MUMBAI UNIVERSITY CBCGS SEM 2
ENGINEERING C Programming - Dec 2023 SOLUTIONS

Q.1. A Define Flowchart? Explain different symbols used in drawing Flowchart. [05]

Ans:- A flowchart is a graphical representation of a process, algorithm, or workflow. It uses standardized symbols and connectors to depict the steps involved in the process and the sequence in which they occur. Flowcharts are commonly used in various fields such as software development, business process mapping, engineering, and education to illustrate complex systems in a visual and easily understandable manner.

Here are some commonly used symbols in flowcharts:

1. Start/End Symbol: This symbol represents the beginning or end of the process. It is usually depicted as a rounded rectangle with the word "Start" or "End" inside.
2. Process Symbol: This symbol represents a specific task, operation, or action performed within the process. It is typically depicted as a rectangle with the description of the task written inside.
3. Input/Output Symbol: This symbol represents input or output operations in the process, such as data input or output. It is usually depicted as a parallelogram with the description of the input or output operation written inside.
4. Decision Symbol: Also known as a conditional or branching symbol, this symbol represents a decision point where the flow of the process may diverge based on a condition. It is typically depicted as a diamond shape with the condition written inside, and arrows indicating the possible paths based on the condition (usually labeled as "yes" and "no").
5. Connector Symbol: This symbol is used to connect different parts of the flowchart that are located on different pages or sections. It is depicted as a small circle, and its purpose is to maintain the continuity of the flowchart when it extends beyond a single page

B. Difference between while loop & do-while loop. [05]

Ans:- Certainly! Here's the table with examples added in the next row:

These examples provide a clearer understanding of how each loop behaves in different situations.

Feature	while Loop	do-while Loop
Execution	Condition is evaluated before loop body	Loop body is executed before condition
Initial Condition	If initial condition is false, loop body is never executed	Loop body is executed at least once regardless of initial condition

Syntax	while (condition) {
	  // Loop body
	}
	Example: int x = 5; while (x > 10) { // Code inside loop }
		do {
		  // Loop body
		} while (condition);
		Example: int y = 5; do { // Code inside loop } while (y > 10);

Example	Loop body will not execute because initial condition (5 > 10) is false	Loop body will execute at least once because the loop condition is evaluated after the first iteration

C. Explain Different data type modifiers available in C Language. [05]

Ans:- C programming language, data type modifiers are keywords used to modify the properties of variables, such as their size, range, or storage location. Here are the different data type modifiers available in C:

1. const: This modifier is used to declare constants, which are variables whose values cannot be modified once initialized.
- const int MAX_SIZE = 100;

2. volatile**:** This modifier indicates that a variable's value may be changed at any time by external entities such as hardware or other threads.
- volatile int sensorValue;

3. register**:** This modifier suggests the compiler to store the variable in a CPU register for faster access. However, modern compilers are smart enough to optimize this automatically, so its usage is less common nowadays.
- register int counter;

4. static: This modifier has different meanings depending on where it's used:

i. When used with a global variable, it limits the variable's scope to the file in which it is declared.
ii. When used with a local variable, it retains the variable's value between function calls.
iii. When used with a function, it limits the function's scope to the file in which it is declared (similar to static global variables).
- E.G. static int globalVariable;

void exampleFunction() {

```
    static int localVariable;
}

static void staticFunction() {
    // Function definition
}
```

D.Write a program to print the following pattern for number of rows given by the user. [05]

1
21
321
4321

Ans:-

```
#include <stdio.h>

int main() {
    int numRows, i, j, k;

    printf("Enter the number of rows: ");
    scanf("%d", &numRows);

    for (i = 1; i <= numRows; i++) {
        // Printing spaces
        for (j = 1; j <= numRows - i; j++) {
            printf(" ");
        }

        // Printing numbers in decreasing order
        for (k = i; k >= 1; k--) {
            printf("%d", k);
        }

        printf("\n");
    }

    return 0;
}
```

Output:-
Enter the number of rows: 4

```
  1
 21
321
4321
```

E Write a program to display all prime numbers from 1 to 50. [05]

Ans:-

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

// Function to check if a number is prime
bool isPrime(int num) {
    if (num <= 1) {
        return false;
    }
    for (int i = 2; i * i <= num; i++) {
        if (num % i == 0) {
            return false;
        }
    }
    return true;
}

int main() {
    printf("Prime numbers between 1 and 50 are:\n");
    for (int i = 1; i <= 50; i++) {
        if (isPrime(i)) {
            printf("%d ", i);
        }
    }
    printf("\n");
    return 0;
}
```

Output:-

```
 Prime numbers between 1 and 50 are:
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

Q.2.A. Differentiate between break and continue [05]

Ans:-

Sure, here's a table differentiating between **break** and **continue** statements in C:

Feature	break Statement	continue Statement

Purpose	Used to terminate the loop prematurely	Used to skip the current iteration of the loop and continue with the next iteration
Loop affected	Terminates the loop in which it is encountered	Skips the remaining code in the current iteration and proceeds to the next iteration of the loop
Usage	Typically used inside loops (e.g., for, while, do-while)	Typically used inside loops (e.g., for, while, do-while)
Effect	Immediately exits the loop and continues with the statement following the loop	Skips the remaining code in the current iteration and proceeds with the next iteration of the loop
Example	for (int i = 0; i < 10; i++) { if (i == 5) { break; } printf("%d ", i); }	for (int i = 0; i < 10; i++) { if (i == 5) { continue; } printf("%d ", i); }
Output (Example)	0 1 2 3 4	0 1 2 3 4 6 7 8 9

This table summarizes the key differences between **break** and **continue** statements in C

B Explain the following functions with proper example. i) pow( ) ii) ceil( ) iii) floor( ) [05]

Ans:-

i) pow() function:

- The pow() function is used to calculate the power of a given number.
- It takes two arguments: the base and the exponent.
- It returns the result of raising the base to the power of the exponent.
- This function is declared in the <math.h> header file.

Example: #include <stdio.h>
#include <math.h>
int main() {
double base = 2.0, exponent = 3.0;
double result = pow(base, exponent);
  printf("Result: %.2f\n", result);
return 0;
}

ii) ceil() function:

- The ceil() function is used to round a floating-point number up to the nearest integer, returning the smallest integer value greater than or equal to the given number.
- It takes a single argument: the floating-point number to be rounded up.
- This function is declared in the <math.h> header file.

Example:

#include <stdio.h>
#include <math.h>
int main()
{
 double number = 4.3;

```
 double roundedUp = ceil(number);
printf("Rounded up: %.2f\n", roundedUp);
return 0;
}
```

iii) floor() function:

- The floor() function is used to round a floating-point number down to the nearest integer, returning the largest integer value less than or equal to the given number.
- It takes a single argument: the floating-point number to be rounded down.
- This function is declared in the <math.h> header file.

Example:

```
#include <stdio.h>
 #include <math.h>
int main()
 {
double number = 4.8;
double roundedDown = floor(number);
printf("Rounded down: %.2f\n", roundedDown);
return 0;
 }
```

C. Write a program to find value of y using recursive function ,where $y = x^n$ **[05]**

Ans:-

```
#include <stdio.h>
// Recursive function to calculate power
double power(double x, int n) {
    if (n == 0) {
        return 1.0;
    } else if (n > 0) {
        return x * power(x, n - 1);
    } else {
        return 1.0 / (x * power(x, -n - 1));
    }
}

int main() {
    double x;
    int n;

    printf("Enter the value of x: ");
    scanf("%lf", &x);
    printf("Enter the value of n: ");
    scanf("%d", &n);

    double result = power(x, n);
```

```
    printf("Result: %.2f\n", result);

    return 0;
}
```

Output:-
Enter the value of x: 2
Enter the value of n: 3
Result: 8.00

Q.3. A Explain the sizeof( ) operator with proper example. [05]

Ans:- The sizeof() operator in C is used to determine the size, in bytes, of a datatype or a variable. It's often used to ensure portability and efficiency when working with different data types, especially when dealing with memory allocation or manipulation.

Here's how the sizeof() operator works with an example:

```
#include <stdio.h>


int main() {
    int integerType;
    char charType;
    float floatType;
    double doubleType;

    printf("Size of int: %lu bytes\n", sizeof(integerType));
    printf("Size of char: %lu bytes\n", sizeof(charType));
    printf("Size of float: %lu bytes\n", sizeof(floatType));
    printf("Size of double: %lu bytes\n", sizeof(doubleType));

    return 0;
}
```

Output:-
Size of int: 4 bytes
Size of char: 1 bytes
Size of float: 4 bytes
Size of double: 8 bytes

B. Write a program to find the factorial of a given number. [05]

Ans:-

```
#include <stdio.h>

// Function to calculate factorial
int factorial(int n) {
```

```
    // Base case: factorial of 0 is 1
    if (n == 0) {
        return 1;
    }
    // Recursive case: calculate factorial recursively
    else {
        return n * factorial(n - 1);
    }
}

int main() {
    int num;

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

    // Check if the number is non-negative
    if (num < 0) {
        printf("Factorial is not defined for negative numbers.\n");
    }
    // Calculate factorial and display result
    else {
        int fact = factorial(num);
        printf("Factorial of %d = %d\n", num, fact);
    }

    return 0;
}
```

Output:-

```
Enter a positive integer: 5
Factorial of 5 = 120
```

C Write a program to store and display the name, roll no. and fees of 100 students using structure. [05]

Ans:-

```
#include <stdio.h>

// Define structure for student
struct Student {
    char name[50];
    int rollNo;
    float fees;
};
```

```
int main() {
    // Declare an array of structures to store information of 100 students
    struct Student students[100];

    // Input data for each student
    for (int i = 0; i < 100; i++) {
        printf("Enter details for student %d:\n", i + 1);
        printf("Name: ");
        scanf("%s", students[i].name);
        printf("Roll No.: ");
        scanf("%d", &students[i].rollNo);
        printf("Fees: ");
        scanf("%f", &students[i].fees);
        printf("\n");
    }

    // Display data for each student
    printf("Details of all students:\n");
    for (int i = 0; i < 100; i++) {
        printf("Student %d:\n", i + 1);
        printf("Name: %s\n", students[i].name);
        printf("Roll No.: %d\n", students[i].rollNo);
        printf("Fees: %.2f\n", students[i].fees);
        printf("\n");
    }

    return 0;
}
```

Output:-

```
Details of all students:
Student 1:
Name: John
Roll No.: 101
Fees: 2500.00

Student 2:
Name: Alice
Roll No.: 102
Fees: 3000.00

... (Output for other students)

Student 100:
Name: David
Roll No.: 200
```

Fees: 2800.00

Q.4.A. Write a short note on Structure [05]

Ans:- A structure in C is a composite data type that allows you to group together variables of different types under a single name. It is a way to organize related data items into a single unit, making it easier to manage and manipulate complex data structures. Here are some key points about structures:

1. Definition: Structures are defined using the struct keyword followed by a structure tag, which is an identifier that represents the structure type. Inside the structure, you define variables of different data types, called members or fields.

- struct Person {
  char name[50];
  int age;
  float height;
  };

2. Declaration: After defining a structure, you can declare variables of that structure type. Each variable holds the collection of data items defined in the structure.

- struct Person person1, person2;

3. Accessing Members: You can access members of a structure using the dot operator (.). This allows you to read or modify individual data items within the structure.

- strcpy(person1.name, "John"); person1.age = 30; person1.height = 6.2;

4. Initialization: Structures can be initialized at the time of declaration using braces {}.

- struct Person person3 = {"Alice", 25, 5.6};

5. Nested Structures: Structures can be nested within other structures, allowing for the creation of complex data structures.

- struct Date { int day; int month; int year; }; struct Student { char name[50]; int rollNo; struct Date dob; };

6. Memory Allocation: The memory required for a structure variable is the sum of the memory required for each member. The members are stored in memory sequentially, with padding added for alignment.

- printf("Size of struct Person: %lu bytes\n", sizeof(struct Person));

7. Passing to Functions: Structures can be passed to functions by value or by reference. When passed by value, a copy of the structure is made. When passed by reference, the function operates directly on the original structure.

- void display(struct Person p) { printf("Name: %s\n", p.name); printf("Age: %d\n", p.age); printf("Height: %.2f\n", p.height); }

B.Write a program to find the largest of ‘n’ numbers taken from user. [05]

Ans:-

```
#include <stdio.h>

int main() {

   int n, i;
```

```
    float number, largest;
  // Input the number of elements
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    // Input the first number
    printf("Enter number 1: ");
    scanf("%f", &largest);

    // Compare the remaining numbers
    for (i = 2; i <= n; i++) {
        printf("Enter number %d: ", i);
        scanf("%f", &number);

        // Update largest if the current number is greater
        if (number > largest) {
            largest = number;
        }
    }

    // Display the largest number
    printf("The largest number is %.2f\n", largest);

    return 0;
}
```

Output:-

```
Enter the number of elements: 5
Enter number 1: 12.5
Enter number 2: 6.3
Enter number 3: 20.8
Enter number 4: 15.4
Enter number 5: 9.7
The largest number is 20.80
```

C Explain conditional Operator with proper example. [05]

Ans:- The conditional operator, also known as the ternary operator (**? :**), is a shorthand way of writing an if-else statement in C. It provides a concise syntax for choosing one of two values based on a condition. The general syntax of the conditional operator is:

- condition ? value_if_true : value_if_false;

Here's how it works:

- The condition is evaluated first. If the condition is true, the expression returns **value_if_true**; otherwise, it returns **value_if_false**.
- It can be thought of as a compact form of the if-else statement.

Here's an example to illustrate the use of the conditional operator:

```
#include <stdio.h>
 int main()
 {
 int x = 10; int y;
 y = (x > 5) ? 100 : 200;
printf("Value of y: %d\n", y);
return 0;
}
```

In this example:

- We have an integer variable **x** initialized to 10.
- We use the conditional operator to assign a value to the variable **y** based on the condition **(x > 5)**. If **x** is greater than 5, **y** will be assigned the value 100; otherwise, it will be assigned the value 200.
- Since **x** is indeed greater than 5 (condition is true), **y** is assigned the value 100.
- Finally, we print the value of **y**, which will be 100.

The conditional operator is useful when you need to choose between two values based on a condition in a single expression, providing a concise and readable alternative to the if-else statement.

Q.5.A Explain flowing functions with proper example.
i) strcmp( ) ii) strcat ( ) iii) strlwr( ) [05]

Ans:- i) **strcmp() function:**

- The **strcmp()** function is used to compare two strings lexicographically.
- It returns an integer value less than, equal to, or greater than zero if the first string is found to be less than, equal to, or greater than the second string, respectively.

- It is declared in the **<string.h>** header file.

Example:

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

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

    int result = strcmp(str1, str2);

    if (result < 0) {
        printf("'%s' is less than '%s'\n", str1, str2);
    } else if (result > 0) {
        printf("'%s' is greater than '%s'\n", str1, str2);
    } else {
        printf("'%s' is equal to '%s'\n", str1, str2);
    }

    return 0;
}
```

Output:

```
'apple' is less than 'banana'
```

ii) **strcat() function:**

- The **strcat()** function is used to concatenate (append) one string to the end of another.
- It appends the content of the second string to the end of the first string.
- The resulting concatenated string is stored in the first string buffer.
- It is declared in the **<string.h>** header file.

Example:

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

int main() {
    char str1[50] = "Hello";
    char str2[] = " world!";

    strcat(str1, str2);

    printf("Concatenated string: %s\n", str1);
```

```
    return 0;
}
```

Output:-

Concatenated string: Hello world!

iii) **strlwr() function:**

- The **strlwr()** function is used to convert all uppercase characters in a string to lowercase.
- It modifies the original string in-place.
- It is declared in the **<string.h>** header file.

Example:

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

int main() {
    char str[] = "Hello WORLD";

    printf("Original string: %s\n", str);

    strlwr(str);

    printf("String in lowercase: %s\n", str);

    return 0;
}
```

Output:

Original string: Hello WORLD

String in lowercase: hello world

B Write a program to accept the elements of square matrix and to find sum of elements present on, above and below principal diagonal. [05]

Ans:-

```
#include <stdio.h>

int main() {

    int matrix[10][10];

    int i, j, rows, cols, sum_on_diag = 0, sum_above_diag = 0, sum_below_diag = 0;


    // Input number of rows and columns
```

```
    printf("Enter the number of rows and columns of the matrix (max 10): ");
    scanf("%d", &rows);
    cols = rows;

    // Input elements of the matrix
    printf("Enter the elements of the matrix:\n");
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            printf("Enter element matrix[%d][%d]: ", i, j);
            scanf("%d", &matrix[i][j]);
        }
    }

    // Calculate sum of elements on, above, and below principal diagonal
    for (i = 0; i < rows; i++) {
        for (j = 0; j < cols; j++) {
            if (i == j) {
                sum_on_diag += matrix[i][j];
            } else if (i < j) {
                sum_above_diag += matrix[i][j];
            } else {
                sum_below_diag += matrix[i][j];
            }
        }
    }

    // Display sum of elements on, above, and below principal diagonal
    printf("Sum of elements on principal diagonal: %d\n", sum_on_diag);
    printf("Sum of elements above principal diagonal: %d\n", sum_above_diag);
    printf("Sum of elements below principal diagonal: %d\n", sum_below_diag);
```

```
    return 0;
}
```

Output:-

Sum of elements on principal diagonal: 15

Sum of elements above principal diagonal: 5

Sum of elements below principal diagonal: 16

1 2 3

4 5 6

7 8 9

C Write a program to find square root of a accepted perfect square integer number without using standard sqrt( ) function. [05]

Ans:-

```
#include <stdio.h>

int main() {
    int num, i, sqrt;

    // Input the perfect square integer number
    printf("Enter a perfect square integer number: ");
    scanf("%d", &num);

    // Calculate the square root using linear search
    for (i = 1; i * i <= num; i++) {
        if (i * i == num) {
            sqrt = i;
            break;
        }
    }

    // Display the square root
    printf("Square root of %d is %d\n", num, sqrt);

    return 0;
}
```

Output:-

Enter a perfect square integer number: 16

Square root of 16 is 4

Q.6.A Write a program to add two numbers using user defined function. [05]

Ans:-
```
#include <stdio.h>

// Function to add two numbers
int add(int num1, int num2) {
    return num1 + num2;
}

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

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

    // Call the add function and store the result
    result = add(num1, num2);

    // Display the result
    printf("Sum of %d and %d is %d\n", num1, num2, result);

    return 0;
}
```
Output:-
```
Enter first number: 10
Enter second number: 20
Sum of 10 and 20 is 30
```

B Define array? Explain the static and dynamic initialization of 1D array [05]

Ans:- An array in C is a collection of elements of the same data type that are stored sequentially in memory. Each element in the array is accessed by its index. Arrays provide a convenient way to store and manipulate a fixed-size collection of data.

Static Initialization of 1D Array:
Static initialization of an array involves initializing the array elements at the time of declaration. The size of the array must be known at compile time. The general syntax for static initialization of a 1D array is:
datatype array_name[size] = {value1, value2, ..., valueN};
Example:
int arr1[5] = {1, 2, 3, 4, 5}; // Static initialization of a 1D array with 5 elements
In this example, we initialize an array **arr1** of size 5 with values {1, 2, 3, 4, 5}.
Dynamic Initialization of 1D Array:
Dynamic initialization of an array involves allocating memory for the array at runtime using dynamic memory allocation functions like **malloc()**, **calloc()**, or **realloc()**. This allows the

size of the array to be determined during program execution. The general steps for dynamic initialization of a 1D array are:

1. Allocate memory for the array using a dynamic memory allocation function.
2. Assign values to the array elements using a loop or any other method.
3. Free the allocated memory after use to prevent memory leaks.

Example:

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

int main() {
    int size;
    int *arr2;

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

    // Dynamically allocate memory for the array
    arr2 = (int *)malloc(size * sizeof(int));

    // Check if memory allocation was successful
    if (arr2 == NULL) {
        printf("Memory allocation failed\n");
        return 1; // Exit program
    }

    // Assign values to the array elements
    for (int i = 0; i < size; i++) {
        arr2[i] = i + 1;
    }

    // Display the array elements
    printf("Array elements: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr2[i]);
    }
    printf("\n");

    // Free the allocated memory
    free(arr2);

    return 0;
}
```

In this example, the user is prompted to enter the size of the array. Memory is dynamically allocated for the array **arr2** using **malloc()** based on the use.

C. Draw flowchart to find largest of Three numbers, accepted from user. [05]

Ans:- Creating a flowchart through text is somewhat limited, but I can provide you with a textual representation of a flowchart to find the largest of three numbers accepted from the user: