INTRODUCTION TO PROGRAMMING AND PROBLEM SOLVING

WEEK 2: Problem Solving Strategies

LECTURER: LEMI AGREY OLIVER

KUMI UNIVERSITY

Recap of the previous

- In the previous session, we discussed digital computers and learned about their components. We highlighted that a computer consists of both hardware and software components. Specifically, we delved into a detailed discussion of these components, which encompass input devices, output devices, the motherboard, processing units, memory, and many other vital elements.

Recap of the previous+

- We also explored the concept of memory hierarchy, where we divided it into several layers, including registers, cache memory, main memory (often referred to as primary memory), secondary memory (such as hard drives), and tertiary memory (like offline storage devices).
- Additionally, we discussed the terms 'cache hit' and 'cache miss.' A 'cache hit' occurs when the CPU successfully retrieves a file from the cache memory, while a 'cache miss' signifies that the CPU couldn't find the file in the cache memory and had to search for it in the main memory.

Recap of the previous++

- Our discussions further extended to the von Neumann architecture, an essential concept in computer science that describes the fundamental structure of modern digital computers.
- Moving on, we explored the three primary categories of software: system software, utility software, and application software. Each of these plays a distinct role in the functioning of a computer system.

Recap of the previous+++

- We then proceeded to delve into the realm of programming languages, emphasizing that they are the tools used for solving problems and instructing computers to perform specific tasks.
- This week, our focus shifts towards understanding the concept of problem-solving and its various facets. I encourage you to stay engaged throughout this week's lecture and in the upcoming sessions. Your commitment to learning will undoubtedly enrich your understanding of the subject matter.

Introduction to Problem Solving

What is a problem ?

- A Problem refers to undesirable situation or set of circumstances that causes discomfort, pain or unhappiness and often requires a solution or resolution
- A problem can be defined as a gap between the actual and desired conditions. It is an unfulfilled customer need [4]

  Problem can affect our personal, professional, academic life or even can be technical in nature
- And is also important to note that problem can either be complex or simple and may have various characteristics.

Features of Problems

The following are some of the common features of problems

- **Obstacle or Challenge:** A problem typically represents an obstacle, challenge, or barrier that stands in the way of achieving a desired goal or outcome.
- **Undefined or Undesirable State:** Problems often involve a situation that is undefined, undesirable, or not in alignment with one's objectives, expectations, or standards.

Features of Problems+

- **Need for Resolution,** Problems necessitate some form of resolution or action to improve the situation, overcome the challenge, or reach a more favorable state.
- **Complexity,** Problems can vary in complexity. Some are simple and straightforward, while others are intricate and multifaceted, requiring careful analysis and consideration.
- **Subjectivity,** The perception of a problem can be subjective. What one person considers a problem might not be seen as such by another, depending on their perspective, priorities, and values.

Features of Problems++

- **Objective and Subjective Aspects:** Problems may have both objective elements, such as measurable data or facts, and subjective elements, such as personal opinions or emotional responses.
- **Variability:** Problems can change over time, and their nature may evolve. What was once a problem may no longer be a concern, or new problems may emerge.

Examples of problems in different contexts include

- In personal life: Financial difficulties, relationship issues, health challenges, and time management problems.

Examples of problems in different contexts

- **In professional settings**: Workplace conflicts, project delays, technical glitches, and resource constraints.
- **In academia**: Academic assignments, research questions, and complex mathematical or scientific problems.
- **In technical or engineering domains**: Software bugs, hardware malfunctions, and system optimization challenges.

PROBLEM ANALYSIS

- **Problem analysis** is a thorough and systematic examination of a situation or circumstance with the intention of gaining an in-depth understanding of the primary causes, constraints and solutions
- In the field of computing, problem analysis is a very important ingredient in problem solving through software development.
- Problem analysis encompasses the logical investigation and consideration of a computational problem or software-related issues before trying to find a solution

Problem Analysis Steps

- While attempting to analysis problems, the following are some of the steps that should be followed
- Problem analysis in computing is the process of understanding a problem in order to develop a solution.
- It is a critical step in the software development process, as it helps to ensure that the solution is addressing the actual problem and not just the symptoms.

Problem Analysis Steps +

- **Define the Problem,** Clearly articulate the problem you're facing. This should be a specific and well-defined issue, such as a software error, system performance bottleneck, or a user-related problem.
- **Understand the Context:** Gather background information about the problem's context. Understand the software or system affected, its purpose, and how it fits into the broader IT infrastructure

Problem Analysis Steps ++

- **Define Objectives:** Determine the goals of your problem analysis. What do you aim to achieve through this analysis? What questions need to be answered?
- **Gather Information:** Collect relevant data, facts, and information related to the problem. This may involve examining log files, error messages, user reports, or system documentation.

Problem Analysis Steps +++

- **Replicate the Problem :** If the issue is reproducible, try to replicate it in a controlled environment. This can help isolate the problem and make it easier to analyze.
- **Identify Stakeholders:** Determine who is affected by or involved in the problem. This includes end-users, system administrators, developers, and other relevant parties.

Problem Analysis Steps +++

- **Analyze Causes and Effects:** Investigate the causes of the problem, including software bugs, misconfigurations, hardware failures, or user errors. Consider the effects of the problem on the system's functionality and performance.
- **Identify Patterns and Trends:** Look for patterns, trends, or correlations in the data and information you've gathered. Identifying recurring issues can provide insights into the root cause.

Problem Analysis Steps ++++

- **Review Code and Configuration:** If the problem is related to software, review the codebase or configuration settings to identify potential issues. Pay attention to coding errors, logic flaws, and compatibility issues.
- **Conduct Debugging and Testing:** Use debugging tools and techniques to trace the problem within the code. Test different scenarios to understand how the problem behaves.

Problem Analysis Steps +++++

- **Review System Architecture,** For system-related issues, analyze the architecture to identify bottlenecks, resource limitations, or compatibility issues.
- **Document Findings,** Document your findings in a clear and organized manner. Include detailed information about the problem, its causes, and any relevant data or observations.

Problem Analysis Steps ++++++

- **Prioritize and Categorize Issues:** Organize and prioritize identified issues based on their impact, urgency, and complexity. Categorize issues to facilitate further analysis.
- **Root Cause Analysis :** If possible, perform a root cause analysis to identify the primary source of the problem. This helps prevent recurring issues.

Problem Analysis Steps

- **Recommend Solutions:** Based on your analysis, propose potential solutions or mitigation strategies. Consider short-term and long-term fixes.
- **Estimate Impact and Risks:** Assess the potential impact of each solution and identify any associated risks or drawbacks.
- **Implement Solutions:** Once solutions are approved, implement them carefully, following best practices and change management procedures.

Problem Analysis Steps

- **Monitor and Validate,** After implementing solutions, monitor the system to ensure the problem has been resolved. Validate the effectiveness of your fixes.
- **Communicate Results,** Communicate your findings, solutions, and any preventive measures to relevant stakeholders and team members.
- **Document the Process,** Maintain documentation of the entire problem analysis process, including findings, solutions, and outcomes. This documentation can be valuable for future reference and learning.

Problem Solving Methodologies

- Problem-solving methodologies are systematic approaches and frameworks used to address complex issues, find solutions, and make decisions effectively.
- These methodologies provide a structured way to analyze problems, generate solutions, and implement them.

Problem Solving Methodologies+

- Some common problem-solving methodologies:
- **Scientific Method,** Often used in research and experimental contexts, this method involves
  - Formulating a clear problem statement or hypothesis.
  - Conducting experiments or gathering data.
  - Analyzing the results.
  - Drawing conclusions and making recommendations.

Problem Solving Methodologies++

- **Root Cause Analysis,** Focuses on identifying the underlying causes of a problem rather than just addressing its symptoms. Common techniques include the "Five Whys" and the Ishikawa (Fishbone) diagram.
- **Design Thinking,** A human-centered approach that emphasizes empathy for the end-users. It typically involves stages such as empathize, define, ideate, prototype, and test. Design thinking encourages creative problem solving and innovation.

Problem Solving Methodologies+++

- ✓ **TRIZ (Theory of Inventive Problem Solving),** A systematic approach to solving engineering and inventive problems by identifying inventive principles and patterns used in solving similar problems in other domains.
- ✓ **DMAIC (Define, Measure, Analyze, Improve, Control),** A methodology within Six Sigma for process improvement. It aims to define, measure, analyze, improve, and control processes to eliminate defects or inefficiencies.

Problem Solving Methodologies+++

- **PDCA (Plan, Do, Check, Act),** A continuous improvement methodology that involves:
  - Planning, Identifying a problem and planning a change.
  - Doing, Implementing the change on a small scale.
  - Checking, Assessing the results and gathering data.
  - Acting, Making necessary adjustments and scaling up the change.

Problem Solving Methodologies+++++

- **Pareto Analysis**: Also known as the 80/20 rule, this method helps prioritize issues by identifying the most significant factors contributing to a problem. Pareto Analysis entails Data Collection, Data Classification, Ranking, Analysis, Focus on the Vital Few, Action

- **Mind Mapping**: A visual technique that helps brainstorm ideas and organize thoughts around a central concept or problem. Mind maps can aid in problem exploration and solution generation.

Problem Solving Methodologies++++++

- **Agile Problem Solving**: Agile Problem Solving is an approach to addressing complex challenges and obstacles using principles and practices derived from the Agile methodology. These principles include customer centered approach, Cross-Functional Teams, Continuous Feedback, Prioritization, Adaptive Planning, Small Experiments/test, Prioritization, Retrospectives, Empowerment and Trust, Risk Tolerance
- **Lean Problem Solving,** Rooted in lean manufacturing principles, this approach focuses on eliminating waste and optimizing processes.

Components of Software solution

Once a software related problem are properly understood, then an appropriate solution is developed

The solution development entails components such as development structure chart, pseudo code and flow chart

- **Structure chart**, A structure chart is a hierarchy that shows the functional flow of a program. The structure chart shows the logical breakdown of the program into parts [2]
- Developing a structure inline with the structure chart is a prerequisite for s good software development

Components of Software solution+

- Pseudocode is a high-level, human-readable representation of a computer program or algorithm.
- **Pseudocode** is defined as a step-by-step description of an algorithm. Pseudocode does not use any programming language in its representation instead it uses the simple English language text as it is intended for human understanding rather than machine reading[3]

Components of Software solution++

- It uses plain language and informal syntax to describe the steps and logic of a program without adhering to the strict rules and syntax of a particular programming language.
- **Pseudocode** is often used during the planning and design phase of software development to outline the algorithm's logic before actual coding begins.
- It allows developers to focus on the algorithm's structure and logic without getting bogged down in the details of a specific programming language
- Example: Pseudo code for body mass index(BMI)

    1. Read weight and height
    2. Compute the BMI(bmi=weight/(height*height)
    3. Return the bmi

Flow chart symbols

- A flowchart is a graphical representation of a process or algorithm that uses various shapes, symbols, and arrows to illustrate the sequence of steps and decision points involved.
- Common flow chart symbols
- Start/Stop, Every flow chart has a starting point and a terminating point. The symbol that is used for both the starting and terminating points is a rounded rectangle, a rectangle with round corners. It is called a ‘terminal’

Fig . 1

Flow chart symbols+

- Input/Output: Every time you take an input from a user and return an output to the user, an input/output symbol is used in the flow chart. The symbol that is used for both input/output-related actions is a parallelogram

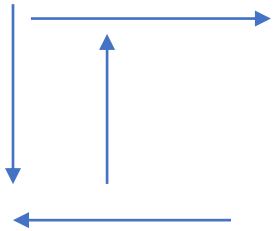
Fig. 2

Flow chart symbols++

- Process: If you are running a processing instruction, you need to use a rectangular box in the flow chart.

Fig. 3

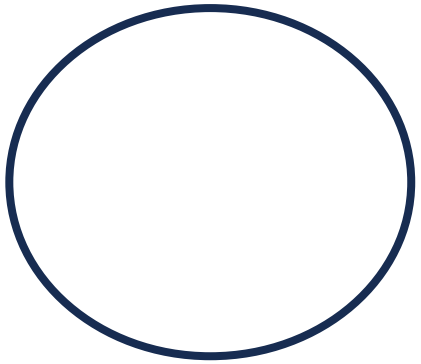
- Flow Lines: Flow lines depict the direction of a flow in a flow chart. There are four types of flow lines



Flow chart symbols+++

- **Connector**: As the name suggests, a connector connects. It connects different steps in a flow chart that are on different pages and gives a sense of continuation. Generally, it is used in extremely complex flow charts and it is denoted by a small circle

- Fig. 4



Flow chart car wipers

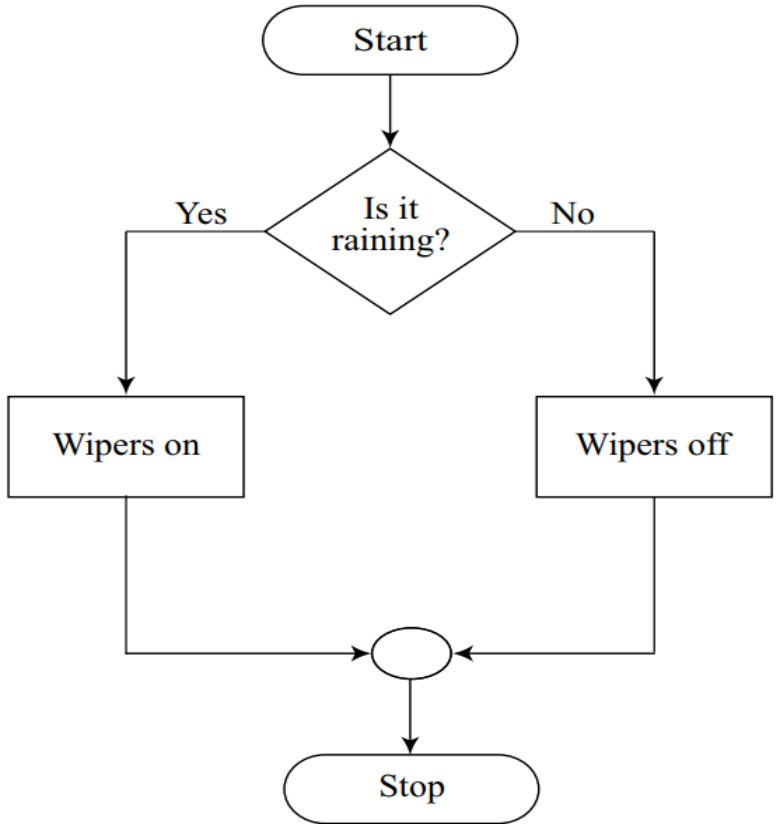


Fig. 5 Selection chart[2]

Guidelines for Drawing Flow Charts

- Drawing flowcharts is an effective way to visually represent processes, workflows, algorithms, or systems. Here are some guidelines for drawing flowcharts:
- Understand the Process: Before you start drawing a flowchart, it's crucial to have a clear understanding of the process or system you want to represent. Gather all necessary information and details.
- Use Standard Symbols: Flowcharts use standard symbols to represent different elements of a process. Common symbols include rectangles for processes, diamonds for decisions, arrows for flow direction, and ovals for start and end points. Adhere to these standard symbols to ensure clarity and consistency.

Guidelines for Drawing Flow Charts+

- Start and End Points: Begin your flowchart with an oval shape to represent the start point and end it with another oval shape to denote the end point. Label these shapes appropriately, often with "Start" and "End."
- Processes: Use rectangles to represent individual processes or actions in the flowchart. Each process should have a clear and concise label describing the action it represents.

Guidelines for Drawing Flow Charts++

- Decisions: Use diamonds to indicate decision points in the process. Label the diamond shape with a question that can be answered with "yes" or "no." The flow should branch accordingly based on the decision outcome.
- Arrows and Lines: Connect the various symbols using arrows or lines to show the flow of the process. Arrows should have clear directionality to indicate the sequence of steps.

Guidelines for Drawing Flow Charts+++

- Connectors: When a flowchart becomes too complex or lengthy, use connectors to join different parts of the flowchart. Connectors are often labeled with letters or numbers that correspond to matching letters or numbers elsewhere in the chart.
- Parallel Processes: If there are parallel processes happening simultaneously, use a dashed line to indicate this parallelism. Ensure that it's clear which processes are happening in parallel.

Guidelines for Drawing Flow Charts++++

- Keep it Neat and Readable: Maintain a consistent and tidy layout for your flowchart. Use spacing and alignment to make it easy to follow. Ensure that labels are legible and not overcrowded.
- Review and Revise: After creating the flowchart, review it for accuracy and clarity. Ensure that it accurately represents the process or system and that it can be easily understood by others.
- Test Your Flowchart: Walk through the flowchart step by step to verify that it correctly represents the process and that all possible scenarios are covered.

Guidelines for Drawing Flow Charts+++++

- Use Flowcharting Software: Consider using specialized flowcharting software or tools to create professional-looking flowcharts. These tools often come with pre-defined symbols and templates.
- Documentation: It's essential to document the flowchart, including a title, date, and any relevant notes or explanations to help others understand the process better.

Guidelines for Drawing Flow Charts++++++

- Feedback: If the flowchart is intended for a team or audience, gather feedback from others to ensure it accurately reflects the process and is easily understandable.
- Remember that the purpose of a flowchart is to simplify complex processes and make them more accessible. Following these guidelines will help you create clear and effective flowcharts for various applications..

Algorithms

- An algorithm is a step-by-step procedure or set of rules for solving a specific problem or accomplishing a particular task.
- Algorithms are essential in computer science, mathematics, and various other fields where problem-solving is required.

Key characteristics of algorithms

- **Well-Defined,** An algorithm must have clear, unambiguous instructions for each step. It should be precise and specific about what actions to take.

Key characteristics of algorithms +

- **Finite**, Algorithms must terminate after a finite number of steps. They should not run indefinitely.
- **Input and Output**: Algorithms typically take some input, process it through a series of steps, and produce an output.
- **Deterministic**, every step of an algorithm should be deterministic. This means that given same inputs it should produce the output each time.

Characteristics of algorithm continues…

- **Efficiency**, Algorithms are often evaluated in terms of their efficiency, which relates to how quickly they solve a problem and how many computational resources they require.
- An efficient algorithm is one that occupy little memory and solves problems with the shortest possible time

Characteristics of algorithm continues…

- **Correctness**: An algorithm is correct if it produces the correct output for all valid inputs. Ensuring the correctness of an algorithm involves rigorous testing and verification.
- **Optimization**: Optimization involves improving the efficiency of an algorithm without changing its fundamental functionality. Optimization can lead to faster execution and reduced resource usage

Characteristics of algorithm continues…

- **Time Complexity**: Time complexity is a measure of how long an algorithm takes to run based on the size of the input. Algorithms with lower time complexity are generally considered more efficient.
- **Space Complexity**: Space complexity measures the amount of memory an algorithm uses relative to the size of the input. Algorithms with lower space complexity use less memory

Types of Algorithms

- **Sorting Algorithms**:
  - **Bubble Sort**: A simple comparison-based sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.
  - **QuickSort**: A divide-and-conquer algorithm that works by selecting a "pivot" element and partitioning the array into smaller sub-arrays.

Types of Algorithms.

- **Merge Sort**: Another divide-and-conquer algorithm that recursively divides the list into smaller sub-lists and then merges them back together.
- **Heap Sort**: A comparison-based sorting algorithm that builds a binary heap to sort elements efficiently.
- **Insertion Sort**: A simple sorting algorithm that builds the final sorted array one item at a time.

Types of Algorithms..

- **Searching Algorithms**:
  - **Binary Search**: A divide-and-conquer search algorithm that efficiently finds the position of a target value within a sorted array.
  - **Linear Search**: A simple search algorithm that checks each element of a list until it finds the desired target.

Types of Algorithms…

- **Graph Algorithms**:
  - **Breadth-First Search (BFS)**: Used for traversing or searching in graph or tree structures level by level.
  - **Depth-First Search (DFS)**: Explores as far as possible along one branch before backtracking in a graph or tree.
  - **Bellman-Ford Algorithm**: Computes the shortest path in a weighted graph, even when it contains negative-weight edges.

Types of Algorithms.....

- **Prim's Algorithm**: Used greedily to find the minimum spanning tree of a connected, undirected graph.
- **Huffman Coding**: A compression algorithm that uses a greedy approach to encode characters more efficiently.

String Matching Algorithms:

Types of Algorithms……

- **Brute-Force String Search**: A basic string matching algorithm that checks for a pattern's occurrence in a text by sliding it character by character.
- **Knuth-Morris-Pratt (KMP) Algorithm**: A more efficient algorithm for substring search based on pattern preprocessing.

Summary

- This week, we delved into the fascinating world of problem-solving, exploring its various facets and intricacies. We commenced by delving into the fundamental concept of problem analysis, examining the diverse features and characteristics that define a problem.
- Our exploration continued as we dissected the crucial steps involved in problem analysis, offering a comprehensive understanding of the processes and methodologies employed to unravel complex issues.

Summary +

- We also dedicated time to explore problem-solving methodologies, shedding light on the systematic approaches used to tackle challenges effectively.
- Furthermore, we delved into the realm of visual problem representation, discussing essential tools such as flowcharts, structure charts, and pseudocode.
- These visual aids play a pivotal role in breaking down complex problems into manageable components, aiding in the development of efficient solutions.

Summary ++

- In addition to these visual aids, we embarked on a deep dive into the concept of algorithms. We examined the various types of algorithms. Algorithms serve as the backbone of problem-solving in the digital realm, providing structured sequences of instructions to efficiently address a wide range of challenges
- As we approach the upcoming week, we are poised to embark on an exciting journey into Python programming.

Summary +++

- Python is a powerful tool widely utilized by computer scientists to harness the capabilities of computers in solving intricate problems. It offers a versatile and user-friendly platform for coding and automation, making it a valuable skill in the world of technology and problem-solving

Reference

- [1] **Maureen sprankle, jim hubbard** (2012)***Problem solving & programming concepts*** *9th edition* pearson education limited
- [2] *Introduction to computing and problem-solving using python*, E Balagurusamy, McGraw Hill Education (India) Private Limited, 2016 page 26
- [3] (2023, May 8). *What is PseudoCode: A Complete Tutorial*. GeeksforGeeks. Retrieved September 29, 2023, from https://www.geeksforgeeks.org/what-is-pseudocode-a-complete-tutorial/
- [4] *Introduction to computing and problem-solving using python*, E Balagurusamy, McGraw Hill Education (India) Private Limited, 2016 page 26