOMES

Course outcomes

On the completion of the course student will be able to:

COn	Description	Duration (Hours)	Cogitative level
CO1	Make use of the basic programming concepts – sequential, conditional, looping structures and functions in C.	11	Applying
CO2	Make use of the concept of arrays to solve real world problems.	11	Applying
CO3	Develop programs using Pointers to solve problems more efficiently.	10	Applying
CO4	Construct user defined data types using structure, union and files.	11	Applying
	Series Test	2	

MODULE. 4

MODULE IV. OUTCOMES

Module outcomes

On the completion of the module student will be able to:

Module Outcomes	Description	Duration (Hours)	Cogitative level
CO4	Construct user defined data types using structure, union and files.		
M4.01	Explain the definition, declaration and processing of structure data type	1	Understanding
M4.02	Develop programs using structure to solve problems	2	Applying
M4.03	Illustrate the array of structure with examples	1	Understanding
M4.04	Illustrate passing of structure as parameters to a function.	1	Understanding
M4.05	Utilize pointers to process structure data type.	1	Applying
M4.06	Explain features of union data type, enumerations	3	Understanding
M4.07	Illustrate the use of file as data storage, input and output to programs.	1	Understanding
M4.08	Illustrate command line arguments	1	Understanding
	Series Test - II	1	

MODULE 4. CONTENTS

Structure – declaration, definition and initialization of structure variables, Accessing ofstructure elements – Array of structure – Structure and Pointer – Structure and Function –Union - enumerations.

File – Defining, opening, closing a file - input and output operations on sequential files - Command Line arguments.

1. STRUCTURE IN C

Structure in C

- Structure in c is a **user-defined data type**.
- Structure is a collection of different data types.
- That is it represent a **record**.
- A structure is a composition of variables, possibly of different data types, grouped together under a single name.
- Each variable within the structure is called a '**member**'.
- The name given to the structure is called a '**structure tag**'.

STRUCTURE IN C

Defining a Structure

- We can define a structure in C by using **struct** keyword along with the **structure_name.**
- Structure can be defined locally or globally

Syntax

```
struct   structure_name
{
datatype   member1;
datatype   member2;
----------------
};
```

Example-1

```
struct  Employee
 {
   int id;
   char Name[10];
   float salary;
  } ;
```

Example-2

```
struct  Book
 {
   int ISBN;
   char title[30];
   float price;
   int pages;
  } ;
```

STRUCTURE IN C

Declaring a Structure

We can declare a variable for the structure so that we can access the member of the structure easily.

There are two ways to declare structure variable:

1. By **struct** keyword within main() function
2. By declaring a variable at the **time of defining the structure**.

1st way:

Syntax

```
struct   structure_name
{
datatype   member1;
datatype   member2;
-------------------------
};
// in main function
struct  structure_name   variable_name;
```

Example

```
struct  Employee
 {
   int id;
   char Name[10];
   float salary;
  } ;
// in main function
 struct  Employee  e1, e2;
```

STRUCTURE IN C

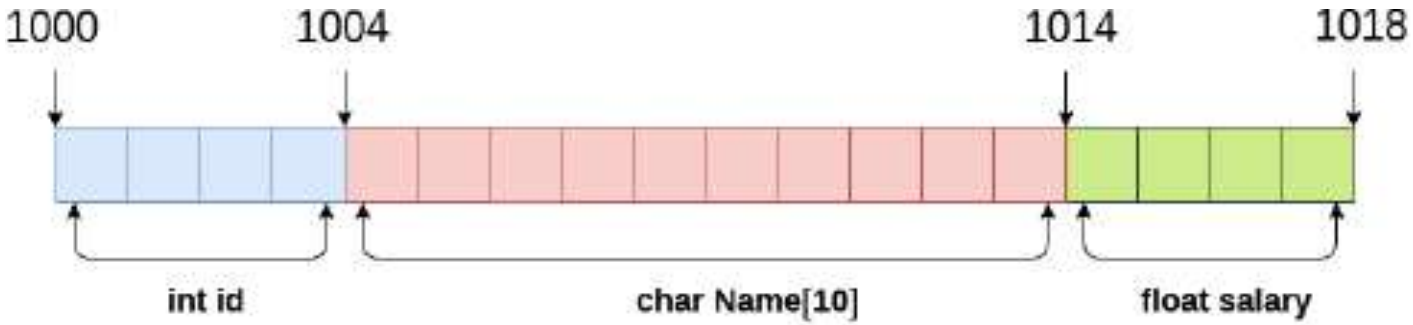
2nd way:

Syntax

```
struct  structure_name
{
datatype   member1;
datatype   member2;
-------------------------
} variable_names;
```

Example

```
struct  Employee
 {
   int id;
   char Name[10];
   float salary;
  } e1, e2;
```




```
struct Employee
{
    int id;
    char Name[10];
    float salary;
} emp;
```

sizeof (emp) = 4 + 10 + 4 = 18 bytes

where;
sizeof (int) = 4 byte
sizeof (char) = 1 byte
sizeof (float) = 4 byte

2. ACCESSING STRUCTURE MEMBERS

Accessing members of a Structure

There are two ways to access structure members:

1. **Using . Operator** (**member access** or **dot** operator)

- To access any member of a structure, we use the **member access operator (.)**
- The member access operator is coded as a period between the structure variable name and the structure member that we wish to access.

2. Using -> operator (**arrow** or **structure pointer** operator)

- The members of the structure can be accessed using a special operator called as an arrow operator (->)
- The structure pointer tells the address of a structure in memory by pointing to the structure variable
- Here we use **pointer to access the structure**

ACCESSING STRUCTURE MEMBERS

Example for accessing members through member access operator

```
#include<stdio.h>
#include <string.h>
struct  employee
{
  int  id;
  char name[50];
};//declaring e1 variable for structure

// main function
void main()
{
//store first employee information
struct employee e1;
e1.id=101;
strcpy(e1.name, “Akshay Kumar");
//printing first employee information
printf( "employee 1 id : %d\n", e1.id);
printf( "employee 1 name : %s\n", e1.name);
}
```

Output:

employee 1 id : **101**

employee 1 name : **Akshay Kumar**

ACCESSING STRUCTURE MEMBERS

Example for accessing members through structure pointer operator

```
#include<stdio.h>
#include <string.h>
struct  employee
{
  int  id;
  char name[50];
}e1; //declaring e1 variable for structure

// main function
int main()
{
// creating a structure pointer
struct  employee  *eptr;
eptr=&e1;    // assign address of e1 to pointer
//store first employee information
eptr->id=101;
strcpy(eptr->name, “Akshay Kumar");
//printing first employee information
printf( "employee 1 id : %d\n", eptr->id);
printf( "employee 1 name : %s\n", eptr->name);
return 0;
}
```

Output:

employee 1 id : **101**

employee 1 name : **Akshay Kumar**

Enter car details using structure-model, year price

```
#include <stdio.h>

struct Car {
    char model[30];
    int year;
    float price;
};
int main() {
    struct Car c;
    printf("Enter car model: ");
    scanf("%s", c.model);
    printf("Enter year: ");
    scanf("%d", &c.year);
    printf("Enter price: ");
    scanf("%f", &c.price);
    printf("\nCar Details:\n");
    printf("Model: %s\n", c.model);
    printf("Year: %d\n", c.year);
    printf("Price: $%.2f\n", c.price);

    return 0;
}
```

7. UNION IN C

Union in C

- Union is a user-defined data type
- Structures allocate enough space to store all their members, **whereas unions can only hold one member value at a time**.

Syntax for defining a union

Syntax

```
union  union_name
{
datatype   member1;
datatype   member2;
------------------------
};
// in main function
union  union_name   variable_name;
```

Example

```
union employee
 {
   int id;
   char Name[10];
   float salary;
  } ;
// in main function
  union employee  e1, e2;
```

UNION IN C

Example

```
#include <stdio.h>
union Job
{
  float  salary;
  int  workerNo;
} j;

int main()
{
 j.salary = 12.3;
 printf("Salary = %f\n", j.salary);
 j.workerNo = 100;
 printf("Number of workers =
 %d", j.workerNo);

return 0;
 }
```

Output

Salary = 12.3
Number of workers = 100

8. STRUCTURE vs UNION IN C

Points	Structure	Union
Memory allocation	Each member of a structure is allocated separate memory space.	All Members share the same space in memory.
Size of structure	Total size of structure variable is the sum of sizes of individual members in the structure.	The amount of memory required is the same as its largest member.
Member access	All structure members can be accessed at a time.	Only one member can be accessed at a time.
Initialization	Members of the structure can be initialized.	Only the first member of the union can be initialized.
Altering	Altering the value of one member will not affect other members.	Altering the value of one member will change the value of other members.

Assignment-3

Write a c programming to display Book Details using structure and union

3. "typedef" keyword in C

"typedef" keyword in C

- We use the typedef keyword to create an **alias** (alternate) name for **data types**.
- It is commonly used with structures to simplify the syntax of declaring variables.

Syntax

```
typedef existing_name alias_name;
```

Example:

```
typedef long double ld;
```

```
typedef int Kilometer;
Kilometer distance = 100
```

```
// C program to implement typedef
#include <stdio.h>
int main()
{
// defining an alias using typedef
  typedef long double ld;
  ld  var = 20.001;
  printf("%Lf", var);
  return 0;
}
```

"typedef" keyword in C

For example, let us look at the following code using typedef with structure:

```
struct Distance
{
   int feet;
   float inch;
 };

int main()
{
  struct Distance  d1, d2;
}
```

We can use typedef to write an equivalent code with a simplified syntax:

```
typedef struct Distance
{
int feet;
float inch;
} distances;

int main()
{
distances d1,  d2;
}
```

4. Initialize a structue

Initialize structure members

- Structure variables can be initialized at the same time they are declared, just like normal variables and arrays.

General form: **structure** **structure_name** **structure_variable** = { value1, value2, ... };

Example

```
struct Book
{
char name[80];
int price;
int publishYear;
}b1={“The Discovery”, 500, 2023};

int main()
{
 struct Book b2 = { “Oliver Twist" , 400, 1996};
}
```

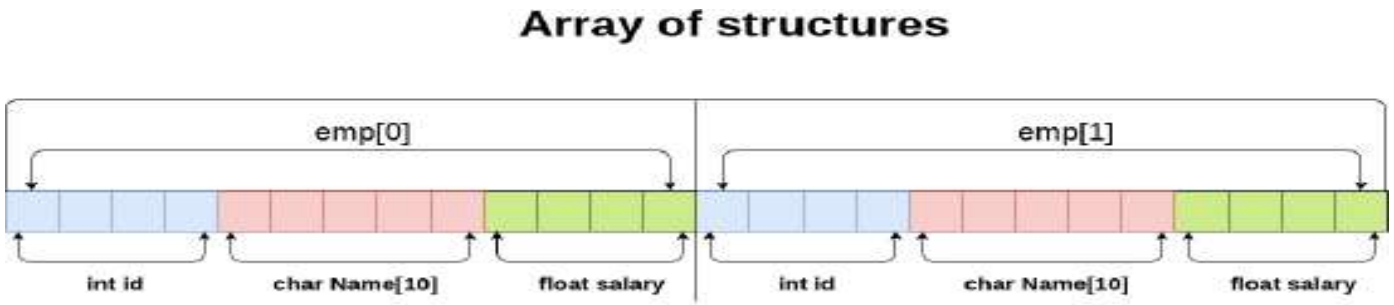
5. ARRAY OF STRUCTURE IN C

Array of Structures in C

- An array of structres in C can be defined as the **collection of multiple structures variables** where each variable contains information about different entities.
- The array of structures in C are used to store information about multiple entities of different data types.
- The array of structures is also known as the **collection of structures**.

Declaring array of structure

```
structure  structure_name  structure_variable [size];
```




```
struct employee
{
    int id;
    char name[5];
    float salary;
};
struct employee emp[2];
```

sizeof (emp) = 4 + 5 + 4 = 13 bytes

sizeof (emp[2]) = 26 bytes

ARRAY OF STRUCTURE

Example of an array of structures that stores information of 5 students and prints it. :

```
#include<stdio.h>
#include <string.h>
struct student
{
 int rollno;
 char name[10];
};

int main()
{
 int i;
 struct student st[5];
 printf("Enter Records of 5 students");
 for(i=0;i<5;i++)
 {
   printf("\nEnter  Rollno:");
   scanf("%d",&st[i].rollno);
   printf("\nEnter   Name:");
   scanf("%s",&st[i].name);
}
```

```
 printf("\nStudent Information List:");
 for(i=0;i<5;i++)
 {
   printf("\nRollno:%d,    st[i].rollno);
   printf("\nName:%s",    st[i].name);
   printf("\n----------------------------");
 }
 return 0;
}
```

Output:

```
Enter  Records of 5 students
Enter Rollno:1
Enter Name:Sonoo
Enter Rollno:2
Enter Name:Ratan
Enter Rollno:3
Enter Name:Vimal
Enter Rollno:4
Enter Name:James
Enter Rollno:5
Enter Name:Sarfraz
```

```
Student Information List:
Rollno:1
Name:Sonoo
-----------------------------------
Rollno:2
Name:Ratan
-----------------------------
Rollno:3
Name:Vimal
-----------------------------
Rollno:4
 Name:James
-----------------------------
Rollno:5
Name:Sarfraz
-----------------------------
```

Example of an array of structures that stores information of 5 employee and prints it. :

```c
#include <stdio.h>
#include <string.h> // Required for strcpy

// Define the structure for employee
details
struct Employee {
int id;
char name[50];
float salary;
};

int main() {
// Declare an array of 5 Employee
structures
struct Employee employees[5];
int i;
printf("Enter details for 5 employees:\n");
for (i = 0; i < 5; i++) {
printf("\nEmployee %d:\n", i + 1);
printf("Enter ID: ");
scanf("%d", &employees[i].id);
printf("Enter Name: ");
scanf("%s",
employees[i].name);
printf("Enter Salary: ");
scanf("%f",
&employees[i].salary);
}

// Print details of 5
employees
printf("\n--- Employee
Details ---\n");
for (i = 0; i < 5; i++) {
printf("\nEmployee %d:\n", i
+ 1);
printf("ID: %d\n",
employees[i].id);
printf("Name: %s\n",
employees[i].name);
printf("Salary: %.2f\n",
employees[i].salary);
}

return 0;
}
```

6. FUNCTION & STRUCTURE

Passing structure to function

- A structure can be passed to any function from main function or from any sub function like any other varibale.
- Structure definition will be available within the function only.
- It won't be available to other functions unless it is passed to those functions by value or by address(reference).

Pass struct variables as arguments to a function.

```
#include <stdio.h>
struct student
{
  char name[50];
  int age;
};
```

```
//function definition

void display(struct student s)
 {
   printf("\nDisplaying information\n");
   printf("Name: %s", s.name);
   printf("\nAge: %d", s.age);
  }
```

```
int main()
{
 struct student s1;
 printf("Enter name: ");
 scanf("%s", s1.name);
 printf("Enter age: ");
 scanf("%d", &s1.age);
 display(s1); // passing struct as an argument

 return 0;
 }
```

Write a program to display student name roll number and mark using function

```
struct Student
{
    char name[50];
    int roll_number;
    float marks;
};
void displayStudent(struct Student s)
{
    printf("Name: %s\n", s.name);
    printf("Roll Number: %d\n", s.roll_number);
    printf("Marks: %.2f\n", s.marks);
}
```

```
void displayStudentPtr(struct Student *s_ptr)
{
    printf("Name: %s\n", s_ptr->name);
    printf("Roll Number: %d\n", s_ptr->roll_number);
    printf("Marks: %.2f\n", s_ptr->marks);
}
int main() {
    struct Student s1;
    strcpy(s1.name, "Alice");
    s1.roll_number = 101;
    s1.marks = 85.5;
    displayStudent(s1); // Pass the structure by value
    displayStudentPtr(&s1); // Pass the address of the structure
    return 0;
}
```

9. “enum” IN C

Enumeration in C

- Enumeration or “enum” is a user-defined data type
- An enumeration type (also called enum) is a data type that consists of integral constants.

Syntax for defining enum

To define enums, the **enum** keyword is used

```
enum calculate { SUM, DIFFERENCE, PRODUCT,
QUOTIENT };
```

```
enum enumeration_name {value1, value2, .....};
```

```
enum item{ PRODUCT, SERVICE };
```

- a user defined data type that contains a set of named integer constants.
- It is used to assign meaningful names to integer values, which makes a program easy to read and maintain.
- **enum names must be unique in their scope.**
- By default, first name is assigned **0,** next is assigned **1** and the subsequent ones are **incremented by 1.**

Example

```
#include <stdio.h>
int main()
{
// defining a enum week
enum week {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday,
Saturday};
// creating today variable of enum week type
enum week today;
today = Wednesday;
printf("Day %d",today+1); return 0;
}
```

Output

Day 4

10. FILES IN C

Files in C

When working with files, you need to **declare a pointer of type file**.

This declaration is needed for communication between the file and the program.

```
FILE *fptr;
```

File Operations are:

- ➢ Creating a new file
- ➢ Opening an existing file
- ➢ Closing a file

FILES IN C

Opening a File – for creation and editing

For reading from and writing information to a file pointer; we should use the **fopen()** function to open a file.

```
FILE *fileptr = fopen("filename","mode");
```

fopen() function defined in the **stdio.h**

Example:

```
FILE *p1 = fopen("E:\\cprogram\\newprogram.txt","w");
FILE *p2 = fopen("E:\\cprogram\\oldprogram.bin","rb");
```

Some common File Modes	
Mode	**Explanation**
r	Open for read
w	Open for write
a	Open for append
rb, wb, ab	Open for reading, writing and appending in binary mode
r+	Open for both reading and writing.
w+	Open for both reading and writing.

FILES IN C

Closing a File

fclose() function to close a file by passing file pointer as the parameter.

```
fclose(fileptr );
```

Example:

```
FILE *p1 = fopen("E:\\cprogram\\newprogram.txt","w");
fclose(p1);
```

FILES IN C

Read & Write a File

fprintf(): function to write to file

Example:

```
fprintf(fptr, ”%d”, &mark);
```

Assuming we have already opened a file using fptr and mark variable value is available.

---

fscanf(): function to read from a file

Example:

```
fscanf(fptr, ”%d”, &mark);
```

Integer value is read from the file and available with mark varibale.

FILES IN C

getc() & **putc()**:- Only single character is read / write

Example:

```
char ch;
ch=getc(fptr);
---
char ch = ‘a’;
putc(ch, fptr);
----------
```

getw() & **putw()**:- Only handle integer values, read/ write

Example:

```
int n;
n=getw(fptr);
---
int n = 100;
putw(n, fptr);
```

- fputs():
  - **Purpose:** Writes a string to a file.
  - **Syntax:** int fputs(const char *str, FILE *stream);
- fgets():
  - **Purpose:** Reads a string from a file.
  - **Syntax:** char *fgets(char *str, int n, FILE *stream);
  - **Reads:** Up to n-1 characters or until a newline character or end-of-file is encountered.
- fseek():
  - **Purpose:** Sets the file position indicator for the stream. Allows moving within the file.
  - **Syntax:** int fseek(FILE *stream, long int offset, int origin);
  - origin options: SEEK_SET (beginning), SEEK_CUR (current position), SEEK_END (end of file)

- rewind():
  - **Purpose:** Sets the file position indicator to the beginning of the file.
  - **Syntax:** void rewind(FILE *stream);
- feof():
  - **Purpose:** Checks if the end-of-file indicator is set for a stream.
  - **Syntax:** int feof(FILE *stream);
  - **Returns:** Non-zero if end-of-file, 0 otherwise.

FILES IN C

Example for file writing

```
#include  <stdio.h>
#include <stdlib.h>
int main()
{
int num;
FILE *fptr;
fptr = fopen("C:\\sample.txt","w");
if(fptr == NULL)
{
printf("Error!");
exit(1);
}
printf("Enter num: ");
scanf("%d",&num);
fprintf(fptr,"%d",num);   // write num to file sample.txt
fclose(fptr);

return 0;
}
```

Write a C program to count the number of characters in the file

```
#include <stdio.h>
#include <stdlib.h> // For exit()

int main() {
FILE *fptr;
char filename[100];
char ch;
int count = 0;

printf("Enter the filename to count characters: ");
scanf("%s", filename);

// Open the file in read mode
fptr = fopen(filename, "r");
if (fptr == NULL) {
printf("Error: Could not open file %s\n", filename);
exit(EXIT_FAILURE); // Terminate the program
}

while ((ch = fgetc(fptr)) != EOF) {
count++; // Increment the character count
}

// Close the file
fclose(fptr);

printf("The file '%s' contains %d characters.\n",
filename, count);

return 0;
}
```

Command-line arguments

Command-line arguments in C provide a mechanism to pass information to a program at the time of its execution via the command-line interface.
This allows for dynamic input and control over program behavior without modifying the source code.
Command-line arguments are typically handled by the `main()` function, which can be defined with two parameters:

```
int main(int argc, char *argv[]) {
// Program logic
return 0;
}
```

•argc (Argument Count):
This integer variable stores the total number of command-line arguments, including the program's name itself. argc will always be at least 1, as argv[0] always holds the program's name.
•argv (Argument Vector):
This is an array of character pointers (char *argv[] or char **argv). Each element in this array points to a string representing a command-line argument.
•argv[0] always contains the name of the executable program.
•argv[1] to argv[argc - 1] contain the actual arguments provided by the user.
•argv[argc] is a null pointer, marking the end of the argument list.

sum and difference using command-line arguments

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc,  char *argv[]) {
// Check if two arguments are passed
if (argc != 3)
{
    printf("Usage: %s <number1> <number2>\n", argv[0]);
    return 1;
}
int num1 = atoi(argv[1]);
int num2 = atoi(argv[2]);
int sum = num1 + num2;
int difference = num1 - num2;
printf("Sum: %d\n", sum);
printf("Difference: %d\n", difference);
return 0;
}
```

program

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    if (argc != 4) {
        printf("Usage: %s string1 string2 string3\n", argv[0]);
        return 1;
    }

    printf("String 1: %s\n", argv[1]);
    printf("String 2: %s\n", argv[2]);
    printf("String 3: %s\n", argv[3]);

    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h> // Required for atoi()

int main(int argc, char *argv[]) {
if (argc != 3)
 {
      printf(" %s ", argv[0]);
     return 1;                          // Indicate an error
}

// Convert the string arguments to integers
int num1 = atoi(argv[1]);              // argv[1] is the first number
int num2 = atoi(argv[2]);                  // argv[2] is the second number

// Calculate the sum
int sum = num1 + num2;

// Print the result
printf("The sum of %d a %d is: %d\n", num1, num2, sum);

return 0; // Indicate successful execution
}
```

Write a program to do the following
a) Declare a single pointer to represent a numeric single dimensional array
b) Allocate space and store the first address to pointer
c) Read 10 numbers to array (use the above pointer)
d) Write elements from array (use the above pointer)

```c
#include <stdio.h>

int main() {
    int arr[10];      // a) Declare a static array of 10 integers
    int *ptr = arr;   // b) Pointer points to the first element of the array

    // c) Read 10 numbers using the pointer
    printf("Enter 10 numbers:\n");
    for (int i = 0; i < 10; i++) {
        scanf("%d", ptr + i);  // or &arr[i]
    }

    // d) Write (print) elements from the array using the pointer
    printf("The entered numbers are:\n");
    for (int i = 0; i < 10; i++) {
        printf("%d ", *(ptr + i));
    }
    printf("\n");

    return 0;
}
```

Write a program for the following
a) Declare a character array of pointers containing 10 addresses
b) Allocate spaces for storing 20 names
c) Read 20 names and store in the array
d) Write 20 names from the array

```c
#include <stdio.h>

int main() {
    char names[20][50];     // b) Static allocation: space for 20 names, max length 49 + '\0'
    char *ptr[20];          // a) Array of 20 pointers to char

    // Point each pointer to each name string in 'names'
    for (int i = 0; i < 20; i++) {
        ptr[i] = names[i];
    }

    // c) Read 20 names
    printf("Enter 20 names:\n");
    for (int i = 0; i < 20; i++) {
        scanf("%49s", ptr[i]);   // read string safely
    }

    // d) Write 20 names
    printf("The entered names are:\n");
    for (int i = 0; i < 20; i++) {
        printf("%s\n", ptr[i]);
    }
```