U.G. 3rd Semester Examination - 2024

Course: [MATHEMATICS HONOURS]

Paper: Skill Enhancement Course (SEC)

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

1. (a) Convert $(101101)_2$ to hexadecimal.

To convert binary to hexadecimal, group the binary digits in sets of four from right.

$(101101)_2 = 0010\ 1101_2$

0010 = 2
1101 = D

Therefore,
$(101101)_2 = (2D)_{16}$

1. (b) Perform the binary subtraction of 1101 and 1011.

Binary subtraction:
```
   1101
 - 1011
 --------
   0010
```

Stepwise:
1 - 1 = 0
0 - 1 → borrow → 10 - 1 = 1
0 - 0 = 0
1 - 1 = 0

Therefore,
$1101_2 - 1011_2 = 0010_2$

1. (c) Write a simple algorithm to find the largest of three numbers.

Algorithm:

```
1. Start
2. Read three 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 largest number
7. Stop
```

Explanation:
The algorithm compares the three values using conditional statements and assigns the greatest value to the variable `largest`.

1. (d) Explain the difference between = and == in C.

= (Assignment Operator)	== (Relational Operator)
Assigns a value to a variable	Compares two values
Used for storing data	Used in conditions
Example: a = 10;	Example: a == 10
Does not return true/false	Returns true or false

Example:
```
int a = 5;      // Assignment
if(a == 5)     // Comparison
printf("a is 5");
```

1. (e) Identify and correct the error in the following program.

Given Code:
```
int arr[5] = {1,2,3,4,5};

for (int i = 0; i <= 5; i++)
{
    printf("%d", arr[i]);
}
```

Error:
The loop condition is wrong. The array has indices from 0 to 4. But the loop runs till i = 5, which causes out-of-bounds access.

Corrected Code:
```
int arr[5] = {1,2,3,4,5};

for (int i = 0; i < 5; i++)
{
    printf("%d", arr[i]);
}
```

Explanation:
The loop should run while `i < 5` because array size is 5. Accessing `arr[5]` causes runtime error.

1. (f) Write a C program that takes an integer as input from the user and checks whether it is even or odd.
```
#include <stdio.h>

int main() {
    int n;

    printf("Enter an integer: ");
    scanf("%d", &n);

    if (n % 2 == 0)
        printf("The number is Even");
    else
        printf("The number is Odd");

    return 0;
}
```

Explanation:
The modulus operator (%) gives the remainder when the number is divided by 2. If the remainder is zero, the number is even; otherwise, it is odd.

1. (g) Differentiate between while and do-while loops with an example.

while loop	do-while loop
Entry-controlled loop	Exit-controlled loop
Condition is checked before execution	Condition is checked after execution
May execute zero times	Executes at least once
Syntax: while(condition)	Syntax: do { } while(condition)

Example of while loop:
```
int i = 1;
while (i <= 3) {
    printf("%d ", i);
    i++;
}
```

Example of do-while loop:
```
int i = 1;
do {
    printf("%d ", i);
    i++;
} while (i <= 3);
```

1. (h) Convert $(-7)_{10}$ to its 8-bit two's complement binary representation.

Step 1: Write binary of +7
$7_{10} = 00000111_2$

Step 2: Find 1's complement
00000111 → 11111000

Step 3: Add 1 to get 2's complement
```
  11111000
+        1
-----------
  11111001
```

Therefore,
$(-7)_{10}$ = 11111001_2 (8-bit two's complement form)

2. (a) Discuss the key features of flowcharts. Draw a flowchart to check if a given number is prime.

Key Features of Flowcharts:

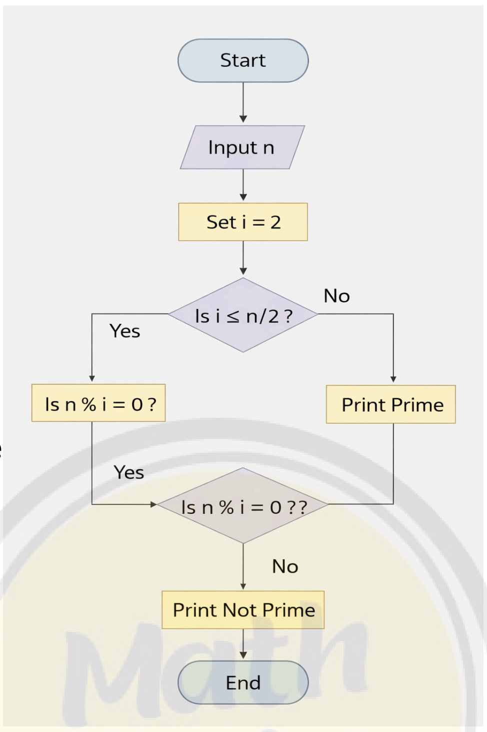
- Graphical representation of an algorithm
- Easy to understand and analyze
- Uses standard symbols
- Shows step-by-step logic flow
- Helpful in debugging programs

Common Flowchart Symbols:

- Oval – Start/End
- Parallelogram – Input/Output
- Rectangle – Process
- Diamond – Decision

Flowchart to check Prime Number:

```
           Start
             ↓
          Input n
             ↓
         Set i = 2
             ↓
       Is i ≤ n/2 ?
    ┌────────Yes───────┐
    ↓                  No
  Is n % i = 0 ?       ↓
┌Yes ┐          Print Prime
          ↓      ↓
    Print Not Prime
            ↓
           End
```



2. (b) Write a C program to find the largest element in a one-dimensional array.

```
#include <stdio.h>

int main() {
    int arr[100], n, i;
    int max;

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

    printf("Enter array elements:\n");
    for(i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    max = arr[0];

    for(i = 1; i < n; i++) {
        if(arr[i] > max)
            max = arr[i];
    }

    printf("Largest element = %d", max);

    return 0;
}
```

Explanation:
The program stores elements in an array. It assumes the first element as maximum and compares it with other elements. If any element is greater, it updates the maximum value.

2. (c) Explain recursion in C. Write a recursive function to compute the factorial of a number.

Recursion:
Recursion is a process where a function calls itself to solve a smaller part of the problem. It continues until a base condition is reached.

Factorial using Recursion:
```
#include <stdio.h>

int factorial(int n) {
    if(n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}

int main() {
    int num;

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

    printf("Factorial = %d", factorial(num));

    return 0;
}
```

Explanation:
The function multiplies the number with factorial of (n-1). When n becomes 0, it returns 1 (base case).

2. (d) Write a C program to print Fibonacci series using a for loop.
```
#include <stdio.h>

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

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

    printf("Fibonacci Series: ");

    for(i = 1; i <= n; i++) {
        printf("%d ", a);
        c = a + b;
        a = b;
        b = c;
    }

    return 0;
}
```

Explanation:
The first two numbers are 0 and 1. Each next number is the sum of previous two numbers. The loop continues till required terms are printed.

3. (a) i) Explain arithmetic, relational and logical operators in C with examples.

Operators in C:
Operators are symbols that tell the compiler to perform specific mathematical or logical manipulations. C is rich in built-in operators.

1. Arithmetic Operators:
Used to perform mathematical calculations.

- `+` (Addition), `-` (Subtraction), `*` (Multiplication), `/` (Division)
- `%` (Modulus): Returns the remainder (e.g., `5 % 2 = 1`).

2. Relational Operators:
Used to compare two values. They return 1 (true) or 0 (false).

- == (Equal to), != (Not equal to)
- > (Greater than), < (Less than)
- >= (Greater than or equal to), <= (Less than or equal to)

3. Logical Operators:
Used to combine two or more conditions or complement the evaluation of the original condition.

- && (Logical AND): True if both operands are true.
- `||` (Logical OR): True if at least one operand is true.
- `!` (Logical NOT): Reverses the logical state (True becomes False).

Example:
```
int a = 10, b = 20; if (a < b && b > 0){
    printf("Result: %d", a + b); // Arithmetic (+) and Logical (&&)
}
```
Explanation:
Arithmetic operators handle the math, relational operators perform the "checks" (like comparing a and b), and logical operators allow us to string multiple checks together into a single decision.

3. (a) ii) Write the syntax to declare and initialize a two-dimensional array. Hence write a C program to multiply two 3×3 matrices.

2D Array Syntax:
A two-dimensional array can be thought of as a table with rows and columns.

Declaration: `data_type array_name[row_size][column_size];`
Initialization: `int arr[2][2] = { {1, 2}, {3, 4} };`

Matrix Multiplication Program (3×3):

```
#include

int main() { int a[3][3], b[3][3], mul[3][3]; int i, j, k;

printf("Enter elements of first 3x3 matrix:\n");
for(i=0; i<3; i++)
    for(j=0; j<3; j++)
        scanf("%d", &a[i][j]);

printf("Enter elements of second 3x3 matrix:\n");
for(i=0; i<3; i++)
    for(j=0; j<3; j++)
        scanf("%d", &b[i][j]);

// Multiplying matrices
for(i=0; i<3; i++) {
    for(j=0; j<3; j++) {
        mul[i][j] = 0;
        for(k=0; k<3; k++) {
            mul[i][j] += a[i][k] * b[k][j];
        }
    }
}

printf("Resultant Matrix:\n");
for(i=0; i<3; i++) {
    for(j=0; j<3; j++)
        printf("%d\t", mul[i][j]);
    printf("\n");
}
return 0;
}
```

Explanation:
Matrix multiplication requires three nested loops. The outermost loops (`i` and `j`) iterate through the rows and columns of the resultant matrix, while the innermost loop (`k`) performs the "dot product" of the i^{th} row of the first matrix and the j^{th} column of the second matrix.

3) b) (i) Differentiate between Iteration and Recursion with examples.

Iteration	Recursion
Uses loops like for, while	Function calls itself
Ends when condition becomes false	Ends when base condition is reached
Consumes less memory	Consumes more memory due to stack calls
Easy to understand and debug	Complex but elegant solution

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

Recursion Example:
```
int print(int n){
    if(n == 0)
        return 0;
    printf("%d ", n);
    return print(n-1);
}
```

3) b) (ii) Write a C program to compute the G.C.D of two numbers using recursion.
```
#include <stdio.h>

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

int main() {
    int x, y;

    printf("Enter two numbers: ");
    scanf("%d %d", &x, &y);

    printf("GCD = %d", gcd(x, y));

    return 0;
}
```

Explanation:
The recursive function repeatedly calls itself with the remainder until the second number becomes zero. The first number at that stage is the GCD.

3) c) (i) Explain how C can be used to approximate the sum of a convergent infinite series.

In C programming, infinite series can be approximated using loops by adding a large number of terms until the change in sum becomes very small.

The program continues adding terms of the series while the next term is greater than a small value (like 0.0001).

This method helps in calculating accurate results for convergent series such as harmonic series, exponential series, etc.

Thus, C uses iteration, floating-point variables, and stopping conditions to approximate infinite series.

3) c) (ii) Write a C program to compute the sum of the infinite series:
S = 1 − 1/2 + 1/3 − 1/4 + 1/5 − ....

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

int main() {
    int i;
    double sum = 0.0;
    int n = 1000;   // number of terms for approximation

    for(i = 1; i <= n; i++) {
        if(i % 2 == 0)
            sum -= 1.0 / i;
        else
            sum += 1.0 / i;
    }

    printf("Approximate Sum = %lf", sum);

    return 0;
}
```

Explanation:
The loop adds positive terms for odd numbers and subtracts terms for even numbers. A large value of n is taken to approximate the infinite series.

4) d) (i) Write short notes on Bubble Sort.

Bubble Sort is a simple sorting algorithm that repeatedly compares adjacent elements and swaps them if they are in the wrong order.

The process continues until the entire list is sorted.

Working Principle:

- Compare first two elements
- Swap if first is greater than second
- Move to next pair and repeat
- After each pass, the largest element moves to the end

Advantages:

- Easy to understand and implement
- Suitable for small datasets

Disadvantages:

- Very slow for large data
- Time complexity is $O(n^2)$

4) d) (ii) Write short notes on Debugging in C.

Debugging in C is the process of identifying, analyzing, and removing errors (bugs) from a C program.

Errors may be:

- **Syntax errors** – mistakes in grammar of code
- **Logical errors** – program runs but gives wrong output
- **Runtime errors** – errors during execution

Common Debugging Methods:

- Using printf() statements to track values
- Using debugging tools like GDB
- Step-by-step execution

Debugging helps improve program correctness and reliability.