Department of Computer Science and Engineering,

PES University, Bangalore, India

Lecture Notes
Problem Solving With C
UE24CS151B

Lecture #7

Problem Solving using Command Line Arguments

By,
Prof. Sindhu R Pai,
Theory Anchor, Feb-May, 2025
Assistant Professor
Dept. of CSE, PESU

Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Center, PES University)
Prof. Nitin V Pujari (Dean, Internal Quality Assurance Cell, PES University)

Unit #: 3

Unit Name: Text Processing and User Defined Types

Topic: Problem Solving using Command Line Arguments

Course objectives: The objective(s) of this course is to make students

- Acquire knowledge on how to solve relevant and logical problems using computing Machine.
- Map algorithmic solutions to relevant features of C programming language constructs.
- Gain knowledge about C constructs and its associated ecosystem.
- Appreciate and gain knowledge about the issues with C Standards and it's respective behaviours.

Course outcomes: At the end of the course, the student will be able to:

- Understand and Apply algorithmic solutions to counting problems using appropriate C Constructs.
- Understand, Analyze and Apply sorting and Searching techniques.
- Understand, Analyze and Apply text processing and string manipulation methods using Arrays, Pointers and functions.
- Understand user defined type creation and implement the same using C structures, unions and other ways by reading and storing the data in secondary systems which are portable.

Sindhu R Pai

Theory Anchor, Feb - May, 2025

Dept. of CSE,

PES University

Solve the below programs using Command Line Arguments.

Link for Solution:
https://drive.google.com/file/d/1H-QGKao8PKO0aaRzbSLL-7pvU9Iys2Ro/view?usp=drive_link

Level-1: Banana

1. In a large project with several utilities and scripts running on a server, it's essential for the system administrator to know **which program is being executed. For instance, when running a script from the terminal, it's helpful to display the name of the script being executed to ensure everything is working as expected**. This feature allows users to confirm which utility they are interacting with, making debugging and logging more efficient in a large system with multiple utilities.
   Input command: myfile1.exe
   Expected Output: Script name - myfile1.exe

2. A system administrator uses a **file-processing tool that takes a list of file names as input arguments**. The tool **processes these files and generates a report for each one with script names**. This way, the tool verifies which files are being handled in each session, ensuring that everything is as expected.
   Input command: a.exe file1.txt file2.txt file3.txt
   Expected Output:
   ```
   a.exe
   file1.txt
   file2.txt
   file3.txt
   ```

3. Imagine you're building a simple financial tool that allows a user to **input various expenses(in Rs) as command-line arguments and calculates the total expenditure.** This functionality is useful for budgeting or for quickly adding up multiple expenses without requiring a user interface, just by typing the integers in the command line.
   Input command: a.exe 10 20 30
   Expected Output: Total Expenditure is Rs. 60

4. A game development team is building a fun feature for a text-based game **where the player's commands need to be reversed for a special effect**. The program takes the player's actions as arguments, but to trigger a reverse action, it **outputs the commands in the reverse order,** just like how a time reversal power might work in a game
   Input command: a.exe a b c d
   Expected Output: d c b a

5. In a customer support system, agents can input commands to search for tickets. The system is case-insensitive but needs **to display results in uppercase for consistency**. When an agent enters the search query, the program takes the input in any case and outputs the results in uppercase to ensure uniformity and ease of reading.
   Input command: a.exe hEllo world **Expected Output:** HELLO WORLD
   Input command: a.exe 123 peS **Expected Output:** 123 PES

Level-2: Orange

6. A performance tracking system requires **identifying the highest score among several tests.** The system processes test scores given as command-line arguments, and the user can input the scores directly. The tool helps the user quickly **determine the highest score from a batch of integers, essential for performance reports.**
   Input command: a.exe 12 56 23 89
   Expected Output: 89

7. A file management system tool is being built to sort a list of files by their names or modification dates. The tool takes **file names as command-line arguments and sorts them alphabetically.** This feature is especially useful for organizing files before backing them up or displaying them in a user interface. Developers or system administrators can use it to quickly verify file ordering from the terminal.
   Input command: a.exe mango.txt apple.txt banana.txt
   Expected Output: apple.txt banana.txt mango.txt

8. A startup is developing **a text summarization tool that analyzes document metadata like titles or keywords.** To prioritize display, the tool **first sorts the input strings based on their lengths** — shorter terms often carry more weight in summarization and UI previews. The developers must be **building a quick command-line utility to test this functionality by entering words or phrases and seeing them sorted by their length**.
   Sample execution #1:
   Input command: a.exe apple fig mango chikkkku
   Expected Output:
   fig (Length: 3)
   apple (Length: 5)
   mango (Length: 5)
   chikkkku (Length: 8)
   Sample execution #2:
   Input command: a.exe
   Expected Output: No arguments are sent in the command line

Level-3: Jackfruit

9. In a computer science class, students are tasked with developing a utility that checks whether the **integers passed through the command line are prime numbers**. A prime number is a number greater than 1 that has no divisors other than 1 and itself. The professor explains this by taking a list of integers from the command line when the program is run. This program helps students learn how to process command-line arguments dynamically and efficiently determine prime numbers using a simple algorithm.
   Sample Executions:
   Input command: a.exe
   Expected Output: Please provide integers in the command-line to check prime or not
   Input command: a.exe 10 17 23 30
   Expected Output: 17 23

10. In a class, students are asked to build a tool that **shows the Fibonacci sequence for a specified number of terms.** The Fibonacci sequence is a series of numbers where each term (starting from the third term) is the sum of the two preceding ones. The first two terms are typically defined as 0 and 1. Take the **number of terms from the command line.**
    Sample Execution #1:
    Input command: fibonacci.exe
    Expected Output: Usage: fibonacci.exe <number_of_terms>
    Sample Execution #2:
    Input command: fibonacci.exe -3
    Expected Output: Please enter positive number.
    Sample Execution #3:
    Input command: fibonacci.exe 6
    Expected Output:
    Fibonacci Sequence up to 6 terms:
    0, 1, 1, 2, 3, 5,

Keep Exploring Command Line Arguments!!!