ÇANKAYA UNIVERSITY

SOFTWARE ENGINEERING DEPARTMENT

SOFTWARE PROJECT I

REPORT SUBTOPICS

Name Surname	**Duru Karacan**
Identity Number	**202128022**
Course	**SENG 271**
Experiment	**Experiment 1**
E-mail	**c2128022@student.cankaya.edu.tr**

REQUIREMENT ANALYSIS

What are the expectations from the program?

The program is expected to simulate a battle game by utilizing a stack data structure to manage soldiers on two sides.

The simulation should accurately reflect the outcome of battles based on the provided commands, considering the soldiers' attributes like health and strength.

The expectations include correctly processing the commands, generating meaningful and accurate outputs, and providing relevant error messages in case of issues with the input file.

How does it work?

The program functions by reading a sequence of commands from the input.txt file. These commands are divided into four types: Add Soldier, Fight, Call Reinforcement, and Critical Shot.

For each command, the program performs specific actions, such as adding soldiers to the stack, simulating battles, introducing reinforcements with random attributes, or executing critical shots.

The battles' outcomes are calculated using a predefined formula for damage, and the soldiers are managed within two separate stacks, each representing one side.

What are the functional requirements?

The functional requirements encompass several key aspects, including:

- Parsing and interpreting commands from the input file correctly.

- Accurately calculating and applying damage during battles between soldiers.

- Adding soldiers to the respective stack with the specified attributes (health and strength).

- Calling reinforcements to either side with randomly generated attributes.

- Handling critical shots that eliminate soldiers from the opposing stack.

- Generating outputs that match the expected results for each command type.

- Providing informative error messages to guide users in the event of invalid commands or input format issues.

Software Usage:

The software is designed to be used by creating an **input.txt** file with a sequence of commands following the specified format. Users should run the program, providing this input file to simulate the battle game and receive the expected outputs.

Error Messages:

The program is expected to display relevant error messages when it encounters invalid commands or incorrect input formats in the input file. These error messages should guide users in identifying and correcting issues with their input.

Requirements:

The program should adhere to the outlined requirements and produce the expected outputs for each type of command, ensuring that the simulated battle game functions as intended.

DESIGN

How will the program perform the desired job and produce the output ?

- Problem: The program's primary challenge is to simulate a battle game by effectively managing soldiers, calculating damage, and responding to specific commands.

- Solution: The program utilizes a stack data structure to represent soldiers on both sides, facilitating efficient management. Soldiers are characterized by their health and strength attributes, which are stored in the stack. The program calculates damage during battles using a predefined mathematical formula. It applies the appropriate logic to each command type, such as adding soldiers to the stack, calling reinforcements with random attributes, or executing critical shots.

- Main Data Variable: The main data variables are the soldiers' health and strength, which are stored as integers within the stack data structure.

- Algorithm: The program follows a clear algorithm that involves reading commands from the input file, parsing these commands, and executing corresponding actions. The algorithm accurately calculates damage based on the soldiers' strengths and manages the soldiers within the stack based on the command type.

- Special Design Properties: A key design property is the utilization of the stack data structure to represent soldiers, allowing for dynamic sizing, efficient insertions and deletions, and reduced memory wastage. Command-specific functions ensure that the simulated battles follow the rules defined for each action.

- Execution Flow Between Subprograms: The main program reads commands from the input file and branches into subprograms based on the command type. These subprograms are responsible for executing the corresponding actions, such as adding soldiers, simulating battles, calling reinforcements, or executing critical shots.

TESTING

How can the program break, produce wrong output, work incorrectly?

- Bugs and Software Reliability: The program may experience issues if there are bugs in the code. These bugs can manifest as incorrect command parsing, flawed damage calculations, or problems related to the stack, such as adding soldiers to a full stack or popping from an empty stack. Ensuring the software's reliability and addressing these issues is crucial to its proper functioning.

- Software Extendibility and Upgradeability: Over time, the program may require enhancements and extensions to support more complex war strategies or additional features. It should be designed in a way that allows for straightforward modifications and upgrades, ensuring its adaptability to evolving requirements.

- Performance Considerations: To guarantee the program's reliability under various scenarios, it should be tested for performance considerations. This includes assessing its ability to handle larger inputs, more extensive battles, or more complex simulations efficiently without a significant drop in performance.

COMMENTS

This report provides a comprehensive overview of the programming assignment, outlining the goals, design decisions, and testing strategies.

The expectations and functional requirements are clearly defined, with a robust analysis of potential issues and solutions.

The program's design, including the use of a stack data structure and specific algorithms, is well-documented.

The testing section covers various scenarios, highlighting the program's reliability and considerations for future enhancements.

Overall Result:

The programming assignment has yielded a functional and well-designed C program that successfully simulates a battle game. The use of a stack data structure, clear algorithms, and adherence to functional requirements contribute to a robust solution.

Success of the Project:

The programming project can be considered a success. It meets the specified objectives, simulating battles effectively and producing expected outputs. The design choices, such as using a stack for soldier management, align with the project goals.