

Unit 1

Problem Solving with Computer

Problem Analysis

- Problem analysis is an important part of software development.
- It involves a detailed analysis of the problem to be solved, and the requirements and constraints that the solution must meet.
- The following are the steps involved in problem analysis:
 - i) *Requirement analysis:* This involves identifying the functional and non-functional requirements of the system. This step involves talking to stakeholders, users, and subject matter experts to understand what they need the system to do.
 - ii) *Program design:* Once the requirements have been identified, the next step is to design the program. This involves creating a detailed design that outlines the software's architecture, algorithms, data structures, and interfaces.
 - iii) *Program coding:* Once the design has been created, the actual coding of the software can begin. This involves writing the code in a programming language such as C, C++, Java, or Python.
 - iv) *Program testing:* After the code has been written, the program must be tested to ensure that it meets the requirements and is free of bugs. This involves creating test cases and running them to ensure that the program behaves as expected.
 - v) *Software installation:* Once the program has been tested and approved, it can be installed on the target system. This involves creating an installer or package that can be distributed to users.
 - vi) *Software maintenance:* Finally, software maintenance is required to ensure that the program continues to function correctly over time. This involves fixing bugs, updating the software to meet changing requirements, and ensuring that it remains compatible with other systems and software.

Coding, compilation, and execution

Coding, compilation, and execution are essential steps in developing and running a C program.

Coding:

- Coding involves writing the source code for the program using a text editor or an integrated development environment (IDE).
- C code is written in a plain text format with a .c file extension.
- The code is written in a specific syntax and structure that the C compiler can understand.

Compilation:

- Compilation is the process of translating the source code into machine code that the computer can understand and execute.
- The C compiler reads the source code, checks it for syntax errors, and converts it into an object file with a .o extension.

- The object file contains the compiled code, but it is not yet executable.

Linking:

- Linking is the process of combining the object files with any libraries that the program requires to create an executable file.
- The linker takes the object files and resolves the references to the external functions and variables.
- The linker then generates an executable file with a .exe extension.

Execution:

- Execution is the final step in running a C program.
- To execute the program, the user runs the executable file by double-clicking on it or by typing its name in the terminal.
- The program then loads into memory and starts running.
- During execution, the program uses the computer's resources to perform its intended tasks.
- When the program finishes running, it returns control to the operating system.

In summary, coding is the process of writing the C program in a text editor or IDE, compilation is the process of translating the source code into machine code, linking is the process of combining the object files with libraries to create an executable file, and execution is the process of running the program to perform its intended tasks.

Debugging, Testing and Documentation

- Debugging, testing, and documentation are essential steps in the software development process that ensure the quality and reliability of the software system.
- Errors can occur during the software development process, and it's important to understand the different types of errors and how to identify and fix them.

Debugging:

- Debugging is the process of identifying and fixing errors in the software code.
- There are three types of errors that can occur during the software development process: compiler errors, linker errors, and runtime errors.
 - Compiler errors:* Compiler errors occur when the C compiler detects an error in the source code during compilation. These errors are usually caused by syntax errors or semantic errors. The C compiler displays an error message that describes the nature of the error, the line number where the error occurred, and sometimes a suggested fix.
 - Linker errors:* Linker errors occur when the linker is unable to resolve external symbols that the program references. These errors can be caused by missing or improperly linked libraries, undefined functions or variables, or conflicting library versions. The linker displays an error message that describes the nature of the error and the library or symbol that is causing the error.

- iii) *Runtime errors:* Runtime errors occur when the program is executing and encounters an error that causes it to crash or behave unexpectedly. These errors can be caused by a variety of factors, including invalid input data, memory allocation errors, or division by zero. Debugging runtime errors can be challenging because the error may not occur consistently or predictably. Developers use debugging tools like breakpoints, step-through execution, and memory dump analysis to identify and fix runtime errors.

Testing:

- Testing is the process of evaluating the software system to ensure that it meets the requirements and performs as expected.
- There are several types of testing, including unit testing, integration testing, system testing, and acceptance testing.
- Testing can be manual or automated, and it is an iterative process that ensures that the software system is functional and reliable.

Documentation:

- Documentation is the process of creating user manuals, technical specifications, and other written materials that describe the software system.
- Documentation is important because it helps users understand how to use the software system and helps developers understand how the system is designed and implemented.
- Good documentation is clear, concise, and up-to-date, and it includes diagrams, examples, and code snippets to help users and developers understand the software system.

History of C

- C is a general-purpose programming language that was developed in the early 1970s by Dennis Ritchie at Bell Labs.
- It was designed as a successor to the B language, which was itself based on the BCPL language.
- One of the main goals of C was to create a programming language that was portable, efficient, and flexible.
- This made it ideal for systems programming, where performance and low-level access to hardware are important.
- In 1978, the first edition of "The C Programming Language" by Brian Kernighan and Dennis Ritchie was published, which quickly became the definitive reference for the language.
- C gained popularity in the 1980s, especially after the release of the Unix operating system, which was written in C.
- The language was used to write many of the tools and utilities that made Unix popular among developers and system administrators.
- C also played an important role in the development of the personal computer industry, as it was the primary language used to write operating systems and applications for early PCs.
- In the 1990s, the C language underwent several revisions, including the release of ANSI C (also known as C89) in 1989 and the later release of C99 in 1999.

- These revisions standardized the language and introduced new features, such as support for variable-length arrays and inline functions.
- Today, C is still widely used for systems programming, embedded systems, and high-performance computing.
- It has also influenced many other programming languages, including C++, Java, and Python.

C Standards (ANSI C and C99)

ANSI C

- ANSI C, also known as C89, was the first standardized version of the C programming language, published by the American National Standards Institute (ANSI) in 1989.
- It was based on the original K&R C language specification from 1978, with some minor additions and changes.
- One of the main goals of ANSI C was to standardize the language, making it more portable and ensuring that programs written in C could be easily compiled on any compliant implementation of the language.
- The ANSI C standard also introduced some new features to the language, including function prototypes, new types of data, and new library functions.

C99

- C99, or the 1999 standard for the C programming language, was the second standardized version of C, published by the International Organization for Standardization (ISO) in 1999.
- C99 added several new features to the language, including support for inline functions, variable-length arrays, flexible array members, and additional data types such as the long long int and complex numbers.
- C99 also introduced new library functions, including functions for working with complex numbers and for manipulating strings, as well as new macros for working with files and directories.
- One of the main goals of C99 was to address some of the shortcomings of ANSI C, such as the lack of support for new data types and the lack of features for working with complex data structures.
- C99 was also designed to make the language more expressive and easier to use, while maintaining compatibility with existing code.

Today, many compilers support both ANSI C and C99, and the choice of which standard to use depends on the specific requirements of the project and the capabilities of the compiler being used.

Algorithm

- Algorithm is the set of rules that define how particular problem can be solved in finite number of steps.
- Any good algorithm must have following characteristics

- i) *Input*: Specify and require input
- ii) *Output*: Solution of any problem
- iii) *Definite*: Solution must be clearly defined
- iv) *Effective*: be adequate
- v) *Finite*: Steps must be finite
- vi) *Correct*: Correct output must be generated

Advantages of Algorithms:

- ✓ It is the way to solve a problem step-wise so it is easy to understand.
- ✓ It uses definite procedure.
- ✓ It is not dependent with any programming language.
- ✓ Each step has its own meaning so it is easy to debug.

Disadvantage of Algorithms:

- ✓ It is time consuming
- ✓ Difficult to show branching and looping statement
- ✓ Large problems are difficult to implement

Flowchart

- The solution of any problem in picture form is called flowchart.
- It is the one of the most important technique to depict an algorithm.







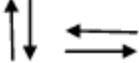
Advantage of Flowchart:

- ✓ Easier to understand
- ✓ Helps to understand logic of problem
- ✓ Easy to draw flowchart in any software like MS-Word
- ✓ Complex problem can be represent using less symbols
- ✓ It is the way to documenting any problem
- ✓ Helps in debugging process

Disadvantage of Flowchart:

- ✓ For any change, Flowchart have to redrawn
- ✓ Showing many looping and branching become complex
- ✓ Modification of flowchart is time consuming

Symbol Used in Flowchart

Symbol	Name	Function
	Process	Indicates any type of internal operation inside the Processor or Memory
	input/output	Used for any Input / Output (I/O) operation. Indicates that the computer is to obtain data or output results
	Decision	Used to ask a question that can be answered in a binary format (Yes/No, True/False)
	Connector	Allows the flowchart to be drawn without intersecting lines or without a reverse flow.
	Predefined Process	Used to invoke a subroutine or an Interrupt program.
	Terminal	Indicates the starting or ending of the program, process, or interrupt program
	Flow Lines	Shows direction of flow.

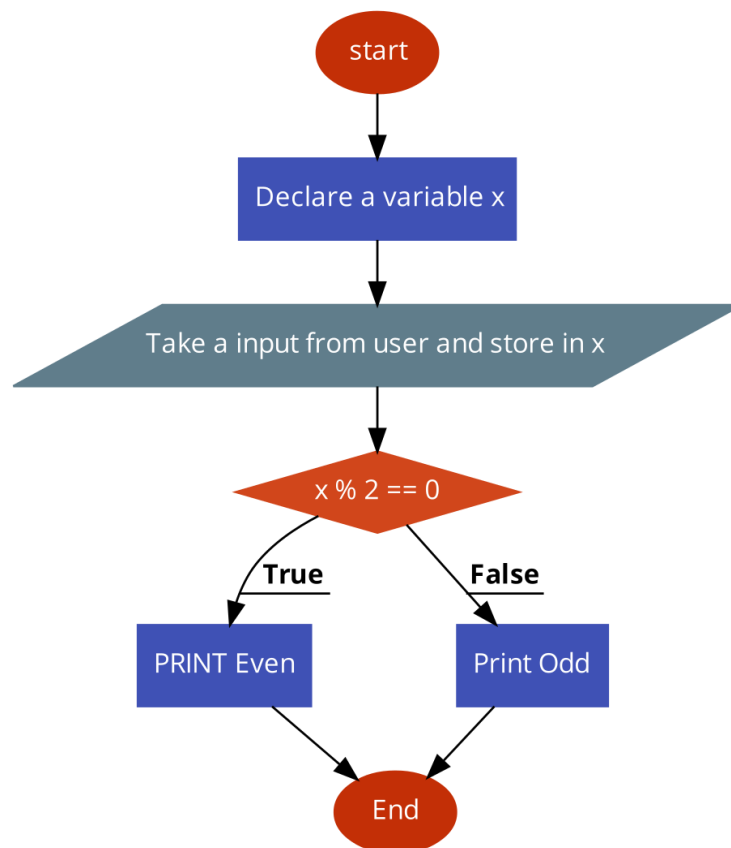
Example: Algorithm and Flowchart to check odd or even

Solution:

Algorithm:

Step 1: Start
Step 2: Declare a variable x
Step 3: Take an input from user and store in x
Step 4: IF $x \% 2 == 0$ THEN
 PRINT Even
 ELSE
 PRINT Odd
Step 5: End

Flowchart:



Preprocessor directive, Header files and library files in C

Preprocessor directive

- In C programming, a preprocessor directive is a statement that begins with the hash symbol (#) and is executed before the program is compiled.
- The preprocessor directives are not part of the C language syntax but are used to perform tasks like file inclusion, macro expansion, conditional compilation, and more.
- Two of the most commonly used preprocessor directives are #include and #define.

#include

- The #include directive is used to include a header file in the source code.
- A header file contains declarations of functions, variables, and other symbols that are defined in another source file.
- The #include directive copies the contents of the specified header file into the source code, allowing the program to use the declarations and definitions contained in the header file.
- Here's an example of the #include directive:

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

- This directive includes the header file "stdio.h" in the source code.
- The "stdio.h" header file contains declarations for input and output functions like printf and scanf.

#define

- The #define directive is used to create macros, which are short names that represent a piece of code or a value.
- Macros can be used to make the code more readable or to avoid repetition of code.
- Here's an example of the #define directive:

```
#define PI 3.14159
```

- This directive defines a macro PI that represents the value 3.14159.

Header files

- Header files are files that contain declarations of functions, variables, and other symbols that are defined in another source file.
- They are included in the source code using the #include directive.
- The C standard library provides many header files that define functions, types, and macros for common operations.

Library files

- Library files are collections of compiled object files that are used by a program to provide additional functionality.
- A library file can contain object files for one or more functions, and these files can be linked to the program during compilation.
- The C standard library provides many library files that provide functions for common operations, like input/output, string manipulation, and mathematical operations.

Exercise

1. What is algorithm? How is it different from flow chart? (5) [TU 2074]
2. What is preprocessor directives? Discuss # define directive with example. (5) [TU 2075]
3. Explain flowchart with example. What are the benefits of using flowcharts? (5) [TU 2077]
4. Compilation and Execution (2.5) [TU 2077]
5. History of C (2.5) [TU 2078]