

# C-PROGRAMMING (DEC 2022)





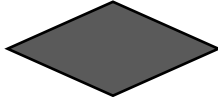
Q. P. CODE: 13334

**Q.1. (15 marks)**

**A. Explain what is flowchart? Explain different symbols in flowchart.**

**(5marks)**

A flowchart is graphical representation of the algorithm of a program. The flowchart gives a nice idea of the flow of the algorithm in sequence as well as the condition changes in the sequence.

Name	Symbol	Use in flowchart
Oval		Used for start or stop of an algorithm.
Flow line		Used to denote the direction of flow.
Parallelogram		Used to perform input or output i.e. take input from keyboard or give output to monitor.
Rectangle		Used to indicate operation is to be performed, for e.g. addition.
Diamond		Used to take a decision, it normally has one input and two outputs based on the condition checked is true or false.

**B. Explain following library functions with proper examples. (5marks)**

### 1. pow( )

This function is available in the header file "math.h". This function calculates and returns the value of  $x^y$ . The parameters x and y are to be passed to it. The parameters as well the answer returned are double data type. But it can also accept parameters of different data type.

#### Code:

```
#include <stdio.h>
#include <math.h>
int main()
{
    int x = 7; // base
    double y = 5.2; // power

    // using the pow() function
    double ans = pow(x, y);
    printf("%f", ans);
    return 0;
}
```

#### Output:

```
24803.319527
```

### 2. ceil( )

This function returns the smallest integral value of the parameter passed to the function. It rounds the value of x upward and returns the value. This function is also in the header file math.h. The parameter accepted and returned of the data type double.

#### Code

```
#include <stdio.h>
#include <math.h>
int main()
```

```
{
    double num = 8.33;
    int result;
    result = ceil(num);
    printf("Ceiling integer of %.2f = %d", num, result);
    return 0;}

```

### Output

Ceiling integer of 7.56 = 8

### 3. floor( )

This function returns the largest integral value of the parameter passed to the function. It rounds the value of x downward and returns the value. This function is also in the header file math.h. The parameter accepted and returned is of double data type.

#### Code

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

int main()
{
    double num = -4.56;
    double result = floor(num);

    printf("Floor integer of %.2f = %.0f", num, result);
    return 0;
}

```

### Output

Floor integer of -4.56 =-5

### 4. sqrt( )

This function finds the square root of the parameter passed to it and the result is returned to the caller function. It is available in math.h. This function also accepts a double as input parameter and returns the result also of double data type.

#### Code

```
#include <math.h>

```

```

#include <stdio.h>
int main()
{
    double number, squareRoot;
    printf("Enter a number: ");
    scanf("%lf", &number);

    // computing the square root
    squareRoot = sqrt(number);

    printf("Square root of %.2lf = %.2lf", number, squareRoot);

    return 0;
}

```

### Output

Enter a number: 5

Square root of 5 = 2.24

**C. Write a program to print following pattern after accepting number of rows from user.**

**(5marks)**

```

1
2 3
4 5 6
7 8 9 10

```

**Code:**

```

#include<stdio.h>
void main()
{
    int i,j,n,k;
    k=1;
    printf("Enter the number of rows:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    {
        for(j=1;j<=i;j++)
        {
            printf("%d",k++);

```

```

    }
    printf("\n");
}

}

```

**Output:**

Enter the number of rows: 4

1

2 3

4 5 6

7 8 9 10

**D. Differentiate between structure and union. (5marks)**

Sr. No.	Structure	Union
1.	Memory allotted for a structure is equal to the space required collectively by all the members of that structure.	Memory allotted for a union is equal to the space required by the largest member of that union.
2.	Data is more secure in structures.	Data can be corrupted in a union.
3.	Structure provides ease of programming.	Unions are comparatively difficult for programming.
4.	Structures require more memory.	Union requires less memory.
5.	Structure must be used when information of all the member elements of structure is to be stored.	Unions must to be used when only one of the member elements of the union is to be stored.

**E. Explain conditional operator with example. (5marks)**

- An operator that requires three operands is called as ternary or conditional operator.
- Syntax of this operator is as given below:  
(condition)? <value if condition is true> : <value if condition is false>;

- **Example:** Program to find greatest of three numbers  
#include <stdio.h>

```
int main()
{
    int n1,n2,n3,greater;
    printf("Enter three numbers:");
    scanf("%d%d%d",&n1,&n2,&n3);
    greater=(n1>n2)?((n1>n3)?n1:n3):((n2>n3)?n2:n3);
    printf("The largest number is %d",greater);

    return 0;
}
```

- **Output:**

```
Enter three numbers:10
20
30
The largest number is 30
```

---

**Q.2.**  
**marks)**

**(15**

- A. What is recursion? Write a program to find GCD of two numbers using recursion. (6marks)**

A function that calls itself is called as a recursive function.

**Code:**

```
#include <stdio.h>

int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }

    return gcd(b, a % b);
}
```

```
int main() {  
    int num1, num2;  
    printf("Enter two numbers: ");  
    scanf("%d %d", &num1, &num2);  
    int result = gcd(num1, num2);  
    printf("GCD of %d and %d is %d\n", num1, num2, result);  
    return 0;  
}
```

**Output:**

Enter two numbers:5

10

GCD of 5 and 10 is 5

**B. Explain different data type modifiers available in C language. (5marks)**

In the C programming language, data type modifiers are used to modify the properties or characteristics of basic data types. These modifiers allow you to fine-tune the behavior and storage of variables. Here are some of the common data type modifiers available in C:

**1. signed / unsigned:**

The `signed` modifier is used to declare a variable as capable of representing both positive and negative values. It is often used implicitly when you declare variables without any modifier.

The `unsigned` modifier is used to declare a variable as capable of representing only non-negative values, effectively doubling the positive range but eliminating the representation of negative numbers.

## 2. short / long:

The `short` modifier is used to declare a variable with a smaller storage size than its default data type. For example, `short int` uses less memory than a regular `int`, but the range of representable values is reduced.

The `long` modifier is used to declare a variable with a larger storage size than its default data type. For example, `long int` uses more memory than a regular `int`, allowing you to represent larger values.

## 3. long long:

The `long long` modifier is used to declare a variable with an even larger storage size than the regular `long` modifier. This is often used to represent very large integers.

## 4. const:

The `const` modifier is used to declare a variable as constant, meaning its value cannot be changed after initialization. It is often used to define constants or to ensure that a variable remains unchanged in a function.

## 5. volatile:

The `volatile` modifier is used to indicate that a variable's value can be changed at any time by external factors that the compiler may not be aware of. This prevents the compiler from optimizing out certain accesses to the variable.

## 6. Bool:

The `Bool` data type modifier is used to declare variables that can hold only two values: `0` (false) or `1` (true). It is typically used in boolean expressions and is often used in conjunction with the `<stdbool.h>` header.

## 7. Complex:

The `Complex` modifier is used to declare variables that hold complex numbers. It is used in conjunction with the `<complex.h>` header and allows you to work with complex arithmetic.



## 8. Imaginary:

The imaginary modifier is used to declare variables that represent the imaginary part of a complex number. Like Complex, it is used with the <complex.h>header.

### C. Write a program to find length of a string using standard library function.

(4marks)

```
#include <stdio.h>

#include <string.h>

int main() {

    char inputString[100];

    printf("Enter a string: ");

    scanf("%s", inputString);

    int length = strlen(inputString);

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

    return 0;

}
```

#### Output:

Enter a string: Mumbai

Length of the string: 6

**Q.3.**

**(15 marks)**

**A. Write a program to display all prime numbers from 100 to 500.  
(6marks)**

```
#include <stdio.h>

int is_prime(int num) {
    if (num <= 1)
    {
        return 0;
    }
    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) {
            return 0;
        }
    }
    return 1;
}

int main() {
    printf("Prime numbers between 100 and 500:\n");
    for (int i = 100; i <= 500; i++) {
        if (is_prime(i)) {
            printf("%d,", i);
        }
    }
    return 0;
}
```

**Output:**

Prime numbers between 100 and, 500:

101,103,107,109,113,127,131,139,149,151,163,167,173,181,191,193,  
197,  
199,211,233,227,229,233,239,241,251,257,263,269,271,277,281,283,  
293,  
307,311,313,317,331,337,347,349,353,359,367,373,379,383,389,397,  
401, 419,421,431,433,439,443,449,457,461,463,467,479,487,491,499

**B. Write a program in C to find average of N elements entered by a user**

**using array.**

**(5marks)**

```
#include <stdio.h>

int main()
{
    int N;

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

    int numbers[N];
    for (int i = 0; i < N; i++) {
        printf("Enter element %d: ", i + 1);
        scanf("%d", &numbers[i]);
    }

    int sum = 0;
    for (int i = 0; i < N; i++) {
        sum += numbers[i];
    }
}
```

```
float average = (float)sum / N;
printf("Average of %d elements: %.2f\n", N, average);
return 0;
}
```

**Output:**

Enter the number of elements: 4

Enter element 1: 25

Enter element 2: 12

Enter element 3: 13

Enter element 4: 14

Average of 4 elements: 16.00

**C. Explain with example left and right shift bitwise operator. (4marks)**

Bitwise shift operators in C are used to shift the bits of a value left or right by a specified number of positions. These operators are ``<<`` for left shift and ``>>`` for right shift. Bitwise shifts can be used for quick multiplication or division by powers of 2, as well as manipulating individual bits in a value.

**1. Left Shift (`<<`):**

The left shift operator shifts the bits of a number to the left by the specified number of positions. It's equivalent to multiplying the number by 2 raised to the power of the shift count.

**2. Right Shift (`>>`):**

The right shift operator shifts the bits of a number to the right by the specified number of positions. It's equivalent to integer division of the number by 2 raised to the power of the shift count.

*Here's a complete example program that demonstrates both left and right shift operators:*

```
#include <stdio.h>

int main() {

    unsigned int num = 12;

    // Left shift

    unsigned int leftShifted = num << 2;

    printf("Left shifted: %u\n", leftShifted);

    // Right shift

    unsigned int rightShifted = num >> 2;

    printf("Right shifted: %u\n", rightShifted);

    return 0;

}
```

---

**Q.4.**  
**(15 marks)**

**A. Write a program to store information of 10 students using structures. Information include roll, name, marks of students. (6marks)**

```
#include <stdio.h>

#include <string.h>
```

```

struct Student {
    int roll;
    char name[50];
    int marks;
};

int main() {
    struct Student students[4];
    for (int i = 0; i < 4; ++i) {
        printf("Enter information for student %d:\n", i + 1);
        printf("Roll: ");
        scanf("%d", &students[i].roll);
        printf("Name: ");
        scanf("%s", students[i].name);
        printf("Marks: ");
        scanf("%d", &students[i].marks);
    }
    printf("\nStudent Information:\n");
    printf("Roll\tname\tmarks\t\n");
    printf("-----\n");
    for (int i = 0; i < 4; ++i)
        {

```

```
printf("%d\t%s\t %d\n",students[i].roll,students[i].name,students[i].marks);  
  
    }  
  
    return 0;  
  
}
```

**Output:**

```
Enter information for student 1:  
Roll: 1  
Name: riya  
Marks: 80  
Enter information for student 2:  
Roll: 2  
Name: tiya  
Marks: 70  
Enter information for student 3:  
Roll: 3  
Name: ram  
Marks: 50  
Enter information for student 4:  
Roll: 4  
Name: sham  
Marks: 40  
Student Information:  
Roll   name   marks  
-----  
1    riya    80  
2    tiya    70  
3    ram     50  
4    sham    40
```

**B. Write a c program to check whether entered string is palindrome or not.**

**(5marks)**

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

```
void main()
```

```
{  
    int n=0,i;  
    char a[100],rev[100];  
    printf("Enter a string:");  
    scanf("%s",a);  
    while(a[n]!='\0');  
    {  
        n++;  
    }  
    for(i=0;i<=(n-1);i++)  
    {  
        rev[n-i-1]=a[i];  
    }  
    for(i=0;i<=n-1;i++)  
    {  
        if(a[i]!=rev[i])  
            break;  
    }  
    if(i==n)  
        printf("The string is palindrome.");  
    else  
        printf("The string is not palindrome.");
```



```
}
```

**Output:**

Enter a string: Mom

The string is palindrome.

**C. Write output of the following code.**

**(4marks)**

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

```
void main()
```

```
{
```

```
int i;
```

```
for(i=0;i<5i++);
```

```
printf(“%d”,i);
```

```
printf(“\n hi”);
```

```
}
```

**OUTPUT:**

5

hi

---

**Q.5.**

**A. Write a program to find largest of three numbers.**

**(6marks)**

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

```
int main()
```

```
{
```

```
int a,b,c;

printf("Enter three numbers:");

scanf("%d%d%d",&a,&b,&c);

if(a>b)
{
    if(a>c)
    {
        printf("%d is greater",a);
    }
    else
    {
        printf("%d is greater",c);
    }
}
else
{
    if(b>c)
    {
        printf("%d is greater",b);
    }
    else
    {
```

```
        printf("%d is greater",c);
    }
}
return 0;
}
```

**Output:**

Enter three numbers:

5

8

4

8 is greater

**B. Differentiate between entry and exit control loop (5marks)**

1. Entry-Controlled Loop:

An entry-controlled loop is a loop where the condition is checked before the loop body is executed. This means that if the condition is initially false, the loop body will not execute at all. In C, the `while` loop and the `for` loop are examples of entry-controlled loops.

Example using a `while` loop (entry-controlled):

```
int i = 0;
while (i < 5) {
    printf("%d\n", i);
    i++;
}
```

## 2. Exit-Controlled Loop:

An exit-controlled loop is a loop where the loop body is executed at least once before the condition is checked. In other words, the loop body is guaranteed to run at least once, even if the condition is false from the beginning. In C, the `do-while` loop is an example of an exit-controlled loop.

Example using a `do-while` loop (exit-controlled):

```
int i = 0;

do {

    printf("%d\n", i);

    i++;

} while (i < 5);
```

In summary:

Entry-Controlled Loop (while loop and for loop): The loop body is executed only if the condition is initially true.

Exit-Controlled Loop (do-while loop): The loop body is executed at least once before the condition is checked.

A function prototype in C serves as forward declaration of a function before its actual implementation or definition. It informs the compiler about the function's name, return type, and parameter type, allowing the compiler to perform type checking and validation during compilation.

### **C. Explain need of function prototype with example. (4marks)**

A function prototype in C serves as forward declaration of a function before its actual implementation or definition. It informs the compiler about the function's name, return type, and parameter type, allowing the compiler to perform type checking and validation during compilation.

The general syntax of function prototype in c :  
return\_type function\_name(parameter1\_type, parameter2\_type, ...);

**Example:**

```
#include <stdio.h>

// Function prototype

int add(int a, int b);

int main() {

    int result;

    // Function call before definition

    result = add(5, 3);

    printf("The sum is: %d\n", result);

    return 0;

}

// Function definition

int add(int a, int b) {

    return a + b;

}
```

**Q.6.**

**A. Write a program to check whether square matrix is symmetric or not.**

**(6marks)**

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

```
#include <stdbool.h>
```

```
#define MAX_SIZE 10
```

```
bool isSymmetric(int matrix[MAX_SIZE][MAX_SIZE], int size)
```

```
{
```

```
for (int i = 0; i < size; i++)
```

```
{
```

```
for (int j = 0; j < size; j++)
```

```
{
```

```
if (matrix[i][j] != matrix[j][i])
```

```
{
```

```
return false;
```

```
}
```

```
}
```

```
}
```

```
return true;
```

```
}
```

```
int main() {  
  
    int size;  
  
    printf("Enter the size of the square matrix: ");  
  
    scanf("%d", &size);  
  
    if (size <= 0 || size > MAX_SIZE)  
    {  
        printf("Invalid matrix size.\n");  
        return 1;  
    }  
  
    int matrix[MAX_SIZE][MAX_SIZE];  
  
    printf("Enter the elements of the matrix:\n");  
  
    for (int i = 0; i < size; i++)  
    {  
        for (int j = 0; j < size; j++)  
        {  
            scanf("%d", &matrix[i][j]);  
        }  
    }  
  
    if (isSymmetric(matrix, size)) {  
        printf("The matrix is symmetric.\n");  
    }  
}
```

```
} else {  
    printf("The matrix is not symmetric.\n");  
}  
  
return 0;  
}
```

**Output:**

Enter the size of the square matrix:3

Enter the elements of the matrix:

1

0

0

0

1

0

0

0

1

The matrix is symmetric

**B. Write algorithm and draw flowchart to check whether entered number is prime or not. (5marks)**

1. Start the program.

2. 2. Declare a function `isPrime` that takes an integer `num` as an argument and returns an integer:

- If `num` is less than or equal to 1, return 0 (indicating not prime).



- Iterate from `i` starting at 2 up to the square root of `num`:
  - If `num` is divisible by `i` (i.e., `num % i == 0`), return 0 (indicating not prime).
  - If no divisors were found, return 1 (indicating prime).
- 3. In the `main` function:
  - Declare an integer variable `num` to store the user-entered number.
  - Display a prompt to the user: "Enter a number: ".
  - Read the user's input into the `num` variable using `scanf`.
- 4. Call the `isPrime` function with `num` as an argument to check whether it's prime or not.
- 5. Based on the return value of `isPrime`, display an appropriate message:
  - If `isPrime(num)` returns 1, print "num is a prime number."
  - If `isPrime(num)` returns 0, print "num is not a prime number."
- 6. End the program.

**C. Explain with syntax and example multi-way branching statement.**

**(4marks)**

**Syntax:**

```
switch (expression) {  
  
    case constant1:  
  
        // Code to be executed if expression equals constant1  
  
        break;  
  
    case constant2:  
  
        // Code to be executed if expression equals constant2  
  
        break;  
  
}
```

```
// Add more cases as needed

default:

    // Code to be executed if no case matches expression
}
```

**Example:**

```
#include <stdio.h>

int main() {

    int day =3;

    switch (day) {

        case 1:

            printf("Monday\n");

            break;

        case 2:

            printf("Tuesday\n");

            break;

        case 3:

            printf("Wednesday\n");

            break;

        case 4:

            printf("Thursday\n");

            break;

        case 5:
```

```
        printf("Friday\n");  
        break;  
    case 6:  
        printf("Saturday\n");  
        break;  
    case 7:  
        printf("Sunday\n");  
        break;  
    default:  
        printf("Invalid day\n");  
    }  
  
    return 0;  
}
```

**Output:**

Wednesday