U.G. 3rd Semester Examination - 2022

Course: [MATHEMATICS HONOURS]

Paper: Skill Enhancement Course (SEC)

Course Code: MATH-H-SEC-T-1A

1.(a) What are the properties that an Algorithm should have?

An algorithm is a finite set of well-defined instructions used to solve a problem. A good algorithm must satisfy the following properties:

- **Input:** An algorithm should have zero or more clearly defined inputs.
- **Output:** An algorithm must produce at least one output.
- **Definiteness:** Each step of the algorithm should be precise, unambiguous, and clearly defined.
- **Finiteness:** The algorithm must terminate after a finite number of steps.
- **Effectiveness:** Every instruction must be basic enough to be executed exactly and efficiently.

1. (b) Explain the coding schemes ASCII and EBCDIC.

ASCII (American Standard Code for Information Interchange):
ASCII is a 7-bit character encoding scheme used to represent text in computers and communication devices. It can represent 128 characters including alphabets, digits, punctuation marks, and control characters. ASCII is widely used in computers, programming languages, and data communication.

EBCDIC (Extended Binary Coded Decimal Interchange Code):
EBCDIC is an 8-bit character encoding scheme developed by IBM. It can represent 256 characters and is mainly used in IBM mainframe and midrange computer systems.

ASCII	EBCDIC
7-bit code	8-bit code
Widely used	Mainly used in IBM systems
Sequential character order	Non-sequential character order

1.(c) Find *x* and *y*, where $(x \,.\, y)_{10} = (10111.1101)_2$.

Given binary number: $(10111.1101)_2$

Integer part:
$10111_2 = (1\times2^4) + (0\times2^3) + (1\times2^2) + (1\times2^1) + (1\times2^0)$
$= 16 + 0 + 4 + 2 + 1 = 23$

Fractional part:
$.1101_2 = (1\times2^{-1}) + (1\times2^{-2}) + (0\times2^{-3}) + (1\times2^{-4})$
$= 0.5 + 0.25 + 0 + 0.0625 = 0.8125$

Therefore,
x = 23 and **y = 0.8125**

1. (d) Explain two types of numeric constants with examples.

Numeric constants in C are classified into two main types:

1. Integer Constants:
These are whole numbers without any fractional part. They can be positive or negative.
Examples: 10, -25, 0, 100

2. Floating Point Constants:
These are numbers containing a decimal point or exponent.
Examples: 3.14, -0.75, 2.5E3

1. (e) Differentiate between Data and Information.

Data	Information
Raw facts and figures	Processed and meaningful data
Unorganized and unprocessed	Organized and structured
Example: 85, 90, 78	Example: Average marks = 84.33
Does not convey meaning	Useful for decision making

1. (f) Explain nested if–else with a suitable example.

A **nested if–else statement** is an if–else statement in which one if or else block contains another if–else statement. It is used when multiple conditions need to be checked in a hierarchical manner.

Syntax:

```
if (condition1) {
    if (condition2) {
        statement;
    } else {
        statement;
    }
} else {
    statement;
}
```

Example:

```
int a = 10, b = 20;

if (a > b) {
    printf("a is greater");
} else {
    if (a < b) {
        printf("b is greater");
    } else {
        printf("a and b are equal");
    }
}
```

Explanation:
First, the condition `a > b` is checked. If it is false, control goes to the else part, where another if–else checks whether `a < b` or both values are equal.

1. (g) What will be the output of the following C code?

```
int main()
{
    constant int ary[4] = {1, 2, 3, 4};
    int *p;

    p = ary + 3;
    *p = 5;

    printf("%d\n", ary[3]);
}
```

Explanation:
The array `ary` contains four elements: {1, 2, 3, 4}. The pointer `p` is assigned the address of `ary[3]` using pointer arithmetic.

The statement `*p = 5;` attempts to modify the value of `ary[3]`. However, the array is declared as `constant int`, which means its elements cannot be modified.

Also, the correct keyword in C for declaring a constant is `const`, not `constant`. Therefore, the line `constant int ary[4] = {1, 2, 3, 4};` will result in a compilation error.

Therefore, this program results in a **compile-time error**.

Output:
Compilation Error (assignment to read-only location)

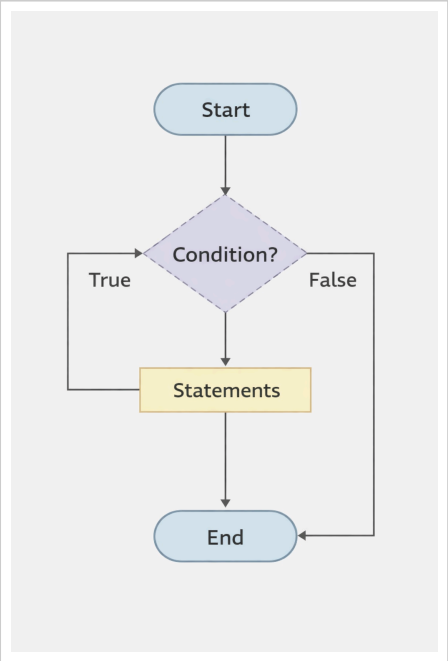
1. (h) Write the syntax of while statement in C and draw the corresponding flow diagram.

Syntax of while loop:

```
while (condition) {
    statements;
}
```

Explanation:
The while loop is an entry-controlled loop. The condition is evaluated first. If it is true, the loop body is executed. This process continues until the condition becomes false.

Flow Diagram:



2. (a) What are the differences between User-defined and Standard Library functions in C? Explain with a suitable example.

Standard Library Functions are pre-defined functions provided by C libraries, whereas **User-defined Functions** are functions written by the programmer to perform specific tasks.

Standard Library Functions	User-defined Functions
Pre-defined and built-in	Defined by the programmer
Stored in header files	Written inside the program
Examples: printf(), scanf(), sqrt()	Examples: sum(), factorial()
Reusable across programs	Reusable only if defined again

Example:

```
#include <stdio.h>

int add(int a, int b) {   // User-defined function
    return a + b;
}

int main() {
    printf("%d", add(5, 3));  // Calls user-defined function
    return 0;
}
```

2. (b) Write the syntax of for-loop in C. Write a C program to find Fibonacci numbers using for-loop.

Syntax of for-loop:

```
for (initialization; condition; increment/decrement) {
    statements;
}
```

Program to generate Fibonacci series:

```
#include <stdio.h>

int main() {
    int n = 10, a = 0, b = 1, c;

    printf("Fibonacci Series: ");
    for (int i = 1; i <= n; i++) {
        printf("%d ", a);
        c = a + b;
        a = b;
        b = c;
    }
    return 0;
}
```

Explanation:
The first two terms are initialized as 0 and 1. Each new term is obtained by adding the previous two terms. The for-loop controls the number of terms generated.

2. (c) How do you initialize a two-dimensional array in C? Can it be passed through a function? Explain with a suitable example.

A two-dimensional array in C is initialized using rows and columns enclosed within braces.

Initialization:

```
int arr[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
```

Passing 2D array to a function:

Yes, a two-dimensional array can be passed to a function by specifying the column size.

```
#include <stdio.h>

void display(int a[2][3]) {
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int arr[2][3] = {
        {1,2,3},{4,5,6}
    };
    display(arr);
    return 0;
}
```

Explanation:

While passing a 2D array to a function, the number of columns must be specified, so that the compiler can correctly access memory locations.

2. (d) Write an Algorithm and draw the corresponding flowchart to find the largest of three distinct numbers.

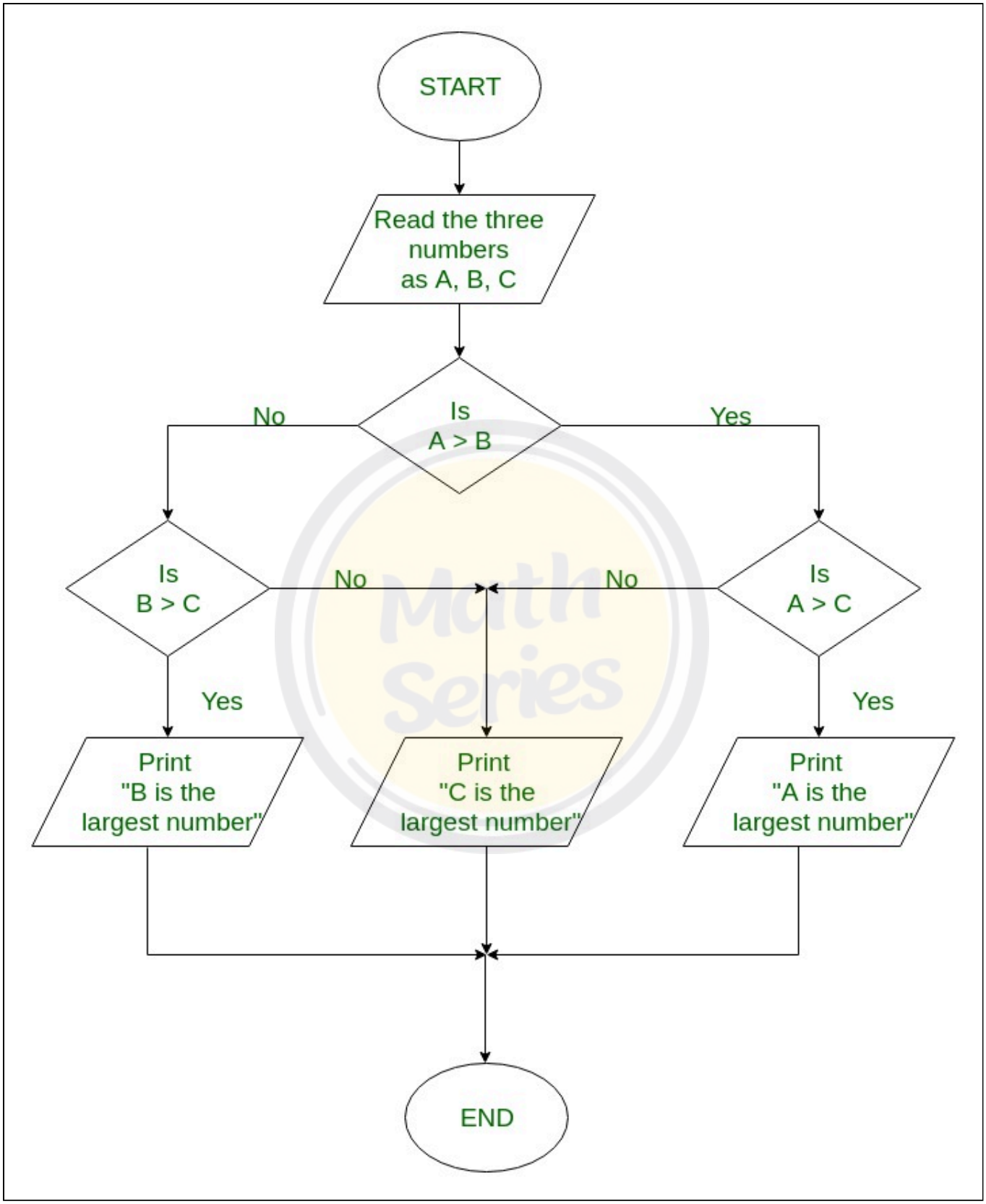
Algorithm:

1. Start
2. Read three distinct numbers a, b, and c
3. If a > b and a > c, then largest = a
4. Else if b > a and b > c, then largest = b
5. Else largest = c
6. Print the value of largest
7. Stop

Explanation:

The algorithm compares the three numbers using conditional statements. First, it checks whether a is greater than both b and c. If not, it checks whether b is the greatest. Otherwise, c is considered the largest. Since the numbers are distinct, only one number can be the largest.

Flowchart:


START
Read the three numbers as A, B, C
Is A > B
No
Yes
Is B > C
No
No
Is A > C
Yes
Yes
Print "B is the largest number"
Print "C is the largest number"
Print "A is the largest number"
END

3. (a) (i) Write a C program to find the sum of the first 1000 natural numbers.

Answer:

```
#include <stdio.h>

int main() {
    int i, sum = 0;

    for (i = 1; i <= 1000; i++) {
        sum = sum + i;
    }

    printf("Sum = %d", sum);
    return 0;
}
```

Explanation:
The loop runs from 1 to 1000 and adds each natural number to the variable `sum`. Finally, the total sum is displayed.

3. (a) (ii) Distinguish between Call by Value and Call by Address.

Answer:

Call by Value	Call by Address
Copy of variable is passed	Address of variable is passed
Original value is not modified	Original value can be modified
No pointers are used	Pointers are used
Safer but less flexible	More flexible

3. (a) (iii) Write a C program to swap two numbers by using Call by Address.

Answer:

```
#include <stdio.h>

void swap(int *x, int *y) {
    int temp;
    temp = *x;
    *x = *y;
    *y = temp;
}

int main() {
    int a = 10, b = 20;
    swap(&a, &b);
    printf("a = %d, b = %d", a, b);
    return 0;
}
```

Explanation:
The addresses of variables are passed to the function. Changes made inside the function affect the original variables.

3. (b) (i) Explain break and continue statements.

Answer:
The **break** and **continue** statements are used to control the flow of loops in C.
break: Terminates the loop immediately and transfers control outside the loop.
continue: Skips the remaining statements of the current iteration and continues with the next iteration.

Example:
```
for (int i = 1; i <= 5; i++) {
    if (i == 3)
        continue;
    printf("%d ", i);
}
```

3. (b) (ii) Write a C program to display prime numbers between 1 and 200 using break and continue statements.

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

int main() {
    int i, j;

    for (i = 2; i <= 200; i++) {
        for (j = 2; j <= i/2; j++) {
            if (i % j == 0)
                break;
        }
        if (j > i/2)
            printf("%d ", i);
    }
    return 0;
}
```

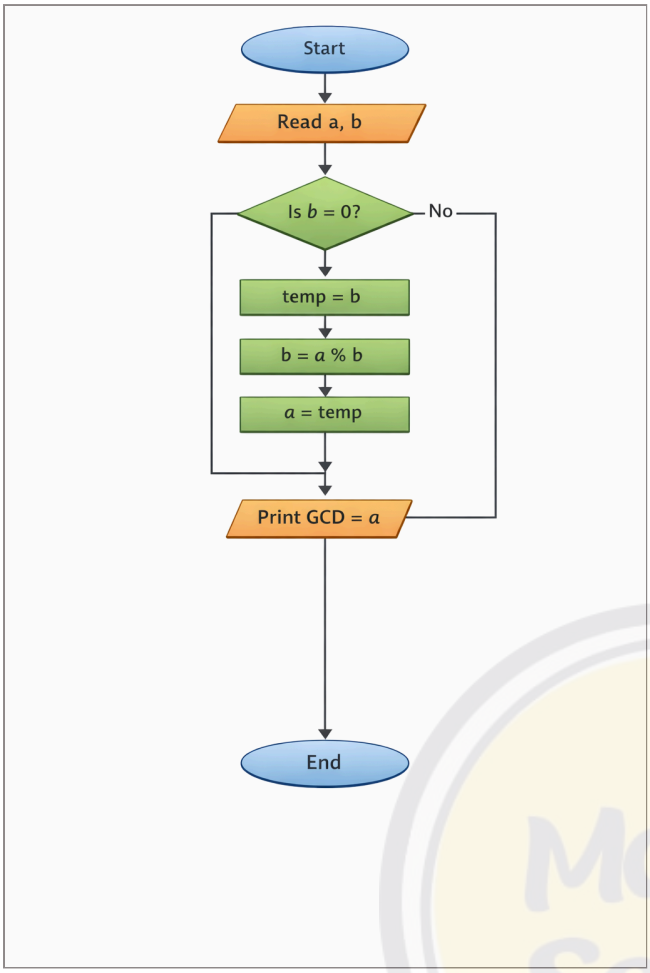
Explanation:
The inner loop checks whether a number is divisible. If divisible, the loop breaks. If no divisor is found, the number is printed as prime.

3. (c) (i) Write the differences between Compiler and Assembler.

Aspect	Compiler	Interpreter
Programming Workflow	Translates the entire program into machine code before execution	Translates and executes code line by line
Program Execution Order	Executes the entire compiled code in one go	Executes code sequentially, line by line
Operational Speed	Generally faster due to pre-compilation	Slower due to real-time translation
Memory Requirements	Requires more memory to store the compiled code	Requires less memory as it translates code on the fly
Data Handling Capacity	Suitable for large programs	Suitable for smaller programs or scripts
Error Detection and Debugging	Detects errors after entire code is compiled	Detects errors line by line, easing debugging
Required Steps	Write code → Compile → Execute	Write code → Execute directly

3. (c) (ii) Design a flowchart to find the G.C.D of two positive integers.

Flowchart:



3. (c) (iii) Write a C program to find the sum of digits of a number.

Answer:

```
#include <stdio.h>

int main() {
    int n, sum = 0, digit;

    printf("Enter a number: ");
    scanf("%d", &n);

    while (n > 0) {
        digit = n % 10;
        sum = sum + digit;
        n = n / 10;
    }

    printf("Sum of digits = %d", sum);
    return 0;
}
```

Explanation:

Each digit is extracted using modulus operator and added to the sum. The loop continues until the number becomes zero.