

# Structured Programming using C

RCP2SFCES101

## Unit-I

### Introduction to Computer, Algorithm and Flowchart

# Contents

- 1 The Turing Machine architecture, Von Neumann architecture
- 2 Number System
  - Conversions Between Number Systems
- 3 Introduction to Operating System Components
- 4 System Software and application Software
- 5 Algorithm and Flowcharts
- 6 Compilation process: Syntax and semantic errors

## The Turing Machine architecture, Von Neumann architecture

# Computer

- A computer is an electronic device that can receive, process, store, and output data according to programmed instructions.
- It consists of hardware (physical components) and software (programs and operating instructions).
- **Hardware:** The physical components of a computer, including CPU, memory (RAM), storage devices, input devices, and output devices.
- **Software:** The set of programs and operating systems that tell the hardware what to do.
- The primary functions of a computer include: Input, Processing, Storage and Output.

# Turing Machine Architecture

## Turing Machine:

### Concept:

- Introduced by Alan Turing in 1936, the Turing machine is a theoretical model of computation used to understand the limits of what can be computed.
- It provides a formalized way to describe an abstract machine that manipulates symbols on a strip of tape according to a set of rules.

# Turing Machine Architecture

## Components:

- **Tape:** An infinite strip divided into cells, each of which can hold a symbol from a finite alphabet
- **Tape Head:** Reads and writes symbols on the tape and moves left or right.
- **State Register:** Stores the current state of the machine.
- **Transition Function:** Determines the next state, symbol to write, and direction to move based on the current state and symbol under the tape head.
- **Finite Control:** The "brain" of the machine, implementing the transition function and controlling operations.

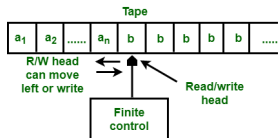


Figure: Turing Machine Model

# Von Neumann architecture

- The von Neumann architecture is a fundamental model for computer design, proposed by John von Neumann in the 1940s
- It consisted of a Control Unit, Arithmetic, and Logical Memory Unit (ALU), Registers and Inputs/Outputs.
- It is based on the stored-program computer concept, where instruction data and program data are stored in the same memory.

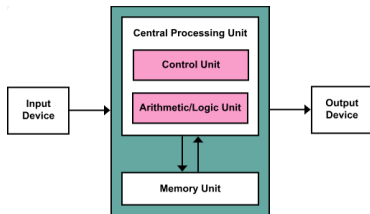


Figure: Von Neumann architecture

## Von Neumann architecture contd...

### Components:

#### Central Processing Unit (CPU):

- **Arithmetic Logic Unit (ALU):** Performs arithmetic operations (addition, subtraction) and logical operations (AND, OR).
- **Control Unit (CU):** Directs the operation of the CPU, including fetching instructions from memory, decoding them, and executing them. It also sends signals to other components to control their operations.

### Memory:

#### Main Memory (RAM):

- Stores data and instructions.
- It is organized in a linear address space, allowing both instructions and data to be accessed using addresses.
- This is also known as the "stored-program" concept.



## Von Neumann architecture contd...

### I/O Devices

- **Input Devices:** Devices like keyboards and mice that provide data to the computer.
- **Output Devices:** Devices like monitors and printers that output data from the computer.

### Bus System

- **Data Bus:** Carries data between the CPU, memory, and I/O devices.
- **Address Bus:** Carries the addresses of memory locations that are being read from or written to.
- **Control Bus:** Carries control signals to manage operations, such as read/write commands and interrupt signals.

# Number System

# Number System

- A Number System is a way to represent numbers using a set of **symbols (digits)** and **a base**.
- Each number system is defined by its **base or radix**, which indicates the number of unique digits, including zero, used to represent numbers.
- **Types of Number Systems:**
  - 1 Decimal Number System (Base-10)
  - 2 Binary Number System (Base-2)
  - 3 Octal Number System (Base-8)
  - 4 Hexadecimal Number System (Base-16)

## Number System contd...

### Decimal Number System (Base-10)

- **Digits Used:** 0 through 9 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- **Base:** 10
- **Application:** The most common system used in everyday arithmetic and counting.
- Each digit in a decimal number represents a power of 10.
- **Example:** Decimal number 8379 consists of the digit 9 in the unit's or one's position, 7 in the tens position, 3 in the hundreds position, and 8 in the thousands position.

Its value can be formulated as:

$$(8 \times 1000) + (3 \times 100) + (7 \times 10) + (9 \times 1)$$

$$(8 \times 10^3) + (3 \times 10^2) + (7 \times 10^1) + (9 \times 10^0)$$

$$8000 + 300 + 70 + 9$$

$$8379$$

## Number System contd...

### Binary Number System (Base-2)

- **Digits Used:** 0 and 1
- **Base:** 2
- **Application:** The foundation of all modern digital and computing systems.
- Each digit in a Binary number represents a power of 2.
- **Example:** 1101 in binary is equal to 13 in decimal.

## Number System contd...

### Octal Number System (Base-8)

- **Digits Used:** 0 through 7 (0, 1, 2, 3, 4, 5, 6, 7)
- **Base:** 8
- **Application:** Often used in computing as a shorthand for binary.
- Every digit in a Octal number represents a power of 8.
- **Example:** The octal number 57 is equivalent to the decimal number 47.

## Number System contd...

### Hexadecimal Number System (Base-16)

- **Digits Used:** 0 through 9 and A through F (where A = 10, B = 11, C = 12, D = 13, E = 14, F = 15)
- **Base:** 16
- **Application:** Used in computer science and digital electronics for memory addresses, color codes in web design, and as a shorthand for binary data.
- Every digit in a Hexadecimal number represents a power of 16.
- **Example:** The hexadecimal number 2F3 represents the decimal number 755.

# Decimal to Binary

## Steps:

- 1 Divide the decimal number by 2.
- 2 Record the remainder.
- 3 Update the decimal number with the quotient.
- 4 Repeat steps 1-3 until the quotient becomes 0.
- 5 The binary equivalent is the remainders read from bottom to top.

**Example:** Convert  $(13)_{10}$  to binary

	Quotient	Remainder
$13 \div 2$	6	1
$6 \div 2$	3	0
$3 \div 2$	1	1
$1 \div 2$	0	1



Reading the remainders from bottom to top, the binary representation of **13** is **1101**




# Decimal to Octal

## Steps:

- 1 Divide the decimal number by 8.
- 2 Record the remainder.
- 3 Update the decimal number with the quotient.
- 4 Repeat steps 1-3 until the quotient becomes 0.
- 5 The octal equivalent is the remainders read from bottom to top.

**Example:** Convert  $(127)_{10}$  to octal

	Quotient	Remainder
$127 \div 8$	15	7
$15 \div 8$	1	7
$1 \div 8$	0	1



Reading the remainders from bottom to top, the octal representation of  $(127)_{10}$  is  $(177)_8$

# Decimal to Hexadecimal

## Steps:

- 1 Divide the decimal number by 16.
- 2 Record the remainder (0-15, with 10-15 represented as A-F).
- 3 Update the decimal number with the quotient.
- 4 Repeat steps 1-3 until the quotient becomes 0.
- 5 The hexadecimal equivalent is the remainders read from bottom to top.

**Example:** Convert  $(945)_{10}$  to Hexadecimal

	Quotient	Remainder
$945 \div 16$	59	1
$59 \div 16$	3	B (11)
$3 \div 16$	0	3



Reading the remainders from bottom to top, the octal representation of  $(945)_{10}$  is  $(3B1)_{16}$

## Binary to Decimal

### Steps:

- 1 Multiply each bit by 2 raised to the power of its position, starting from 0 on the right.
- 2 Sum all the values.

**Example:** Convert  $(101101)_2$  to Decimal

$$\begin{aligned}1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\= 32 + 0 + 8 + 4 + 0 + 1 \\= 45\end{aligned}$$

## Octal to Decimal

### Steps:

- 1 Multiply each digit by 8 raised to the power of its position, starting from 0 on the right.
- 2 Sum all the values.

**Example:** Convert  $(365)_8$  to Decimal

$$\begin{aligned} & 3 \times 8^2 + 6 \times 8^1 + 5 \times 8^0 \\ &= 3 \times 64 + 6 \times 8 + 5 \times 1 \\ &= 192 + 48 + 5 \\ &= 245 \end{aligned}$$

## HexaDecimal to Decimal

### Steps:

- 1 Multiply each digit by 16 raised to the power of its position, starting from 0 on the right.
- 2 Sum all the values.

**Example:** Convert  $(2F2)_{16}$  to Decimal

$$\begin{aligned} & 2 \times 16^2 + 15 \times 16^1 + 2 \times 16^0 \\ &= 2 \times 256 + 15 \times 16 + 2 \\ &= 512 + 240 + 2 \\ &= 754 \end{aligned}$$

# Introduction to Operating System Components

# Introduction to Operating System Components

## Operating System :

- An operating system is a software that acts as an interface between the user of a computer and the computer hardware.
- It provides an environment in which a user may execute programs.
- An OS is an important part of almost every computer system.

# Operating System Components

## 1. Kernel:

- Kernel is a core part of the operating system and is loaded on the main memory when it starts up.
- It is responsible for managing system resources, including the CPU, memory, and devices.
- It operates in the background, managing the hardware-software interaction.

## Functions of Kernel:

- To provide a mechanism for the creation and deletion of processes
- To provide CPU scheduling, memory management, and device management for these processes
- To provide synchronization tools so that the processes can synchronize their actions
- To provide communication tools so that processes can communicate with each other



# Operating System Components

## 2. Command Interpreter (Shell):

- The command interpreter, often referred to as the shell, is the interface that allows users to interact with the operating system.
- It interprets user commands and converts them into actions that the kernel can execute.

### Functions of Command Interpreter (Shell):

- **User Interface:** Provides a command-line interface (CLI) where users can enter text commands.
- **Command Execution:** Translates user commands into actions performed by the kernel, such as file operations or process control.
- **Scripting:** Allows users to write scripts that automate tasks by executing a sequence of commands.

## System Software and application Software

# Software

- Software can be described as a set of related programs.
- Types of software
  - 1 System Software
  - 2 Application Software

# System Software

- System software is a set of computer programs, which is designed to manage system resources.
- It is responsible for running and smooth functioning of your computer system with other hardware.
- System software runs and functions internally with application software and hardware.
- It is interface between a hardware device and the end-user.
- **Examples:** Operating system, loader, linker etc.
- System Software is usually developed using Low-level language such as Assembly language.
- Functions of System software are:
  - Disk Management
  - Memory Management
  - Device controlling
  - Loading and execution of other programs.

# Application Software

- It is a type of software that is mainly developed to perform a specific task as per the user's request.
- It acts as an interface between the end-user and system software.
- **Examples:** Microsoft Word, Microsoft Excel, Paint, Web-browser etc.
- Application software is usually developed with the help of High-level languages such as Java.
- Functions of application software are given below:
  - Data Manipulation
  - Writing Reports
  - Creating Spreadsheets
  - Managing records.

# System Software VS application Software

Sr. No.	System Software	Sr. No.2	Application Software
1	System s/w are designed to manage system resources	1	Application s/w are designed to perform a specific task
2	A Computer can not run without system s/w	2	A Computer can easily run without application s/w
3	System s/w do not depends on application s/w	3	Allication s/w depends on system s/w and can not run without it.
4	It is interface between a hardware device and the end-user.	4	It acts as an interface between the end-user and system software.
5	Developed using Low-level language such as Assembly language	5	Developed with the help of High-level languages such as Java.
6	<b>Examples:</b> Operating system, loader, linker etc.	6	<b>Examples:</b> Microsoft Word, Microsoft Excel, Paint, Web-browser etc.

# Algorithm and Flowcharts

# History of C

- The C programming language was developed by **Dennis Ritchie** at **Bell Laboratories** (AT&T Bell Labs) in **1972**.
- The C language was standardized by the American National Standards Institute (ANSI) in 1989, resulting in the ANSI C standard, also known as C89.
- C was designed to be a general-purpose programming language that could be used for system software
- C was originally developed for the Unix operating system and played a crucial role in its implementation.



# Algorithm

**Definition:** An algorithm is well defined, finite set of computational instructions that accomplishes a particular task, which may or may not take *inputs* and produces some value or a set of values as *output*.

All algorithms must satisfy the following criteria:

- Zero or more quantities are externally supplied: *Input*
- At least one quantity is produced: *Output*
- Each instruction is clear and unambiguous: *Definiteness*
- The algorithm terminates after a finite number of steps: *Finiteness*
- For every input instance, it halts with the correct output: *Correct*

# Algorithm

## Example: Algorithm for sum of two integer numbers

This algorithm computes the addition of two numbers.

The variables used are:

**a, b** : type integer

**sum** : storing the result of addition, type integer

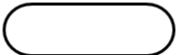
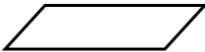



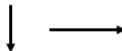
- Step 1: [Input the two integers]  
    read a, b
- Step 2: [Compute the addition of two numbers]  
     $\text{sum} = a + b$  ;
- Step 3: [Write the sum]  
    write (sum)
- Step 4: [Finished]  
    exit

# Flowchart

- Flowchart is a graphical representation of an algorithm.
- It uses various symbols to show the operations and decisions to be followed in a program.
- Basic elements of Flowchart:
  - Terminal(Start/Stop)
  - Input/Output
  - Process
  - Decision
  - Connector
  - Flow Lines

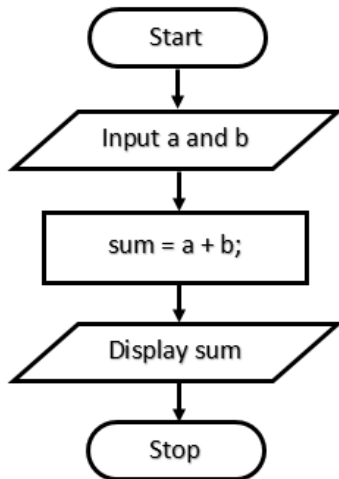
# Flowchart

## Basic elements of Flowchart:

Symbol Name	Symbol	Description
Terminal (Start / Stop)		Represents the beginning or end of the flowchart
Input / Output		Represents input or output operations, such as reading data from the user or displaying results.
Process		Indicates a process or operation, such as a calculation, assignment, or initialization in the program.
Decision		Represents a decision point in the flowchart, where the flow can branch based on a condition.
Connector		Used to connect different parts of the flowchart, often when the flowchart is spread over multiple pages.
Flow Lines		Indicates the direction of the flow of control from one element to the next.

# Flowchart

## Example: Flowchart for sum of two integer numbers



# Three constructs of algorithm and flowchart

## Sequence:

- Sequence means each step or process in the algorithm is executed in the specified order
- Each step is carried out exactly once, and then the next step in the sequence is executed.
- Each step or process must be in the proper place otherwise the algorithm will fail.
- A sequence is represented by a series of connected rectangles, each containing a single statement or operation.
- The “Sequence” can be thought of as “do this, then do this, then do this”

# Three constructs of algorithm and flowchart

## Decision (Selection):

- In algorithms the outcome of a decision is either true or false.
- The decision construct allows branching based on conditions.
- It enables the program to choose different paths of execution depending on whether a condition is true or false.
- Decision logic is depicted as either an IF...THEN...ELSE or IF...THEN structure
- Decision can be thought of as “if something is true, take this action, otherwise take that action”
- It is typically represented in flowcharts with a diamond-shaped decision box.

# Three constructs of algorithm and flowchart

## **Repetition (Looping or Iteration):**

- The repetition construct involves executing a block of code multiple times, as long as a certain condition is true.
- Repetition is a looping construct
- Repetition is a combination of decision and sequence and can repeat steps
- Repetition can be thought of as “while something is true, do this, otherwise stop”



## Compilation process: Syntax and semantic errors

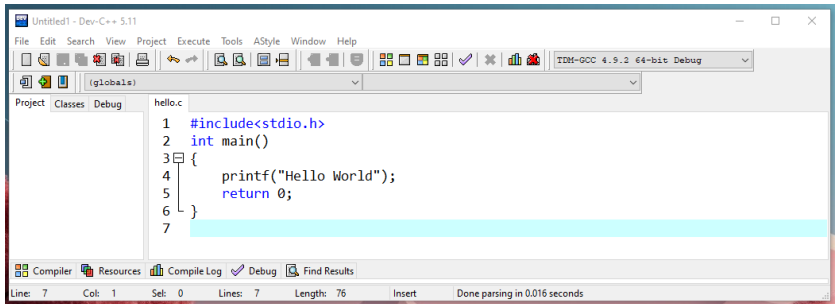
# Steps to develop and execute program in C

There are three steps to develop and execute program in C:

- 1 Writing the C program
- 2 Compiling the program
- 3 Executing the program

# 1. Writing the C program

This involves writing a new program code or editing an existing source program using a text editor or an IDE and saving it with .c extension.



The screenshot shows the Dev-C++ 5.11 IDE interface. The title bar reads 'Untitled1 - Dev-C++ 5.11'. The menu bar includes File, Edit, Search, View, Project, Execute, Tools, AStyle, Window, and Help. The toolbar contains icons for file operations, editing, and execution. The compiler is set to 'TDM-GCC 4.9.2 64-bit Debug'. The project explorer on the left shows a project named '(globals)'. The main editor window displays a C program named 'hello.c' with the following code:

```
1 #include<stdio.h>
2 int main()
3 {
4     printf("Hello World");
5     return 0;
6 }
7
```

The status bar at the bottom shows 'Line: 7 Col: 1 Sel: 0 Lines: 7 Length: 76 Insert Done parsing in 0.016 seconds'.

## 2. Compiling the program

The compilation process in C involves following stages that convert the source code written in `.c` file into an executable program `.exe`.

- 1 Preprocessing
- 2 Compilation
- 3 Assembly
- 4 Linking

Each stage must complete successfully for the process to move to the next, culminating in an executable program that can be run on the target machine.

## 2. Compiling the program contd...

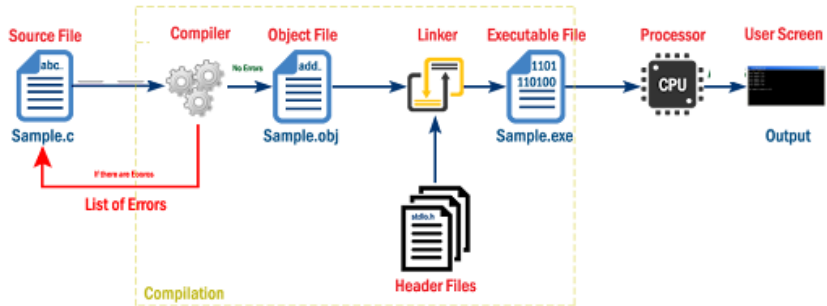


Figure: Execution of a C program

## 2. Compiling the program contd...

### 1. Pre-processing:

- The source code (written in **.c file**) is first passed through the preprocessor.
- The preprocessor handles directives such as `#include`, `#define`, and conditional compilation (`#ifdef`, `#endif`).
- It generates an expanded source code file by including the contents of header files, replacing macros, and resolving other preprocessor directives.
- The output is usually an intermediate file (**.i file**).

### 2. Compilation:

- The compiler translates the preprocessed source code into assembly code (**.s file**) specific to the target machine.
- Syntax and semantic checks are performed during this stage.
- Errors detected here prevent the process from proceeding.

## 2. Compiling the program contd...

### 3. Assembly:

- The assembler translates the assembly code into machine code, producing an object file (**.o or .obj file**).
- The object file contains the machine code and data that will eventually be executed by the computer, but it is not yet a complete executable.

### 4. Linking:

- The purpose of the linking phase is to get the program into a final form for execution on the computer.
- The linker combines one or more object files, along with any libraries, into a final executable file (**.exe or .out**).
- During this stage, the linker resolves references between different object files and external libraries, ensuring that all function calls and variable references are correctly connected.

## 2. Compiling the program contd...

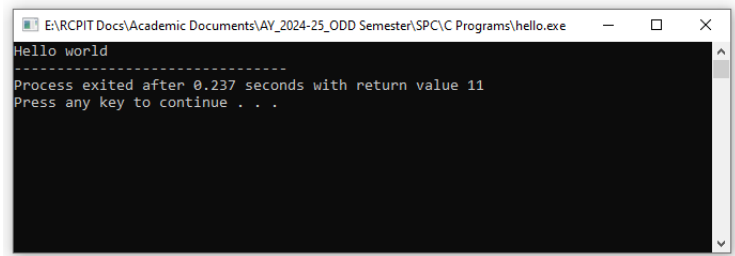
The final output of compilation process is an executable file that can be run on the target machine.

```
E:\RCPIT Docs\Academic Documents\AY_2024-25_ODD Semester\SPC\SPC Practical Programs\hello.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project Classes Debug hello.c
1 #include<stdio.h>
2 int main()
3 {
4     printf("Hello World");
5     return 0;
6 }
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
Compilation results...
- Errors: 0
- Warnings: 0
- Output Filename: E:\RCPIT Docs\Academic Documents\AY_2024-25_ODD Semester\SPC\SPC Practical Programs\hello.exe
- Output Size: 148.16796875 KiB
- Compilation Time: 0.22s
Line: 7 Col: 1 Sel: 0 Lines: 7 Length: 76 Insert Done parsing in 0.016 seconds
```



### 3. Executing the program

- When the program is executed, each of the statements of the program is sequentially executed
- If the program requests any data from the user, known as **input**, the program temporarily suspends its execution so that the input can be entered.
- Results that are displayed by the program, known as **output**, appear in a window, sometimes called the **console**



The screenshot shows a Windows command prompt window titled "E:\RCPIT Docs\Academic Documents\AY\_2024-25\_ODD Semester\SPC\C Programs\hello.exe". The window has a black background with white text. The output of the program is as follows:

```
Hello world
-----
Process exited after 0.237 seconds with return value 11
Press any key to continue . . .
```

# Syntax and semantic errors

## Errors:

- Errors in a C program refer to issues or problems that occur in the code, preventing it from compiling or running correctly.
- These errors need to be identified and corrected to ensure that the program behaves as expected.
- Programming errors are also known as the bugs or faults, and the process of removing these bugs is known as **debugging**.
- These errors are detected either during the time of compilation or execution.

# Syntax and semantic errors

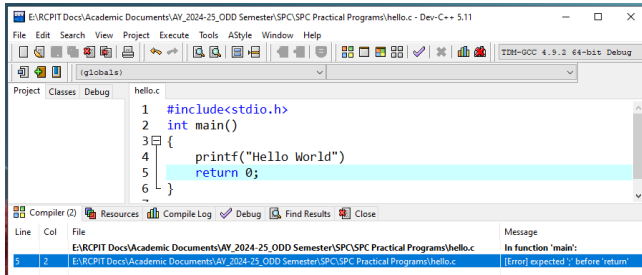
## Types of Errors:

Following are the primary types of errors encountered in C programming:

- **Syntax Errors**
- Runtime Errors
- Linker Errors
- Logical Errors
- **Semantic Errors**

## Syntax errors

- Syntax errors are also known as the compilation errors as they occurred at the compilation time, or we can say that the syntax errors are thrown by the compilers.
- These errors are mainly occurred due to the mistakes while typing or do not follow the syntax of the specified programming language.
- **Examples:**
  - Missing semicolon at the end of a statement.
  - Mismatched parentheses, braces, or brackets.
  - Incorrectly formed statements or declarations.



The screenshot shows the Dev-C++ IDE with a C program named 'hello.c' open. The code is as follows:

```
1 #include<stdio.h>
2 int main()
3 {
4     printf("Hello World")
5     return 0;
6 }
```

The line containing 'return 0;' is highlighted in light blue. At the bottom of the IDE, the 'Compiler (2)' window displays an error message:

Line	Col	File	Message
5	2	E:\RCPIIT Docs\Academic Documents\AY_2024-25_ODD Semester\SPC\SPC Practical Programs\hello.c	In function 'main': [Error] expected ';' before 'return'

# Semantic errors

- Semantic errors occur when the statements in the code are syntactically correct but have incorrect meanings or logic.
- The compiler might not catch certain semantic errors, which can lead to unexpected behavior when the program is executed.
- **Examples:**
  - Using an uninitialized variable.
  - Passing the wrong number or types of arguments to a function.
  - Logic errors, like incorrect conditions in **if** statements.

*Thank  
you*

