


## Solution with scheme-Model Answer

Prof.Lynsha Helena Pratheebea HP/Prof.Kavyashree/Prof.Reshma										
Internal Assessment Test II – June 2025										
Sub	Principles of Programming Using C					Sub code	BPOPS203	Branch	CSE, CSDS	
Date	20.06.2025	Duration	90 mins	Max Marks	50	Sem /Sec	II Sem P-Cycle (I,J,K,L)		OBE	
Answer any FIVE FULL Questions								MARKS	C O	RBT
1.	Explain the different storage classes available in C programming. Provide the purpose, scope, and lifetime of each, along with suitable example programs.						[10]	CO3	L2	
2.	Define Recursion. Explain its types with example and demonstrate a program to find the factorial of ‘n’ numbers with a sample output.						[10]	CO3	L3	
3.	Define String. Explain any four string manipulating functions with example.						[10]	CO4	L2	
4.	Explain the concept of pointers in C with proper syntax. Describe any five types of pointers with examples. Write a C program using pointers to calculate the sum, mean, and standard deviation of elements in an array.						[10]	CO4	L2	
5.	Mentions the application of pointers and write a C Program to swap two numbers using call by reference.						[10]	CO4	L2	
6.	Illustrate Structure declaration and how structure members are accessed with an example. Also implement a structure to read, write and compute average marks and the students scoring above and below average of class N students.						[10]	CO5	L3	
7.	Define an enumeration for the days of the week in C. Write a program to prompt the user to enter a number (0 to 6), map it to the corresponding day using the enum, and display whether it is a weekday or weekend. Use a switch-case to handle the logic."						[10]	CO5	L3	
8.	Write a C program that demonstrates the use of file handling functions such as fopen(), fclose(), fscanf(), and fprintf(). Explain each function with its syntax and purpose.						[10]	CO5	L3	

- 1) Explain the different storage classes available in C programming. Provide the purpose, scope, and lifetime of each, along with suitable example programs.
- Each storage class with purpose[1m]+ example[1m]+output[0.5m] [2.5\*4m]**

Different storage classes available in C programming:

1. Auto
2. Register
3. Static
4. Extern

Auto storage class:

The auto keyword is applied to all local variables automatically. It is the default storage class that is why it is known as automatic variable. It's default value is Garbage Value.

Example :-

```
void main()
{
int b = 5;
auto int a = 5; // Same as above . Auto keyword is default keyword.
printf(" %d, %d ", a,b);
}
```

Output :-

5 , 5

Static storage class:

static variables are initialised only once in a lifetime of program. That means it gets initialised only once when function is called . It's default value is zero.

Example

```
#include <stdio.h>
void function()
{
static int a = 0; /* This statement gets initialised only once then it is ignored. */
int b = 0;
a++;
b++;
printf("\na = %d , \nb = %d",a,b);
}
```

Output

a = 1 b = 1

a = 2 b = 1

a = 3 b = 1

```
int main ()
{
function ();
function ();
function ();
return 0;
}
```

Register storage class:

register variables are mostly used in places where we need fast execution .

Because it is stored in register and registers are always fast than RAM. It is mostly used in incrementing the counter or in loops.

```
register int counter=0;
```

```
#include <stdio.h>
void main()
{
int b = 5;
register int a = 5;
printf("%d%d",a,b);
```

```
}
Output
5 5
```

### extern storage class

The extern storage class is used to give a reference of a global variable that is visible to ALL the program files. When you use 'extern', the variable cannot be initialized however, it points the variable name at a storage location that has been previously defined.

The extern modifier is most commonly used when there are two or more files sharing the same global variables or functions as explained below.

#### **First File: main.c**

```
#include <stdio.h>
int count ;
extern void write_extern();
main() {
count = 5;
write_extern();
}
```

#### **Second File: support.c**

```
#include <stdio.h>

extern int count;

void write_extern(void) {
printf("count is %d\n",
count);
}
```

Here, extern is being used to declare count in the second file, where as it has its definition in the first file, main.c.

2)

Define Recursion. Explain its types with example and demonstrate a program to find the factorial of 'n' numbers with a sample output.

#### **Recursion Definition : 1m + Types[2m+2m] + program [5m]**

##### **Definition :**

A recursive function is a function that calls itself to solve a smaller version of its task until a final call is made which does not require a call to itself.

Program explanation:

Fibonacci Series in C: 0, 1, 1, 2, 3, 5, 8, 13, 21 etc

The Fibonacci series is a sequence of numbers in which each number is the sum of the two preceding ones. The sequence typically starts with 0 and 1, and the subsequent numbers are generated by adding the two previous numbers. The first few numbers in the Fibonacci sequence are:  
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

In general, the Fibonacci sequence is defined by the recurrence relation:

- ☐  $F(0) = 0$
- ☐  $F(1) = 1$
- ☐  $F(n) = F(n-1) + F(n-2)$  for  $n > 1$

Types of Recursion:

**Direct Recursion:** This is the most straightforward type, where a function calls itself directly within its own body.

```
void direct_recursive_function(int n) {  
if (n == 0) return;  
direct_recursive_function(n - 1); // Direct call to itself  
}
```

**Indirect Recursion:** This occurs when a function calls another function, which in turn calls the first function (or a chain of functions eventually leading back to the first).

```
void function_A(int n);  
void function_B(int n);  
  
void function_A(int n) {  
if (n == 0) return;  
function_B(n - 1); // Calls function_B  
}  
  
void function_B(int n) {  
if (n == 0) return;  
function_A(n - 1); // Calls function_A, creating indirect recursion  
}
```

### Program

```
#include<stdio.h>  
int fib(int)  
int main()  
{  
int n, i = 0, res;  
printf("Enter the number of terms\n")  
scanf("%d" &n);  
printf("Fibonacci series\n")  
for ( i= 0 ; i<n ; i++ )  
{  
res=fib(i);  
printf("%d\n",res)  
}  
return 0;  
}  
  
int fib(int n)  
{  
if ( n == 0 )  
return 0;  
else if ( n == 1 )  
return 1;
```

```
else
return ( Fibonacci(n-1) + Fibonacci(n-2) );
}
```

Output

Enter the number of terms :5

Fibonacci series:0, 1, 1, 2, 3, 5

3) Define String. Explain any four string manipulating functions with example.

**Definition**[1] + [2\*4m]

A **string** is a sequence of characters or a data structure used to represent text.

```
char str[] = "string_value";
```

1. `strlen()`

**Purpose:** Returns the length of a string

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

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

```
int main() {
```

```
    char str[] = "Hello, World!";
```

```
    printf("Length of the string: %lu\n", strlen(str));
```

```
    return 0;
```

```
}
```

2. `strcpy()`

**Purpose:** Copies the contents of one string to another.

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

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

```
int main() {
```

```
    char src[] = "Hello, World!";
```

```
    char dest[50];
```

```
    strcpy(dest, src);
```

```
    printf("Destination string: %s\n", dest);
```

```
    return 0;
```

```
}
```

3. `strcat()`

**Purpose:** Concatenates (appends) one string to the end of another.

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

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

```
int main() {
```

```
char str1[50] = "Hello, ";
char str2[] = "World!";

strcat(str1, str2); // Append str2 to str1
printf("Concatenated string: %s\n", str1); // Output: Hello, World!

return 0;
}
```

#### 4. strcmp()

**Purpose:** Compares two strings lexicographically. Returns 0 if the strings are equal, a positive value if the first string is greater, and a negative value if the second string is greater.

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

int main() {
    char str1[] = "apple";
    char str2[] = "banana";

    int result = strcmp(str1, str2);

    if (result == 0) {
        printf("Strings are equal.\n");
    } else if (result < 0) {
        printf("%s is less than %s.\n", str1, str2);
    } else {
        printf("%s is greater than %s.\n", str1, str2);
    }
    return 0;
}
```

- 4) Explain the concept of pointers in C with proper syntax. Describe any five types of pointers with examples. Write a C program using pointers to calculate the sum, mean, and standard deviation of elements in an array.

Definition[1]+syntax[1]+5type[2]+program[5]

A pointer in C is a variable that stores the memory address of another variable.

Declaration of Pointer Variables:

The declaration of a pointer variable in C consists of specifying the data type of the variable it points to, followed by an asterisk (\*) and the name of the pointer variable.

Syntax:

datatype \*pointer\_name;

Types of pointer:

1. Integer pointers:

Pointers that point to integer value.

Syntax: int \*ptr;

2. Null pointer: Do not point to any valid memory location.

Datat type \*ptr=NULL

### 3. Structure pointer:

Pointer pointing to structure type

```
Struct structure_name *ptr;
```

### 4. void pointer:

A pointer that can point to any data type

```
void *ptr;
```

5. Function pointer: A function pointer is a variable that stores the address of a function, allowing you to call the function indirectly.

```
return_type (*pointer_name)(parameter_types);
```

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

```
#include <math.h>
```

```
int main()
```

```
{
```

```
int i, n;
```

```
float a[10], sum, mean, var, sd;
```

```
float *p;
```

```
printf("Enter Number of elements:");
```

```
scanf("%d", &n);
```

```
printf("Enter %d numbers :", n);
```

```
p = a;
```

```
for (i=0; i<n; i++)
```

```
{
```

```
scanf("%f", p);
```

```
p++;
```

```
}
```

```
sum = mean = var = sd = 0.0;
```

```
p = a;
```

```
for(i=0; i<n; i++)
```

```
{
```

```
sum = sum + (*p);
```

```
p++;
```

```
}
```

```
mean = sum / (float) n;
```

```
// Compute Variance
```

```
p = a;
```

```
for(i=0; i<n; i++)
```

```
{
```

```
var = var + pow( (*p - mean), 2);
```

```
p++;
```

```
}
```

```
var = var / (float) n;
```

```
sd = sqrt(var);
```

```
printf("Sum = %f\n", sum);
```

```
printf("Mean = %f\n", mean);
```

```
printf("Standard Deviation = %f\n", sd);
```

```
return 0;
```

```
}
```

```
/*
```

Sample Output 1:

```
$ cc prog11.c -lm
```

```
$ ./a.out
```

Enter Number of elements:5  
Enter 5 numbers :1  
2  
3  
4  
5  
Sum = 15.000000  
Mean = 3.000000  
Standard Deviation = 1.414214

- 5) Mentions the application of pointers and write a C Program to swap two numbers using call by reference.

Application[5]+program[5]

- Arrays and Strings:

Pointers can be used to traverse and manipulate arrays and strings more efficiently.

- Function Pointers:

Pointers to functions allow you to create arrays of functions, pass functions as arguments to other functions, or return functions from functions.

- Structures:

Pointers can be used to work with structures more efficiently, especially when dealing with large structures.

- Efficient Parameter Passing:

Passing pointers to functions instead of passing large data structures can be more efficient in terms of both time and space

- Dynamic Data Structures:

Pointers are crucial in implementing dynamic data structures such as trees and graphs.

Call by Reference: Both the actual and formal parameters refer to the same locations, so any changes made inside the function are actually reflected in actual parameters of the caller.

Example:

```
#include<stdio.h>
void swapx(int*,int*)
int main()
{
int a = 10, b = 20;
swapx(&a,&b);
printf("a=%d b=%d\n", a, b);
return 0;
}
void swapx(int*x,int*y)
{
int t = *x;
*x = *y;
*y = t;
printf("x=%d y=%d\n", *x, *y);
}
```

**Sample output:** x=20 y=10

a=10 b=20

Illustrate Structure declaration and how structure members are accessed with an example. Also implement a structure to read, write and compute average marks and the students scoring above and below average of class N students.

6) Structure explanation[5]+program[5]

Structure: - Structure is a collection of one or more variables of same or different data types grouped to gather under a single name for easy handling.

Declaration of a Structure :- A structure is declared using the keyword struct followed by structure name and variables are declared within a structure.

The structure is usually declared before the main( ) function.

Syntax : -

```
struct structure_name
{
};
```

Example :-

```
datatype member 1;
datatype member 2;
datatype member 3;
.....
.....
datatype member n;
```

```
struct employee
{
int emp_no;
char name[20];
int age;
float emp_sal;
};
```

Initialization of structure :- A structure initialization is done after the declaration of the Structure.

Syntax :-

```
struct structure_name structure_variable _name;
```

Example :-

```
struct employee emp1;
```

Accessing structure members :- A structure uses a .(dot) [Member Access Operator] to access any member of the structure.

Example :-

```
emp1.emp_no =12345;
```

Initialization of structure variables :-

```
struct employee
{
int emp_no;
```

```
char name[20];
int age;
float emp_sal;
} emp1={65421, "Hari",29,25000.00};
```

The order of values enclosed in the braces must match the order of members in the structure definition.

Program:

```
struct student
```

```
{
    int id;
    char name[20];
    float sub[6];
    float avg;
};
```

```
int main()
```

```
{
    struct student s[20];
    float sum=0;
    int i,j,n;
    printf("Enter the number of records :");
    scanf("%d",&n);
    printf("Enter %d student details...\n",n);
    for(i=0;i<n;i++)
    {
        printf("\n\nEnter student ID, name :");
        scanf("%d%s",&s[i].id, s[i].name);
        printf("Enter 6 subject marks :");
```

```
for (j=0;j<6;j++)
{
    scanf("%f", &s[i].sub[j]);
}
}
```

```
for(i=0;i<n;i++)
{
    sum=0;
    for (j=0;j<6;j++)
    {
        sum = sum + s[i].sub[j];
    }
    s[i].avg = sum / 6;
}
printf("Students scoring above the average marks....\n");
printf("\n\nID\tName\tAverage\n\n");
```

```
for(i=0;i<n;i++)
{
    if(s[i].avg>=35.0)
        printf("%d\t%s\t%f\n",s[i].id,s[i].name,s[i].avg);
}
printf("\n\nStudents scoring below the average marks....\n");
printf("\n\nID\tName\tAverage\n\n");
for(i=0;i<n;i++)
```

```

{
    if(s[i].avg<35.0)
        printf("%d\t%s\t%f\n",s[i].id,s[i].name,s[i].avg);
    }
    return 0;

}

```

7

. Define an enumeration for the days of the week in C. Write a program to prompt the user to enter a number (0 to 6), map it to the corresponding day using the enum, and display whether it is a weekday or weekend. Use a switch-case to handle the logic. [10 Marks]

Definition [2 marks]

Enumeration (enum) is a user-defined data type in C that consists of integral constants. It allows you to assign names to integer values, making your program more readable and manageable.

Syntax: enum enum\_name { value1, value2, value3, ... };

Program [8 Marks]

```

#include <stdio.h>
enum Day { Sunday = 0, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };
int main() {
    int dayNumber;
    enum Day day;
    printf("Enter a number (0 for Sunday to 6 for Saturday): ");
    scanf("%d", &dayNumber);
    if (dayNumber < 0 || dayNumber > 6) {
        printf("Invalid input! Please enter a number between 0 and 6.\n");
        return 1;
    }
    day = (enum Day)dayNumber;
    switch (day) {
        case Sunday:
            printf("Day: Sunday\n");
            printf("It is a Weekend.\n");
            break;
        case Monday:
            printf("Day: Monday\n");
            printf("It is a Weekday.\n");
            break;
        case Tuesday:
            printf("Day: Tuesday\n");
            printf("It is a Weekday.\n");
            break;
        case Wednesday:
            printf("Day: Wednesday\n");
            printf("It is a Weekday.\n");
            break;
        case Thursday:
            printf("Day: Thursday\n");
            printf("It is a Weekday.\n");
            break;
        case Friday:
            printf("Day: Friday\n");
            printf("It is a Weekday.\n");
            break;
        case Saturday:
            printf("Day: Saturday\n");

```

```

        printf("It is a Weekend.\n");
        break;
default:
    printf("Unknown day.\n");
    }
return 0;
}

```

Output:

Enter a number (0 for Sunday to 6 for Saturday): 0

Day: Sunday

It is a Weekend.

Write a C program that demonstrates the use of file handling functions such as fopen(), fclose(), fscanf(), and fprintf(). Explain each function with its syntax and purpose. [10 Marks]

File Handling Functions [4 marks]

1. **fopen()** - Opens a file for reading, writing, or appending.

FILE \*fopen(const char \*filename, const char \*mode);

2. **fclose()** - Closes an opened file and flushes the output buffer.

int fclose(FILE \*stream);

3. **fscanf()** - Reads formatted data from a file.

int fscanf(FILE \*stream, const char \*format, ...);

4. **fprintf()** - Writes formatted data to a file.

int fprintf(FILE \*stream, const char \*format, ...);

Program [ 6 Marks]

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

```
int main()
```

```
{
```

```
char src_fname[20], tar_fname[20], ch;
```

```
printf("Enter input file name and target file name :");
```

```
scanf("%s%s", src_fname, tar_fname);
```

```
FILE *fp1,*fp2;
```

```
fp1 = fopen(src_fname, "r");
```

```
if (fp1 == NULL)
```

```
{
```

```
printf("Unable to open file - %s in Read mode\n",src_fname);
```

```
return 1;
```

```
}
```

```
fp2 = fopen(tar_fname, "w");
```

```
if (fp2 == NULL)
```

```
{
```

```
printf("Unable to open file - %s in write mode\n", tar_fname);
```

```
return 2;
```

```
}
```

```
while ((ch = fgetc(fp1)) != EOF)
```

```
{
```

```
fputc(ch, fp2);
```

```
}
```

```
printf("File copied successfully\n");
```

<div data-bbox="79 560 135 600">8 a)</div> <div data-bbox="79 985 135 1025">b)</div>	<pre data-bbox="153 56 890 280">fclose(fp1); fclose(fp2); } Output: Enter input file name and target file name : src.txt des.txt File copied successfully</pre>	<div data-bbox="1497 560 1536 600">[6]</div>
--	---	--