5.1 Programming Concepts

5.1.1 Introduction to Programming Languages

Programming languages are used to write instructions that a computer can understand and execute. These languages help us communicate with the computer to perform tasks, such as solving problems or creating applications. Examples of programming languages include Python, Java, and C.

5.1.2 Low Level, High Level, 4 GL Programming Languages

- **Low-level languages** are close to machine code, like Assembly language. They are faster but harder to understand.
- **High-level languages** are easier for humans to understand, like Python or Java.
- **4GL (Fourth-generation languages)** are more user-friendly and designed to reduce programming time, such as SQL for databases.

5.1.3 Compiler, Interpreter and Assembler

- A **compiler** translates the entire program into machine code at once, making it faster to run.
- An **interpreter** translates the program line by line, which makes it slower but useful for testing.
- An **assembler** translates assembly language code into machine code.

5.1.4 Syntax, Semantic, and Runtime Errors

- **Syntax errors** occur when the rules of the programming language are broken (e.g., missing parentheses).
- **Semantic/logical errors** happen when the program runs, but the logic is wrong (e.g., calculating the wrong result).
- **Runtime errors** occur while the program is running, often due to problems like dividing by zero or trying to access invalid memory.

5.1.5 Control Structures: Sequence, Selection and Iteration

- **Sequence** means executing instructions one after another.
- **Selection** involves making decisions (using "if" statements).
- **Iteration** involves repeating a block of code (using loops like "for" or "while").

5.1.6 Program Design Tools – Algorithm, Flowchart and Pseudocode

- **Algorithm** is a step-by-step plan to solve a problem.

Example: To convert Fahrenheit temperature into celsius

Step 1 : Start

Step 2 : Enter two number A = 10 and B = 20

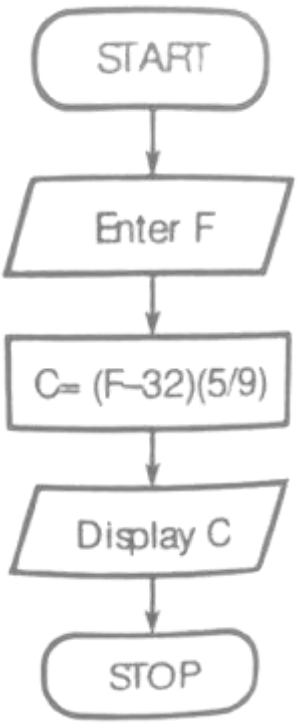
Step 3 : Add A + B = 10 + 20

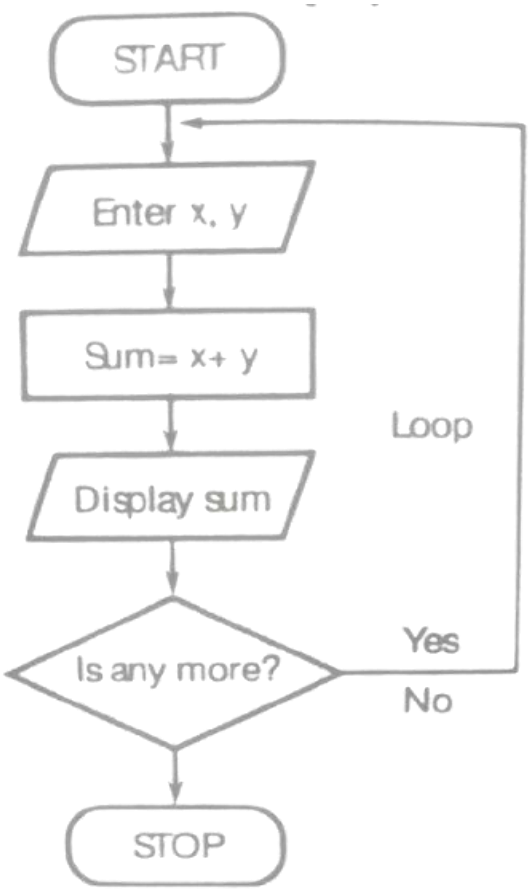
Step 4 : Displaying the result of addition

Step 5 : Stop

- A **flowchart** is a visual representation of the algorithm using symbols.

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision





- **Pseudocode** is writing the algorithm in simple, readable language that's not tied to any programming language.

Example:

IF age > = 18

THEN CASTE vote

ELSE

WAIT until age is not 18

ENDIF

5.1.7 Absolute Binary, BCD, ASCII, and Unicode

- **Absolute binary** is the raw binary representation of data.
- **BCD (Binary-Coded Decimal)** represents decimal digits in binary.
- **ASCII** is a character encoding standard that represents letters and symbols using binary numbers.
- **Unicode** is a universal character encoding standard that supports a wide range of characters from different languages.

5.2 C Programming Language

5.2.1 Introduction and Features of C Language

C is a powerful, general-purpose programming language that is widely used for system software and applications. It is fast, efficient, and easy to learn for beginners.

5.2.2 Structure of C Program

A C program consists of functions, with the main function being where execution starts. It has sections like declarations, functions, and statements.

5.2.3 C Preprocessor and Header Files

The **preprocessor** handles tasks like including libraries and defining constants before the code is compiled. **Header files** contain function declarations and constants used in the program.

5.2.4 Character Set Used in C

C uses a set of characters that includes alphabets, digits, and special symbols. These characters are used to form keywords, identifiers, and values in C.

5.2.5 Use of Comments

Comments are used to explain the code in simple language. They don’t affect the program’s execution but help programmers understand the code better.

Single line Comment

```
// This is a single-line comment
```

Multi line Comment

```
/* This is a multi-line

omment */
```

5.2.6 Identifiers, Keywords, and Tokens

1. Identifiers are names given to variables, functions, and other elements.

- **Must begin with a letter (A-Z or a-z) or an underscore (_).**
  - ✓ Valid: `myVar`, `_count`, `age1`
  - ✗ Invalid: `1age`, `@value`
- **Can contain letters, digits (0-9), and underscores (_).**
  - ✓ Valid: `num_1`, `count123`
  - ✗ Invalid: `price$`, `total-score`
- **Cannot be a C keyword (reserved word).**
  - ✗ Invalid: `int`, `float`, `return` (since they are C keywords)
- **Case-sensitive (C treats uppercase and lowercase differently).**
  - `Total` and `total` are different identifiers.
- **No spaces or special characters (except underscore _).**
  - ✗ Invalid: `first name`, `num#value`

2. Keywords are reserved words in C with special meanings, like "if", "int", or "return."

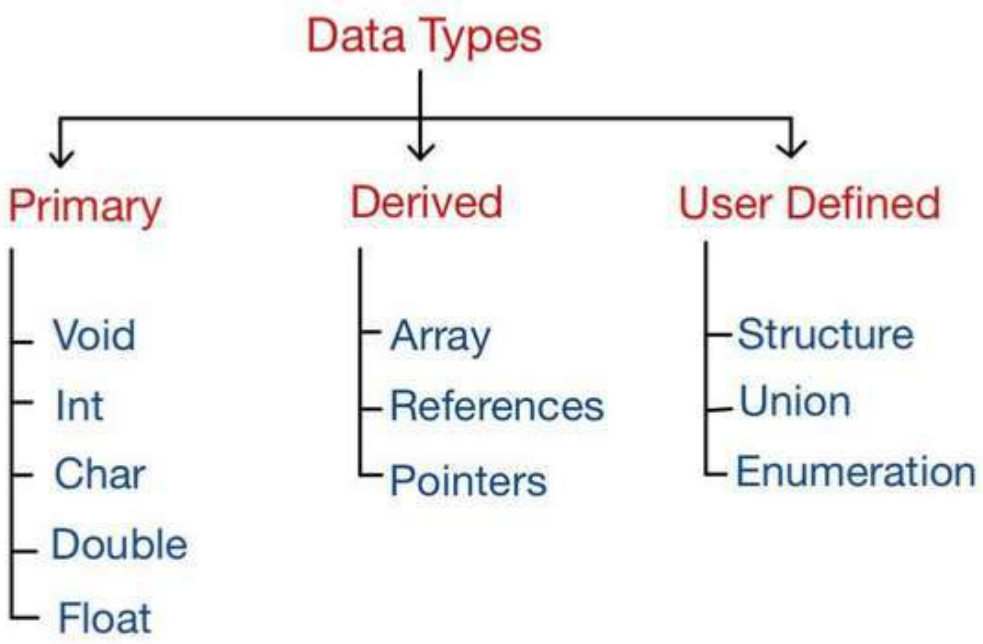
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	Volatile
const	float	short	Unsigned

3. Tokens are the smallest units of a C program, including keywords, identifiers, operators, and constants.

5.2.7 Basic Data Types in C

C supports various **data types** like:

- **int** for integers,
- **float** for decimal numbers,
- **char** for single characters.



5.2.8 Constants and Variables

- **Constants** are fixed values that do not change, like the number 10.
- **Variables** are used to store values that can change, like a student's score.

5.2.9 Type of Specifier

A **type specifier** in C indicates the type of a variable, such as "int" for integers or "float" for decimal numbers.

Data type	Format specifier
Int	%d
Float	%f
char	%c
Char[ ]	%s

5.2.10 Simple and Compound Statements

- A **simple statement** is a single instruction like "x = 5".
- A **compound statement** is a group of statements enclosed in curly braces ({}), used to organize multiple instructions together.

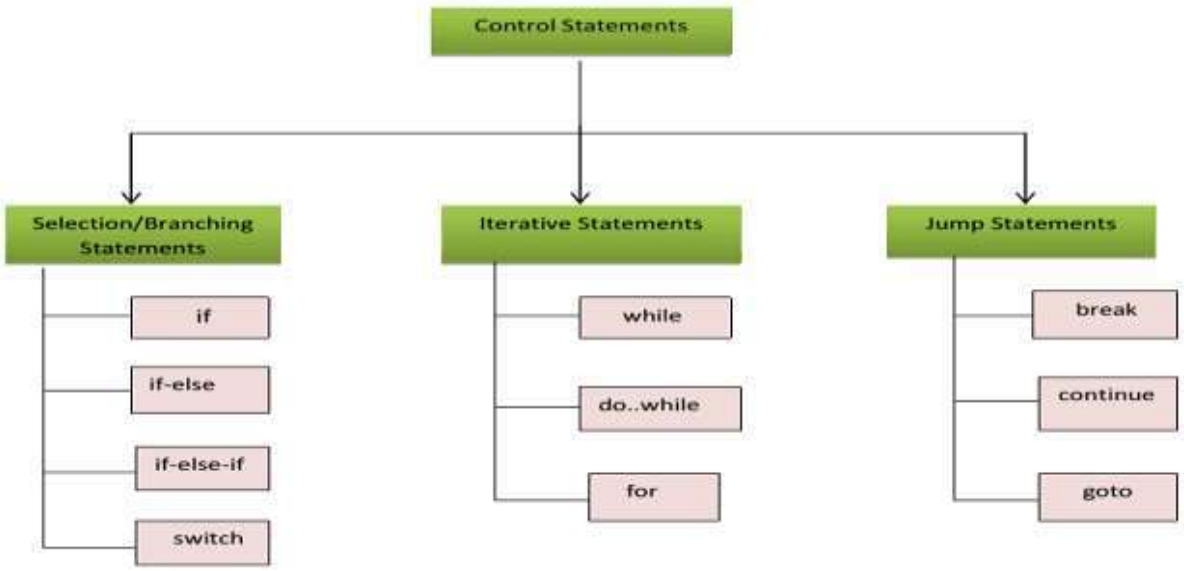
5.2.11 Operators and Expressions: Arithmetic, Relational, Logical, Assignment, Unary, and Conditional Operators

- **Arithmetic operators** (e.g., +, -, *, /) perform math calculations.
- **Relational operators** (e.g., ==, <, >) compare values.
- **Logical operators** (e.g., &&, ||) perform logical operations.
- **Assignment operator** (=) assigns values to variables.
- **Unary operators** (e.g., ++, --) operate on a single value.
- **Conditional operators** (?:) are used for decision-making in a shorthand form.

5.2.12 Input/Output (I/O) Functions

C provides functions like **printf** to display output and **scanf** to take input from the user.

5.2.13 Control Statement



Selection Statement

If statement

The keyword if tells the compiler that what follows is a decision control instruction. The if statement allows us to put some decision -making into our programs.

Syntax of If statement:

```
if (condition )
{
Statement n;
}
```

```c
// Using if statement
if (marks > 90) {
    printf("Excellent!\n");
}
```

If – else statement:

The if statement by itself will execute a single statement, or a group of statements, when the expression following if evaluates to true.

Syntax of if-else statement

```
if (condition )
{
Statement n;
} else {
Statement n;
}
```

```c
// Using if-else statement
if (marks >= 50) {
    printf("You have passed.\n");
} else {
    printf("You have failed.\n");
}
```

else-if Statement: This sequence of if statements is the most general way of writing a multi−way decision. The expressions are evaluated in order; if an expression is true, the statement associated with it is executed, and this terminates the whole chain.

Sytax of else-if statement:

```
if (condition )
{
Statement n;
} else if {
Statement n;
}
else if {
Statement n;
}
else if {
Statement n;
}
else {
Statement n;
}
```

switch case: This structure helps to make a decision from the number of choices. The switch statement is a multi−way decision that tests whether an expression matches one of a number of constant integer values, and branches accordingly

Syntax of switch case:

```
switch( integer expression){

case constant 1 :

statement 1;

case constant 2 :

statement 2;

case constant 3 :

statement 3;

default :

statement n;

}
```

```c
int grade = marks / 10; // Dividing by 10 to get a range

switch (grade) {
    case 3:
        printf("Grade: C\n");
        break;
    case 2:
        printf("Grade: D\n");
        break;
    case 1:
        printf("Grade: E\n");
        break;
    default:
        printf("Grade: F (Failed)\n");
        break;
}
```

5.2.14 Iteration Control Statement: Looping (while, do while, for)

1. while loop repeats code while a condition is true.

Syntax:

```
while(condition) {

    // Code to execute

}
```

```c
#include <stdio.h>

int main() {
    int i = 1;
    while(i <= 5) {
        printf("This is while loop %d\n", i);
        i++; // Increment i
    }
    return 0;
}
```

2. do-while loop repeats code at least once and then checks the condition.

Synatx:

```
do {
    // Code to execute
} while(condition);
```

Example:

```c
#include <stdio.h>

int main() {
    int i = 1;
    do {
        printf("This is do-while loop %d\n", i);
        i++; // Increment i
    } while(i <= 5);
    return 0;
}
```

3. for loop is used when the number of iterations is known.

Syntax:

```
for(initialization; condition; increment/decrement) {
    // Code to execute
}
```

```c
#include <stdio.h>

int main() {
    for(int i = 1; i <= 5; i++) {
        printf("Hello, World! %d\n", i);
    }
    return 0;
}
```

Jumping statement (break, continue, goto)

Continue : The continue statement is used inside loops to skip the rest of the code for the current iteration and move to the next iteration.

Syntax: continue ;

```c
#include <stdio.h>

int main() {
    for(int i = 1; i <= 5; i++) {
        if(i == 3) {
            continue;  // Skips iteration when i is 3
        }
        printf("Number: %d\n", i);
    }
    return 0;
}
```

Break

The break statement is used to exit a loop or switch statement immediately, regardless of the loop condition.

Syntax : break ;

```c
#include <stdio.h>

int main() {
    for(int i = 1; i <= 5; i++) {
        if(i == 3) {
            break;  // Exit the loop when i is 3
        }
        printf("Number: %d\n", i);
    }
    return 0;
}
```

goto

The goto statement transfers control to a labeled statement.

Syntax: goto ;

```c
int main() {
    int num;
    printf("Enter a positive number: ");
    scanf("%d", &num);

    if(num < 0) {
        goto error;  // Jumps to the error label
    }

    printf("You entered: %d\n", num);
    return 0;

error:
    printf("Error! You entered a negative number.\n");
    return 1;
}
```

5.2.15 Array: Definition, Types (1D and 2D), Matrix Addition and Subtraction

An **array** is a collection of similar elements.

- A **1D array** is a single list, like scores of students.
- A **2D array** is a table or matrix, like a chessboard. Matrix addition and subtraction involve adding or subtracting corresponding elements of two matrices.

```c
#include <stdio.h>

int main() {
    int numbers[10]; // Array to store 10 numbers
    int i;

    // Input 10 numbers
    printf("Enter 10 numbers: \n");
    for(i = 0; i < 10; i++) {
        printf("Number %d: ", i + 1);
        scanf("%d", &numbers[i]);
    }

    // Display the stored numbers
    printf("\nThe entered numbers are: \n");
    for(i = 0; i < 10; i++) {
        printf("%d ", numbers[i]);
    }
    printf("\n");

    return 0;
}
```

5.2.16 String: Definition and String Functions (strlen(), strcat(), strcmp(), strrev(), strcpy(), strlwr(), strupr())

A **string** is a series of characters, such as "Hello". Common string functions include:

Ways to initialize string:

char str[ ] = "Hello, World!"; //implicit size

char str[20] = "Hello, World!"; //explicit size

char str[ ] = {'H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd', '!', '\0'}; //manaul initialize

- **strlen()**: Returns the length of a string.
- **strcat()**: Concatenates (joins) two strings.
- **strcmp()**: Compares two strings.
- **strrev()**: Reverses a string.
- **strcpy()**: Copies one string to another.
- **strlwr()**: Converts a string to lowercase.

- **strupr()**: Converts a string to uppercase.

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

int main() {
    char str[] = "Hello, World!"; // Define a string

    // Calculate and display the length of the string
    printf("The length of the string is: %lu\n", strlen(str));

    return 0;
}
```