SAPTHAGIRI NPS UNIVERSITY

PROBLEM SOLVING TECHNIQUES USING C

Course Code : 24BTPHY104

SAPTHAGIRI NPS UNIVERSITY

PROBLEM SOLVING TECHNIQUES USING C

Unit 1

Teja'S
Asst. Professor
Department of Electronics & Communication Engineering

INTRODUCTION TO PROGRAMMING CONCEPTS

- A **programming language** is a set of instructions that humans use to communicate with computers and tell them what to do.

Most Used Programming Languages by Developers
Python
Java
JavaScript
Kotlin
C/C ++
C#
PHP
Go
Swift
R

COMPUTER LANGUAGES

- A **computer language** is a type of language that a computer can understand and use to perform tasks. These languages allow humans to communicate instructions to a computer in a form that it can process.
- Computer languages can be broadly categorized into different types:

1. **Low-level languages** : These are the programming languages that are closer to machine language and hardware, making them more difficult for humans to read and write but highly efficient for the computer to execute.

The low level languages are classified as:

COMPUTER LANGUAGES

i. Machine Language :

- The most basic and fundamental level of programming language.
- Composed entirely of binary code (0s and 1s) that the computer's CPU can directly execute.
- It is specific to the architecture of the computer, meaning that machine code written for one type of CPU will not work on another without modification.

COMPUTER LANGUAGES

ii. Assembly Language :

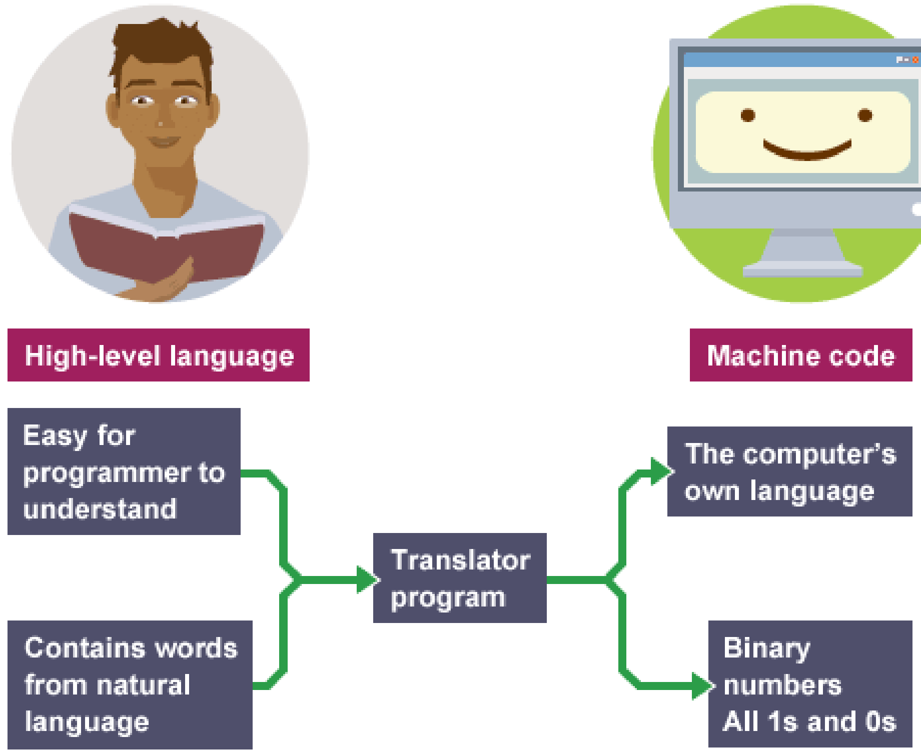
- A step above machine language, assembly language uses symbolic names or mnemonics to represent machine-level instructions, making it slightly easier for humans to write and understand.
- Requires an assembler to convert the assembly code into machine code.

2. High Level Language : **High-level languages** are programming languages that are designed to be easy for humans to read and write.

- High-level languages allow developers to focus more on solving problems and less on the underlying machine architecture.

PROGRAMMING LANGUAGE TRANSLATORS

- **Programming language translators** are tools that convert code written in one programming language into another form that can be executed by a computer.

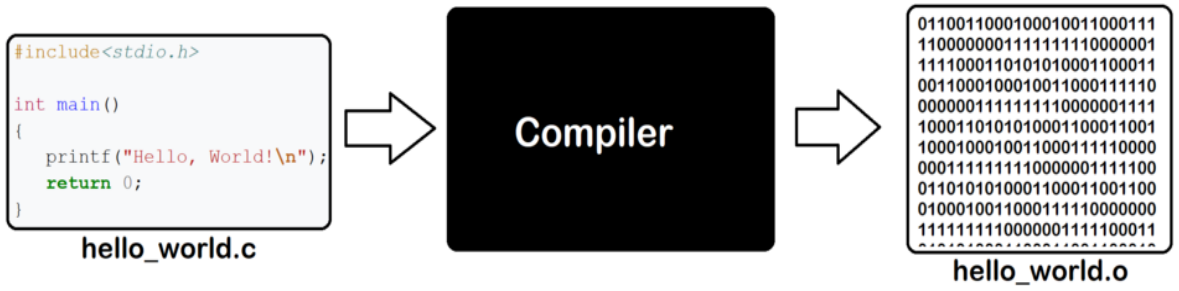

High-level language
Machine code
Easy for programmer to understand
Contains words from natural language
Translator program
The computer's own language
Binary numbers All 1s and 0s

PROGRAMMING LANGUAGE TRANSLATORS

There are three main types of programming language translators:

1. **Compilers :**

- A compiler translates the entire source code of a high-level programming language into machine code (binary code) or intermediate code in one go, creating an executable file.
- **Examples** :
  - **GCC (GNU Compiler Collection)**: Compiles C, C++, and other languages.
  - **Javac** : The Java compiler that converts Java Code into bytecode.



```c
#include<stdio.h>

int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

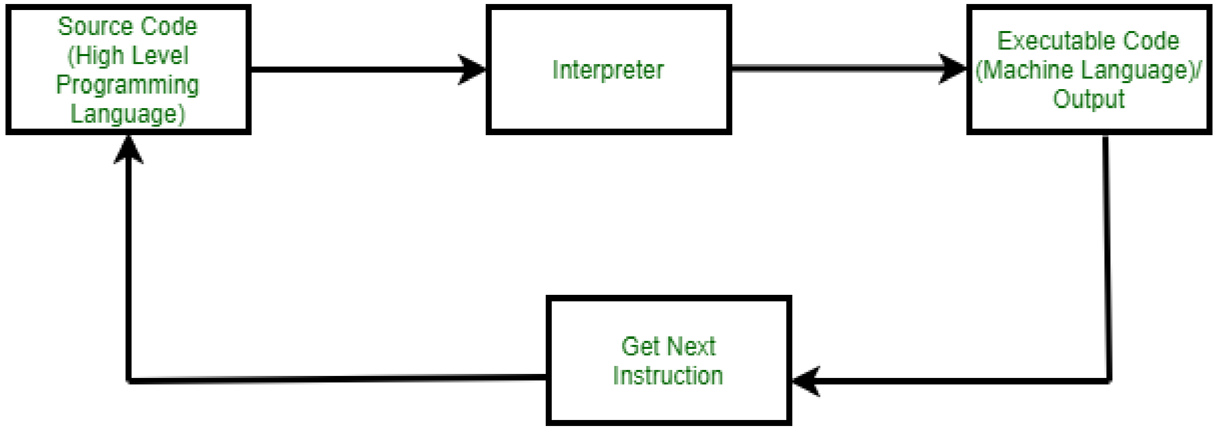
```
011001100010001001100011 1
110000000111111111000000 1
```

PROGRAMMING LANGUAGE TRANSLATORS

SAPTHAGIRI NPS UNIVERSITY

2. Interpreters :

- An interpreter translates and executes code line by line, rather than compiling the entire program at once. It reads each line of code, translates it to machine code, and then executes it immediately.
- **Examples** :
  - **Python Interpreter**: Interprets Python code.

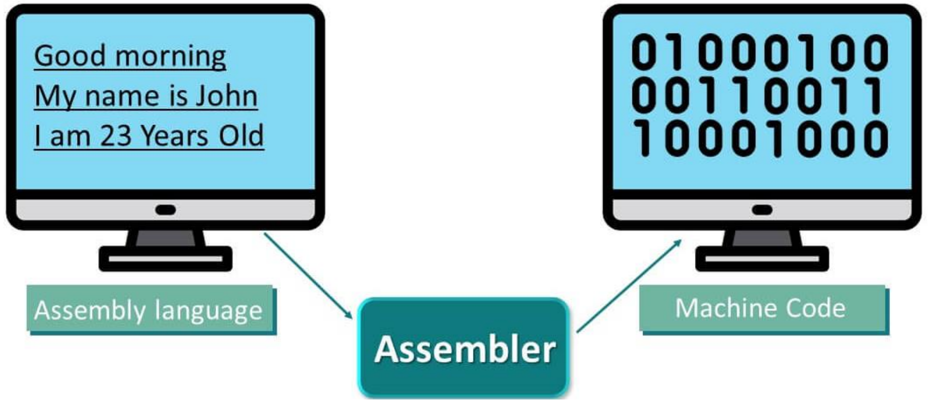


Source Code (High Level Programming Language)
Interpreter
Executable Code (Machine Language)/ Output
Get Next Instruction

PROGRAMMING LANGUAGE TRANSLATORS

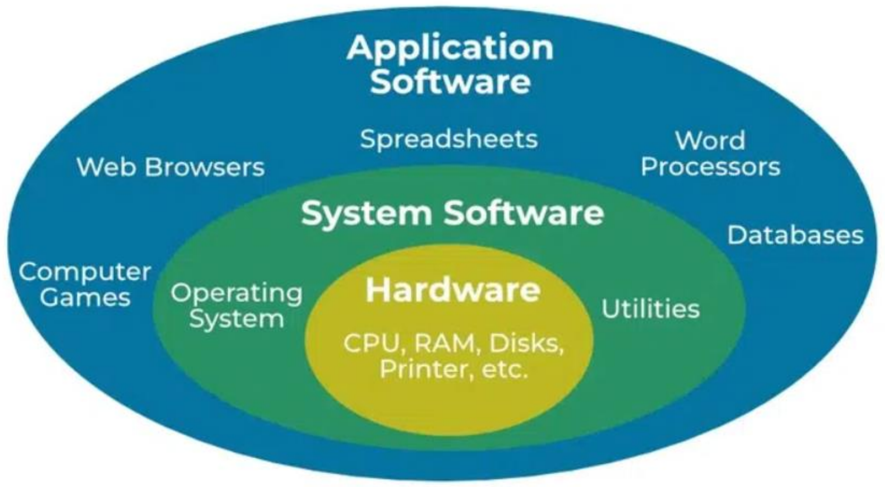
3. Assemblers :

- An **assembler** translates assembly language, which is a low-level human-readable representation of machine code, into actual machine code that can be executed by the computer's CPU.
- **Examples** :
  - **MASM (Microsoft Macro Assembler)**: Assembles code for x86 architecture.



SOFTWARE

- **Software** is a collection of instructions, data, or programs used to operate computers and execute specific tasks.
- Unlike hardware, which refers to the physical components of a computer, software is intangible and runs on the hardware to perform a wide range of functions.



SOFTWARE

Software can be categorized into different types based on its functionality and purpose.

1. **System Software** : System software manages and controls the computer hardware, allowing the application software to perform tasks.

Types:

i. **Operating Systems (OS)**: The main software that manages all hardware and other software on a computer. Examples include Windows, macOS, Linux.

SOFTWARE

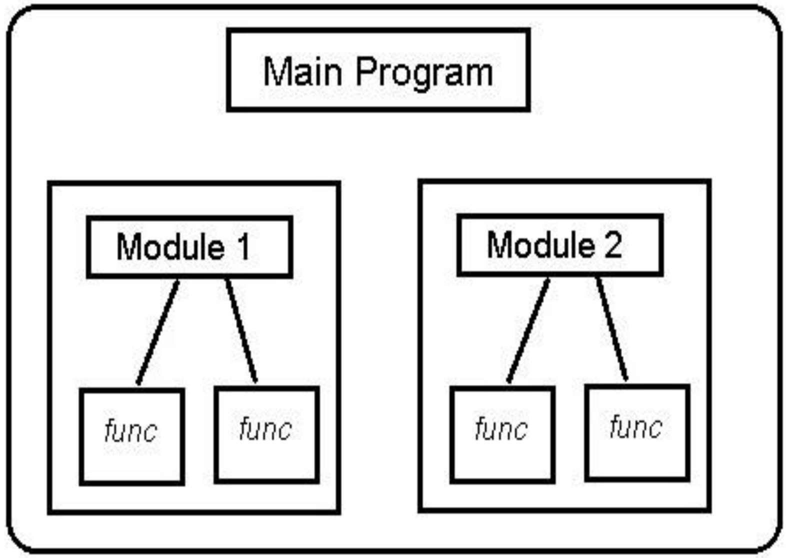
ii. **Device Drivers**: Specialized software that allows the operating system to communicate with hardware devices like printers, graphics cards, and network cards.

iii. **Utility Software**: Programs that perform maintenance tasks, such as antivirus software, disk cleanup tools, and file management tools.

2. **Application Software :** Application Software is designed to help users perform specific tasks or applications. It is built on top of system software and includes programs that serve various user needs.

MODULAR PROGRAMMING

- **Modular programming** is a method of designing and developing software in which the functionality of a program is broken down into separate, interchangeable modules. Each module contains everything necessary to execute only one aspect of the desired functionality.
- The Objective is to improve code organization, reduce complexity, and facilitate code reuse and maintenance.



MODULAR PROGRAMMING

- **Structured Programming :** It is a programming paradigm that emphasizes breaking down a program into smaller, manageable sections or blocks. It follows a logical structure that includes sequences, selections (if-else statements), and loops, without using goto statements.
- **Benefits of Structured Programming :**
    - Improved Readability
    - Ease of Maintenance
    - Reduced Complexity
    - Error Reduction

MODULAR PROGRAMMING

- **Unstructured Programming :** Unstructured programming refers to a style of programming in which the code is written without following a specific structure or methodical approach. The flow of control in the program is often managed through **goto** statements, making the code less organized and harder to follow.

FLOWCHART

A flowchart is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step - by - step approach to solving a task.

Flowchart Symbols:

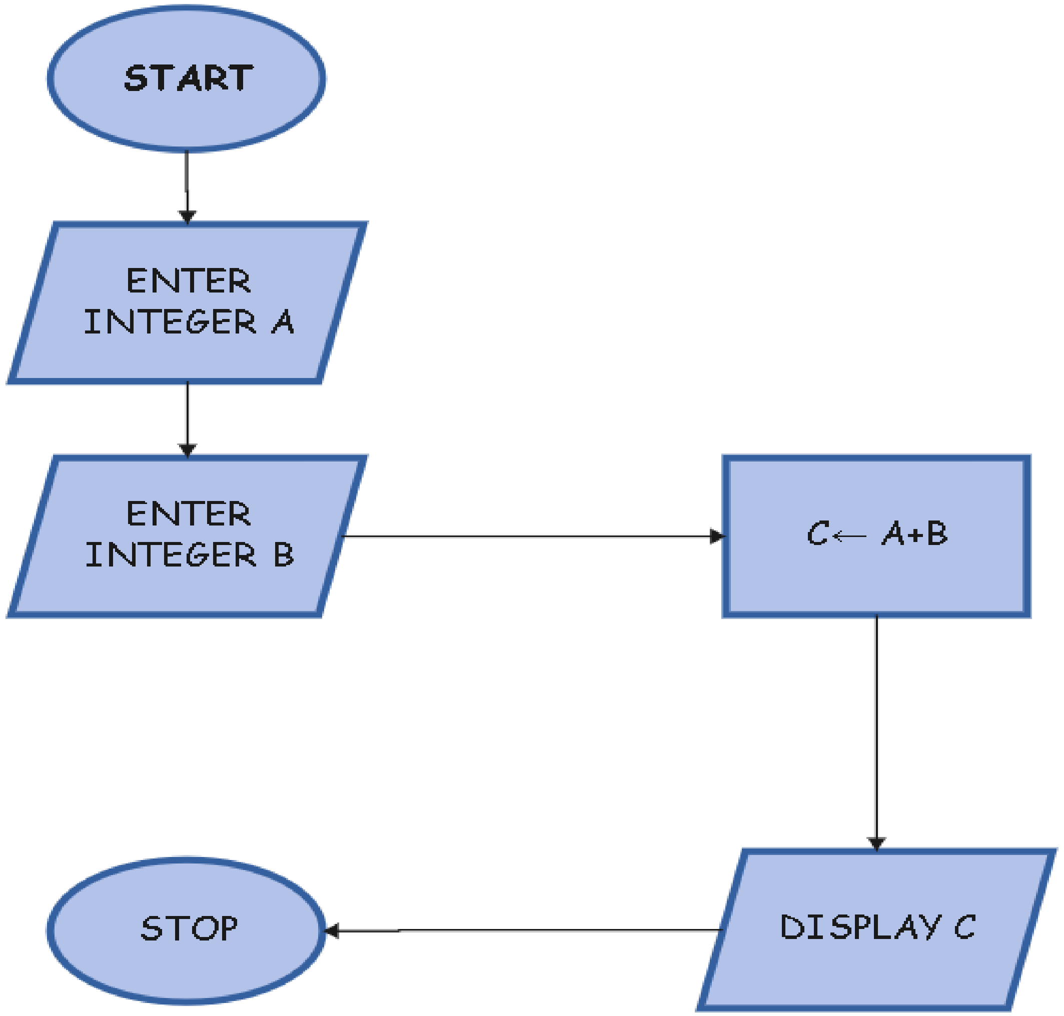
Different types of boxes are used to make flowcharts Symbols. All the different kinds of boxes are connected by arrow lines. Arrow lines are used to display the flow of control.

FLOWCHART

1. **Oval**: Represents the start or end of a process.
2. **Rectangle**: Represents a process or an action step.
3. **Diamond**: Represents a decision point where the flow can branch based on a yes/no or true/false question.
4. **Parallelogram**: Represents an input or output operation.
5. **Arrow**: Indicates the direction of the flow from one step to the next.

FLOWCHART

Example : Find the Sum of Two Numbers Entered



ALGORITHM

An algorithm is a step-by-step procedure or formula for solving a problem or completing a task. It is a sequence of instructions that can be followed to achieve a specific goal.

Basic Structure of an Algorithm :

- **Input**: The data or parameters required to start the process.
- **Process**: The sequence of steps or instructions to transform the input.
- **Output**: The result or outcome after applying the process to the input.

ALGORITHM

Example : Calculate the Sum of Two Numbers

1. Start
2. Input: Read two numbers (let's call them **‘a’** and **‘b’**)
3. Process: Calculate the sum of **‘a’** and **‘b’** (let’s call the sum ‘**sum**’)
4. Output: Display the value of **‘sum’**
5. End

INTRODUCTION TO C

- **C is a general-purpose programming language**: It is used for writing various types of software, from operating systems to complex applications.
- **Procedural Programming Language**: C follows a procedural paradigm, which means it focuses on procedures or functions to perform tasks.
- **Low-Level Language**: While C can be used to write high-level programs, it also provides low-level access to memory and hardware, making it ideal for system programming.

INTRODUCTION TO C

Features of C

- Simple and Efficient
- Rich Library
- Portability
- Low-Level Access
- Structured Language

PSEUDOCODE

- Pseudocode is a plain language description of the steps in an algorithm .
- Acts as a bridge between human logic and computer programming.
- Helps in planning and structuring code before actual implementation.
- Focuses on logic rather than syntax , making it language – agnostic.
- **Example** : Calculate the Sum of Two Numbers

```
START
  READ number1
  READ number2
  sum = number1 + number2
  PRINT sum
END
```

CHARACTER SET

The character set in C defines the set of valid characters that can be used in writing C programs.

Categories of Character in C :

Letters:

- Uppercase : ‘A – Z’
- Lowercase : ‘a – z’

Digits:

- Numbers : ‘0 – 9’

CHARACTER SET

Special Characters :

` ~ @ ! $ # ^ * % & ( ) [ ] { } < > + = _ – | / \ ; : ‘ “ , . ?

Whitespace Characters :

- Space : ‘ ‘
- Horizontal Tab : ‘\t’
- Newline : ‘\n’
- Carriage Return : ‘\r’

TOKENS

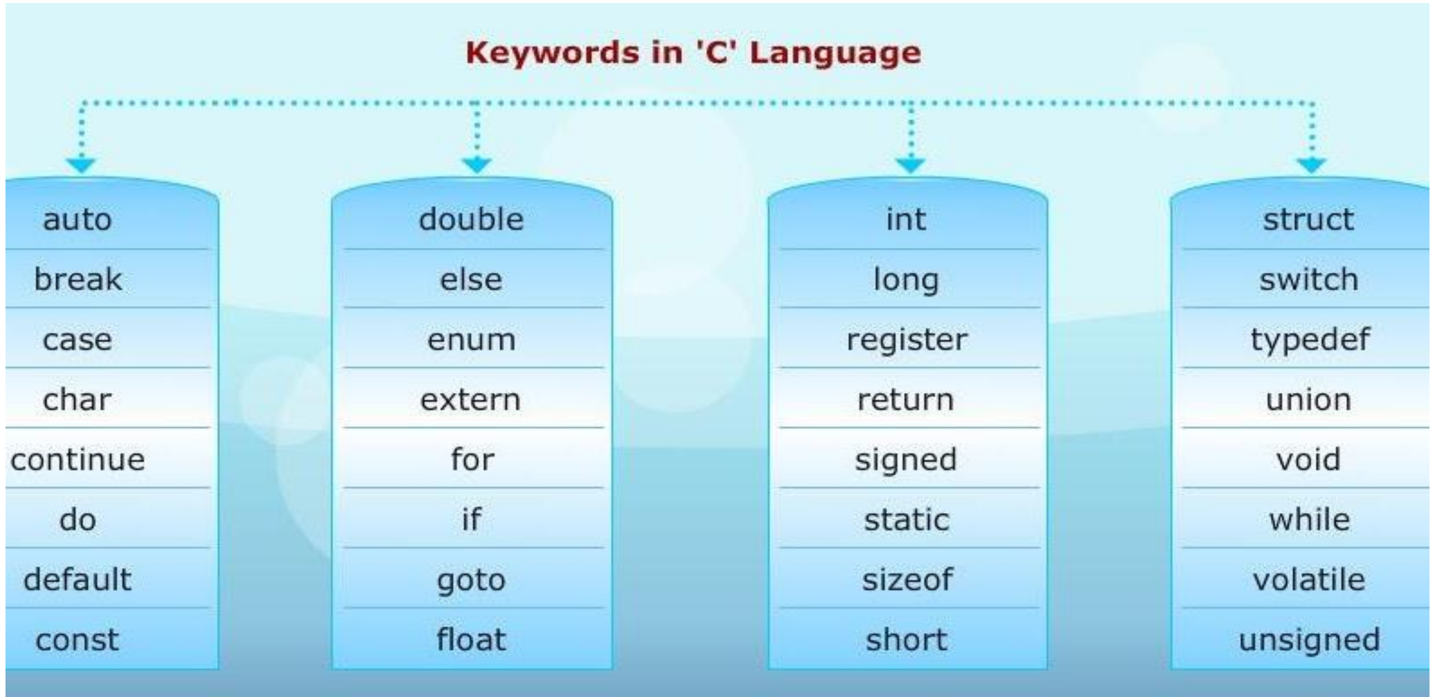
- Tokens are the smallest individual units in a C program.
- They are the basic building blocks used to write statements and expressions in a C program.

Types of Tokens :

1. Keywords
2. Identifiers
3. Constants
4. Strings
5. Operators
6. Special Symbols

TOKENS - KEYWORDS

- Reserved words in C that have a specific meaning to the compiler.
- Cannot be used as identifiers.
- All keywords are written in lowercase.
- In Total there are 32 keywords supported in C.



Keywords in 'C' Language

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

TOKENS - IDENTIFIERS

- Names given to variables, functions, arrays, etc., by the programmer.
- Rules for naming :
  - Must begin with a letter (uppercase or lowercase) or an underscore (‘_’).
  - Can contain letters, digits, and underscores.
  - Cannot use keywords as identifiers.
  - Case-sensitive
  - **Example** : ‘main’ , ‘_temp’ , ‘count123’ , ‘totalAmount’

TOKENS - CONSTANTS

- Fixed values that do not change during the execution of a program.
- Types of Constants :
- **Integer Constants:** Whole numbers (e.g, 10 , 21)
- **Floating-point Constants:** Numbers with a fractional part (e.g, 3.14 , 0.01)
- **Character Constants:** Single characters enclosed in single quotes (e.g, ‘A’ , ‘$’)
- **String Constants:** Sequence of characters enclosed in double quotes (e.g, “Hello” , “123”)

TOKENS - STRINGS

- A Sequence of characters enclosed in double quotes.
- Automatically ends with a null character (‘\0’).
- **Example** : “Hello World!” , “C Programming”

TOKENS - OPERATORS

- Symbols that perform operations on variables and values.
- **Types of Operators :**
    - Arithmetic Operators: ‘+’ , ‘-’ , ‘*’ , ‘/’ , ’%’
    - Relational Operators: ‘==‘ , ‘!=‘ , ‘>’ , ‘<‘ , ‘<=‘ , ‘>=‘
    - Logical Operators: ‘&&’ , ‘||’ , ‘!’
    - Assignment Operators: ‘=‘ , ‘+=‘ , ‘-=‘
    - Bitwise Operators: ‘&’ , ‘|’ , ‘^’ , ‘<<‘ , ‘>>’
    - Unary Operators: ‘++’ , ‘--’ , ‘+’ , ‘-’ , ‘!’
    - Conditional Operator: ‘? :’

TOKENS – SPECIAL SYMBOLS

- Special characters that have a specific meaning in C.
- **Examples and Uses:**
  - **Comma (,):** Separates variables in declarations .

    e.g, int a,b;
  - **Semicolon (;):** Terminates statements .

    e.g, int a = 10;
  - **Colon (:):** Used in labels and conditional expressions .

    e.g, default:
  - **Parentheses (()):** Used in function calls and expressions.

TOKENS – SPECIAL SYMBOLS

e.g, printf(“Hello;)

- **Braces ({}):** Define the body of functions and loops .

  e.g, if(condition) { /*code*/ }

- **Brackets ([]):** Used for array indexing .

  e.g, arr[5];

- **Pound Sign (#):** Used in preprocessor directives .

  e.g, #include<stdio.h>;

TOKENS

Example Program with Tokens

```
#include <stdio.h>  // Preprocessor directive
int main() {  // Function definition
  int a = 5, b = 10;  // Variable declarations (Identifiers, Constants, Operators)
  int sum = a + b;  // Expression (Identifiers, Operators)
  printf("Sum = %d\n", sum);   // Function call (Keywords, Identifiers, Strings,
   Operators)
  return 0;  // Return statement (Keyword, Constant)
}
```

DATA TYPES

- Data types in C define the type and size of data that a variable can hold.
- They are essential for memory allocation and to specify the operations that can be performed on the data.
- **Categories of Data Types in C :**
  1. Basic Data Types
  2. Derived Data Types

DATA TYPES – BASIC DATA TYPES

SAPTHAGIRI NPS UNIVERSITY

- The memory size of the basic data types may change according to 32 or 64-bit operating system.

1. **Integer Types:** Used to store whole numbers.

- **‘int’** : Typically 2 or 4 bytes, stores whole numbers (e.g, int age = 19;)
- **‘char’** :1 byte,stores single characters or small integers(e.g,char letter = ‘A’;)
- **‘short int’ or ‘short’** : Typically 2 bytes, smaller range of integers (e.g, short int temp = 100;)
- **‘long int’ or ‘long’** : Typically 4 or 8 bytes, larger range of integers (e.g, long int distance = 100000L;)

DATA TYPES – BASIC DATA TYPES

Signed vs Unsigned :

- **Signed:** Can represent both positive and negative values . e.g , int x = -5;
- **Unsigned:** Can only represent positive values, which allows for a larger positive range .e.g, unsigned int x=10;

Size and Range:

- **int:** Typically 4 bytes, range: '-2147483648 to 2147483647' (signed) or '0 to 4294967295' (unsigned).
- **short int :** Typically 2 bytes, range: '-32768 to 32767' (signed) or '0 to 65535' (unsigned).
- **long int:** Typically 8 bytes, range: '9223372036854775808 to 9223372036854775807' (signed) or '0 to 18446744073709551615 ' (unsigned).

DATA TYPES – BASIC DATA TYPES

2. **Floating-Point Types:** Used to store real numbers.

- **‘float’** : Typically 4 bytes, stores single-precision floating-point numbers

  e.g, float pi = 3.14f;

- **‘double’ :** Typically 8 bytes, stores double-precision floating-point numbers

  e.g, double e = 2.71828;

- **‘long double’** : Typically 10 or 16 bytes, stores extended precision floating-point numbers.

  e.g, long double bigPi = 3.141592653589793;

DATA TYPES – BASIC DATA TYPES

Precision Levels :

- **float:** Single precision, typically 6-7 decimal digits of precision.

  e.g, float temperature = 36.6f;

- **double:** Double precision, typically 15-16 decimal digits of precision.

  e.g, double mass = 5.972e24;

- **long double:** Extended precision, typically more than 19 decimal digits of precision.

  e.g, long double preciseValue = 2.718281828459045L;

DATA TYPES – DERIVED DATA TYPES

Derived from the basic data types to construct more complex data structures.

Types of Derived Data Types:

Arrays: Collection of elements of the same data type. e.g, int arr[10];

Pointers: Variables that store the memory address of another variable. e.g, int *ptr;

Structures: Collection of variables of different data types

e.g, struct Person { char name[50]; int age; };

Unions: Similar to structures but with shared memory for all members union Data.

e.g, {int i; float f; };

Functions: Blocks of code that perform a specific task and can return a value

e.g, int add(int a, int b);

DATA TYPES

Example Program Using Various Data Types

```
#include <stdio.h>
int main() {
    int age = 25;  // Integer data type
    float height = 5.9f;  // Float data type
    char grade = 'A';  // Character data type
    double pi = 3.141592653589793;  // Double data type
    printf("Age: %d\n", age);
    printf("Height: %.2f\n", height);
    printf("Grade: %c\n", grade);
    printf("Pi: %.15lf\n", pi);
    return 0;
}
```

TYPE CONVERSIONS

Type conversions in C refer to the process of converting a variable from one data type to another.

Types of Conversions:

- **Implicit Type Conversion (Automatic Type Conversion):** Handled automatically by the compiler.
- **Explicit Type Conversion (Type Casting):** Manually done by the programmer.

TYPE CONVERSIONS – IMPLICIT TYPE CONVERSION

The compiler automatically converts one data type to another when required also known as Automatic Type Conversion.

Example : Converting an ‘int’ to ‘float’ in arithmetic operations:

```
int a = 5;
float b = 6.2;
float result = a + b;  // a is automatically converted to float
```

Lower to Higher Precision: If operands of different types are involved, the lower precision type is promoted to the higher precision type.

Hierarchy**:** ‘char’ -> ‘int’ -> ‘float’ -> ‘double’ - > ‘long double’

TYPE CONVERSIONS – EXPLICIT TYPE CONVERSION

- The programmer explicitly converts one data type to another using casting.
- To avoid unwanted conversions and to explicitly define the type of a variable in operations.

Syntax : (data_type) expression

Example : Converting ‘double’ to ‘int’ in arithmetic operation :

```
double x = 9.99;
int y = (int) x + 10;  // Result is 19
```

LIBRARY FUNCTIONS IN C

Predefined functions provided by C's standard library to perform common tasks.

Advantages :

- Saves time by reusing tested and optimized code.
- Increases program readability and maintainability.
- Provide range of Functionalities.

Categories of Library Functions :

Input/Output Functions: Used for data input and output operations**.**

printf() , scanf() , getchar() , putchar() , gets() , puts().

String Handling Functions: Operate on character arrays or strings.

strlen() , strcpy() , strcat() , strcmp() , strstr() , strchr().

LIBRARY FUNCTIONS IN C

Mathematical Functions: Perform common mathematical operations.

sqrt() , pow() , floor()

Character Handling Functions: Manipulate or query character data.

Isalpha() , isdigit() , isalnum() , isupper() , islower() , toupper() , tolower()

Memory Allocation Functions: Manage dynamic memory allocation.

malloc() , calloc() , realloc() ,free()

Time-Related Functions: Deal with date and time operations.

time() , difftime() , strftime()

LIBRARY FUNCTIONS IN C

Example : Basic input and output operation:

```
#include <stdio.h>
int main() {
    int age;
    printf("Enter your age: ");
    scanf("%d", &age);
    printf("Your age is: %d\n", age);
    return 0;
}
```

LIBRARY FUNCTIONS IN C

Example : Performing mathematical operations.

```
#include <stdio.h>

#include <math.h>

int main() {

    double base = 2.0, exp = 3.0;

    double result = pow(base, exp);  // result is 8.0

    printf("2.0 raised to the power 3.0 is %.2f\n", result);

    return 0;

}
```

SAPTHAGIRI NPS UNIVERSITY

THANK YOU

Teja'S