1: INTRODUCTION TO PROGRAMMING & ALGORITHMIC THINKING

Title: Introduction to Programming and Problem Solving

Introduction: Programming is the process of solving problems using computers. Before writing programs, it is essential to understand how a computer system works and how problems can be solved logically using algorithms.

Learning Objectives

After completing this module, learners will be able to:

- Identify components of a computer system
- Understand where programs are stored and executed
- Define the algorithm and problem-solving steps
- Represent algorithms using flowcharts and pseudocode

Main Content

Components of a Computer System

- Input and Output devices
- Processor (CPU)
- Memory (RAM)
- Storage devices (hard disk)
- Operating System
- Compiler

Idea of Algorithm

- Step-by-step procedure to solve a problem
- Logical and numerical problem solving

Representation of Algorithms

- Flowchart
- Pseudocode

From Algorithm to Program

- Source code
- Variables and data types
- Memory locations
- Syntax errors
- Logical errors
- Object code and executable code

Example

Algorithm to find the sum of two numbers:

1. Start
2. Read two numbers
3. Add the numbers
4. Display the result
5. Stop

Activity: Write a pseudocode to find the largest of two numbers.

Assessment

1. What is an algorithm?
2. The difference between a syntax error and a logical error.

Summary: Understanding computer components and algorithmic thinking is the foundation of programming.

2: OPERATORS, CONDITIONAL BRANCHING & LOOPS

Title: Operators and Control Structures in Programming

Introduction: Operators and control structures allow a program to make decisions and repeat actions efficiently.

Learning Objectives

Learners will be able to:

- Use different types of operators
- Write conditional statements

- Implement looping constructs

Main Content

Operators

- Arithmetic operators (+, -, *, /, %)
- Relational operators (<, >, <=, >=, ==, !=)
- Logical operators (AND, OR, NOT)
- Bitwise operators
- Operator precedence

Conditional Branching

- if statement
- if-else statement
- nested if-else

Loops

- while loop
- for loop
- do-while loop
- Iteration and repetition of statements

Example

```
if (a > b)
    printf("A is greater");
else
    printf("B is greater");
```

Activity: Write a program to check whether a number is even or odd.

Assessment: What is the purpose of loops?

Summary: Operators and control statements help programs take decisions and repeat tasks efficiently.

3: ARRAYS AND STRINGS

Title: Arrays and String Handling

Introduction: Arrays and strings allow the storage and manipulation of multiple values efficiently.

Learning Objectives

Students will be able to:

- Declare and initialise arrays
- Work with one-dimensional and two-dimensional arrays
- Understand strings and character arrays

Main Content

Arrays

- Array declaration and initialisation
- Bound checking
- One-dimensional arrays
- Two-dimensional arrays

Strings

- Character arrays
- String initialization
- String input and output
- Basic string operations

Example

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

Activity: Write a program to find the sum of elements of an array.

Assessment: Difference between array and string.

Summary: Arrays and strings help manage collections of data in an organised manner.

4: FUNCTIONS, RECURSION & POINTERS

Title: Functions, Recursion and Pointer Concepts

Introduction: Functions divide a program into smaller parts. Recursion provides an alternative problem-solving approach, while pointers enable efficient memory handling.

Learning Objectives

Learners will:

- Use functions and libraries
- Understand parameter passing
- Implement recursion
- Learn pointer basics

Main Content

Functions

- Function declaration and definition
- Built-in libraries
- Call by value
- Passing arrays to functions

Recursion

- Recursive function concept
- Factorial
- Fibonacci series
- Ackermann function

Pointers

- Pointer declaration
- Pointer variables
- Call by reference
- Self-referential structures

Example

```
int fact(int n) {
    if(n==0) return 1;
    return n * fact(n-1);
}
```

Activity: Write a recursive program to calculate factorial.

Assessment: What is recursion?

Summary: Functions, recursion, and pointers improve program modularity and efficiency.

5: USER-DEFINED DATA TYPES & FILE HANDLING

Title: Structures, Unions and File Handling

Introduction: User-defined data types and file handling allow complex data organization and permanent storage.

Learning Objectives

Students will:

- Create and use structures and unions
- Handle text and binary files
- Perform file I/O operations

Main Content

Structures

- Structure definition
- Structure variables
- Array of structures
- Nested structures
- Structure and function

Union

- Definition and declaration
- Memory sharing
- Accessing members

File Handling

- File declaration
- Opening and closing files
- Text and binary files

- File I/O operations
- Error handling
- Random file access

Example

```
struct student {
    int roll;
    char name[20];
};
```

Activity: Create a structure to store student information.

Assessment: Difference between structure and union.

Summary: User-defined data types and file handling enable efficient data storage and management.

6: BASIC ALGORITHMS & COMPLEXITY IDEA

Title: Basic Algorithms and Introduction to Complexity

Introduction: Algorithms are essential for efficient problem-solving. Searching and sorting algorithms are widely used in programming.

Learning Objectives

Learners will:

- Understand searching techniques
- Learn basic sorting algorithms
- Get an idea of algorithm complexity

Main Content

Searching Algorithms

- Linear search
- Binary search

Sorting Algorithms

- Bubble sort
- Insertion sort
- Selection sort

Other Algorithms

- Finding roots of equations

Order of Complexity

- Comparison of algorithms using examples
- No formal mathematical definition

Example

Bubble sort idea using repeated comparisons.

Activity: Write steps for insertion sort.

Assessment: Which sorting algorithm is simplest to implement?

Summary: Basic algorithms form the foundation of efficient and optimised programming.