Project 3 - Witcher Tracker
CmpE 230, Systems Programming, Spring 2025

Instructor: Can Özturan
TAs: Gökçe Uludoğan
SA: Abdullah Rüzgar

Due: June 1, 2025, 23:59 (Strict)

We assign the first project to be written in C++ using the object-oriented programming (OOP) paradigm.

1 Introduction

The Witcher is a popular fantasy series created by Polish writer Andrzej Sapkowski. The series has been adapted into several formats, including video games that gained immense popularity. Set in a world full of magic, monsters, and political intrigue, the story follows Geralt of Rivia, beloved by fans, a Witcher — a genetically enhanced human trained to hunt and kill dangerous monsters. Witchers are often seen as outcasts, feared by many for their abilities and mysterious origins.

In this project, you will implement an interpreter and an inventory-event tracking system for Geralt in C programming language. You can work individually or as a group of two.

The inventory will store alchemical ingredients, potions and monster trophies. There will be a bestiary containing information about effective potions and signs against monsters. An alchemy section will track formulae for brewing potions.

2 Details

Your program will process three types of input:

- **Sentence**: Defines inventory actions, new knowledge, or monster encounters.
- **Question**: Retrieves information about Geralt's inventory, characters in the world, bestiary, or alchemy.
- **Exit Command**: Terminates the program.

Grammar rules are provided in section 4. Grammar for the System.

The corresponding responses are provided with example inputs. Inputs are indicated with » and outputs are indicated with « in the examples (Do not include "«" in the output yourselves!!!). Every input that does not follow the grammar rules must be responded with `INVALID`.

2.1 Initial State

Geralt is starting from scratch after he lost his memory. He has no items, potions in his inventory. With his memories he has forgotten the formulae of potions and how to fight against monsters. He needs to gather ingredients and learn to survive in this world from the beginning.

2.2 Entities

- **Alchemy**:
  - **Ingredients**: Alchemical ingredients. Used to make potions.
  - **Potions**: Used during combat. Certain potions are effective against certain monsters.
- **Magic**:
  - **Signs**: Witcher signs. They are infinitely sourced. They are effective against certain monsters just like potions.
- **Bestiary**:
  - **Monsters**: Geralt can defeat them with the aid of signs or potions, but prior knowledge is required to counter them. This knowledge is stored in the bestiary.
  - **Trophies**: Geralt gains a trophy for each defeated monster. These can be traded for ingredients.

2.3 Actions

- **Loot**: Geralt loots ingredients. The loots can contain one or more kinds of ingredients. Every kind of ingredient has positive quantity in the loot.

  Example inputs:

```
>> Geralt loots 5 Rebis
>> Geralt loots 3 Rebis, 4 Vitriol, 2 Aether
```

Output:

- Loot action always has the same response:

```
<< Alchemy ingredients obtained
```

- **Trade**: Geralt trades trophies for ingredients. Trade will not happen if any of the trophies are insufficient or non-existent in the inventory. In that case nothing should change in his inventory.

  Example inputs:

```
>> Geralt trades 2 Wyvern trophy for 8 Vitriol, 3 Rebis
>> Geralt trades 1 Basilisk, 3 Drowner, 1 Striga trophy for 3 Albedo
```

  Outputs:

  - If Geralt is missing any of the said trophies in his inventory:

```
<< Not enough trophies
```

  - After successful trades:

```
<< Trade successful
```

- **Brew**: Geralt brews a potion using ingredients at hand. To brew a potion Geralt needs prior knowledge about the formula of the potion. After brew actions his inventory should contain one more of that potion.

  Example inputs:

```
>> Geralt brews Black Blood
>> Geralt brews Swallow
```

  Outputs:

  - If he does not know the potion formula:

```
<< No formula for Black Blood
```

  - If he does not have enough ingredients:

```
<< Not enough ingredients
```

  - If the alchemy action is successful:

```
<< Alchemy item created: Black Blood
```

- **Learn**: Geralt learns effective signs or potions against enemies. There can be multiple effective counters to a monster though one action can only contain one of them. He can also learn potion recipes.

  Example inputs for effectiveness:

```
>> Geralt learns Igni sign is effective against Harpy
>> Geralt learns Swallow potion is effective against Drowner
```

  Outputs:

  - If it is the first time the beast is mentioned:

```
<< New bestiary entry added: Harpy
```

  - If there was already another effective potions or signs known:

```
<< Bestiary entry updated: Harpy
```

  - If the effective potion or sign is already in bestiary:

```
<< Already known effectiveness
```

  Example inputs for formulae:

```
>> Geralt learns Black Blood potion consists of 3 Vitriol, 2 Rebis, 1 Quebrith
>> Geralt learns Willow potion consists of 2 Aether, 1 Quebrith
```

  Outputs:

  - If the potion formula was already in alchemy base:

```
<< Already known formula
```

  - If the potion formula is new to the alchemy base:

```
<< New alchemy formula obtained: Black Blood
```

- **Encounter**: Geralt encounters a monster. The outcome depends on the Witcher's knowledge and potions in his inventory. If he has at least one effective potion or sign the outcome of the encounter is successful. After successful encounters, Geralt takes a part of the monster as a trophy. These trophies are valuable items that can be traded in for alchemy ingredients. You must add a trophy to Geralt's inventory after each successful encounter.

  Geralt will consume one potion of every kind that is known to be effective against the monster he encounters and is available in the inventory. He only needs one effective potion in the inventory to win the fight but he always consumes one of every effective potion in the inventory just in case.

  Example inputs:

  ```
  >> Geralt encounters a Bruxa
  >> Geralt encounters a Golem
  ```

  – If Geralt has at least one effective potion or sign:

  ```
  << Geralt defeats Bruxa
  ```

  – If Geralt does not have effective potions or knowledge of effective signs, he will not be able to defeat the monster:

  ```
  << Geralt is unprepared and barely escapes with his life
  ```

3 Questions

Your program should answer various queries about Geralt's inventory, bestiary, alchemy

3.1 Inventory Queries

- **Specific Ingredient**: Retrieve the total number of a specific ingredient.

  ```
  >> Total ingredient Vitriol ?
  << 10
  ```

  – If the ingredient is not in the inventory

  ```
  >> Total ingredient Albedo ?
  << 0
  ```

- **Specific Potion**: Retrieve the total number of a specific potion.

```
>> Total potion Black Blood ?
<< 3
```

  - If the potion is not in the inventory:

```
>> Total potion Cat ?
<< 0
```

- **Specific Trophy**: Retrieve the total number of a specific monster trophy.

```
>> Total trophy Basilisk ?
<< 2
```

  - If the trophy is not in the inventory:

```
>> Total trophy Kikimore ?
<<  0
```

- **All Ingredients**: Retrieve the total number of all ingredients in the inventory. Sorted by ingredient names.

```
>> Total ingredient ?
<< 7 Aether, 5 Rebis, 10 Vitriol
```

  - If there are no ingredients in the inventory:

```
>> Total ingredient ?
<< None
```

- **All Potions**: Retrieve the total number of all potions in the inventory. Sorted by potion names.

```
>> Total potion ?
<< 2 Black Blood,  2 Cat, 1 Maribor Forest, 1 Thunderbolt
```

  - If there are no potions in the inventory:

```
>> Total potion ?
<< None
```

- **All Trophies**: Retrieve the total number of all trophies in the inventory. Sorted by monster names.

```
>> Total trophy ?
<< 1 Basilisk, 2 Bruxa, 1 Leshen
```

  - If there are no trophies in the inventory:

```
>> Total trophy ?
<< None
```

3.2 Bestiary and Alchemy Queries

- **Potion/Sign Effectiveness**: Retrieve all known potions and signs effective against a monster. Sorted by potion/sign names.

```
>> What is effective against Bruxa ?
<< Black Blood, Yrden
```

  - If Geralt has no knowledge of effective potions or signs against the monster:

```
>> What is effective against Wraith ?
<< No knowledge of Wraith
```

- **Potion Formula**: Retrieve ingredients needed for a potion. Sorted by ingredient quantity. If there are multiple ingredients with the same quantity sort them by ingredient names.

```
>> What is in Black Blood ?
<< 3 Vitriol, 2 Rebis, 1 Quebrith
```

  - If Geralt does not have the formula for the potion:

```
>> What is in Full Moon ?
<< No formula for Full Moon
```

4 Grammar for the System

The following grammar defines the structure of valid sentences for the program. It follows a formal notation using Backus-Naur Form (BNF) with additional annotations to clarify constraints.

Every input must follow these grammar rules. Inputs that do not follow these grammar rules are `INVALID`.

4.1 Entities

This subsection outlines the naming conventions and grammar rules for the main entities recognized by the system.

4.1.1 Ingredients

The names of the ingredients consist of **exactly one** word and only **alphabetical** characters. Ingredients **must** come after a **positive** integer representing the quantity of the ingredient. Multiple quantity-ingredient pairs are separated by a **comma** ",". There must be **at least one** empty space between quantities and ingredients. There can be zero or more space between commas and quantities or ingredients.

```
<quantity> ::= <integer> // A positive integer
<ingredient> ::= "Rebis" | "Vitriol" | "Aether" | "Quebrith" | "Vermilion" | ...
<ingredient_list> ::= <quantity> <ingredient>
             | <quantity> <ingredient> "," <ingredient_list>
```

4.1.2 Monsters

The names of the monsters consist of **exactly one** word and only **alphabetical** characters.

```
<monster> ::= "Wyvern" | "Bruxa" | "Griffin" | "Harpies" | "Leshen" | ...
```

4.1.3 Trophies

Trophy lists contain **at least** one quantity-monster pair followed by a "trophy". Each pair contains **exactly one positive** integer as quantity and monster. Each pair is separated by a comma (,) in the list.

```
<trophy_list> ::= <quantity> <monster> "trophy" |
                <quantity> <monster> "," <trophy_list>
```

4.1.4 Potions

The names of the potions consist of **at least one** word and only **alphabetical** characters. Each word in a potion name is separated by **exactly one** empty space (there can be multiple empty spaces before or after the potion name but there can only be a single space between words of the potion name).

```
<potion> ::= "Black Blood" | "Swallow" | "White Raffards Decoction" | ...
```

4.1.5 Signs

The names of the signs consist of **exactly one** word and only **alphabetical** characters.

```
<sign> ::= "Igni" | "Axii" | "Aard" | "Yrden" | "Quen" | ...
```

4.2 Overall Structure

This section outlines the general structure of commands in the system. Every command is defined as either a sentence, a question, or an exit command.

In the inputs, there can be any number of spaces between the tokens unless otherwise specified. And each token is case-sensitive.

```
<command> ::= <sentence> | <question> | <exit_command>
```

4.3 Sentence Structure

This section describes how a sentence is formed. A sentence can be an action sentence, a knowledge sentence, or an encounter sentence.

```
<sentence> ::= <action_sentence> | <knowledge_sentence> | <encounter_sentence>
```

4.4 Action Sentences

```
<action_sentence> ::= <loot_action> | <trade_action> | <brew_action>
```

4.4.1 Loot Action

```
<loot_action> ::= "Geralt loots" <ingredient_list>
```

4.4.2 Trade Action

```
<trade_action> ::= "Geralt trades" <trophy_list> "for" <ingredient_list>
```

4.4.3 Brew Action

```
<brew_action> ::= "Geralt brews" <potion>
```

4.5 Knowledge Sentences

```
<knowledge_sentence> ::= <effectiveness>
                       | <potion_formula>


<effectiveness> ::= "Geralt learns" <sign> "sign is effective against" <monster>
    | "Geralt learns" <potion> "potion is effective against" <monster>

<potion_formula> ::= "Geralt learns" <potion> "potion consists of" <ingredient_list>
```

4.6 Encounter Sentence

```
<encounter_sentence> ::= "Geralt encounters a" <monster>
```

4.7 Question Structure

There must be a question mark (?) at the end of each question. There can be zero or more space characters between the question mark and the last word of the question. There can be zero or more spaces after the question mark.

```
<question> ::= <inventory_query>
             | <bestiary_query>
             | <alchemy_query>
```

4.7.1 Inventory Queries

```
<inventory_query> ::= <total_specific_query>
                    | <total_all_query>

total_specific_query> ::= "Total" <category> <specific_item> "?"
<specific_item> ::= <ingredient> | <potion> | <monster>

<total_all_query> ::= "Total" <category> "?"
<category> ::= "ingredient" | "potion" | "trophy"
```

4.7.2 Bestiary Queries

```
<bestiary_query> ::= "What is effective against" <monster> "?"
```

4.7.3 Alchemy Queries

```
<alchemy_query> ::= "What is in" <potion> "?"
```

4.8 Exit Command

```
<exit_command> ::= "Exit"
```

5 Input & Output

Your program should:

- Read input with » prompt.
- Handle up to 1024-character lines.
- Print the corresponding response for valid sentences and `INVALID` for errors.
- Test inputs will cover invalid cases for syntax errors mostly. There won't be cases for semantic inconsistencies like "Geralt loots Cat" where Cat is known to be a potion.
- Terminate the program with the exit command "Exit".

5.1 Demonstration

Below is a sample session demonstrating various actions and queries along with their expected outputs.

```
>> Geralt loots 5 Rebis
Alchemy ingredients obtained
>> Geralt loots 4 Vitriol, 1 Quebrith
Alchemy ingredients obtained
>> Geralt learns Black Blood potion consists of 3 Vitriol, 2 Rebis, 1 Quebrith
New alchemy formula obtained: Black Blood
>> Geralt brews Black Blood
Alchemy item created: Black Blood
>> Geralt learns Igni sign is effective against Harpy
New bestiary entry added: Harpy
>> Geralt encounters a Harpy
Geralt defeats Harpy
>> Total ingredient ?
3 Rebis, 1 Vitriol
>> Total potion Black Blood ?
1
```

```
>> Total trophy Harpy ?
1
>> Geralt trades 1 Harpy trophy for 8 Vitriol, 3 Rebis
Trade successful
>> Total ingredient ?
6 Rebis, 9 Vitriol
>> What is in Black Blood ?
3 Vitriol, 2 Rebis, 1 Quebrith
>> What is effective against Harpy?
Igni
>> Geralt brews Swallow
No formula for Swallow
>> Geralt trades 1 Wyvern trophy for 8 Vitriol, 3 Rebis
Not enough trophies
>> Geralt learns Swallow potion consists of 10 Vitriol, 4 Rebis
New alchemy formula obtained: Swallow
>> Geralt brews Swallow
Not enough ingredients
>> Geralt loots 2 Vitriol, 2 Rebis
Alchemy ingredients obtained
>> Geralt brews Swallow
Alchemy item created: Swallow
>> Total potion Swallow ?
1
>> Exit
```

5.2 Invalid Input Examples

```
>> Geralt loots Rebis (missing quantity)
INVALID
>> Geralt loots -1 Vitriol (negative quantity)
INVALID
>> Geralt loots 0 Rebis (zero quantity)
INVALID
>> Geralt trades 2 Wyvern trophy in exchange 5 Rebis (wrong keyword)
INVALID
>> Geralt brew Black Blood (missing "s")
INVALID
>> Geralt learns is effective against Golem (missing potion/sign name)
INVALID
>> Geralt trades Basilisk trophy for 3 Rebis (missing quantity before Basilisk)
INVALID
>> Geralt encounters a B4rghest (numeric character in the monster name)
```

```
INVALID
>> Total ingredient (missing question mark)
INVALID
>> Geralt brews Black  Blood (multiple spaces inside the potion name)
INVALID
>> Geralt loots 2Vitriol (missing spaces)
INVALID
>> Geralt trades 1 Basilisk, Striga trophy for 5 Rebis (missing quantity)
INVALID
>> Total Black Blood? (missing category name)
INVALID
>> Geralt encounters Drowned_Dead (non-alphabetic character)
INVALID
>> Gerald loots 5 Rebis (misspelled Geralt)
INVALID
>> Geralt encounters Harpy (missing "a")
INVALID
>> Exit
```

6 Submission

Your project will be tested automatically on the provided docker image. Thus, it's important that you carefully follow the submission instructions. Instructions for setting up the development environment can be found at this page. The root folder for the project should be named according to your student id numbers. If you submit individually, name your root folder with your student id. If you submit as a group of two, name your root folder with both student ids separated by an underscore. You will compress this root folder and submit it with the `.zip` file format. Other archive formats will not be accepted. The final submission file should be in the form of either `2020400144.zip` or `2020400144_2020400099.zip`.

You must create a Makefile in your root folder which creates a `witchertracker` executable in the root folder, do not include this executable in your submission, it will be generated with the `make` command using the Makefile. The `make` command should not run the executable, it should only compile your program and create the executable.

Additionally, commit your progress and submit your project to the Github Classroom assignment to verify its structure and basic functionality. You can join the assignment using this invite link. Please commit your progress and submit your project to this assignment to verify its structure and basic functionality. For team formation, either create a team with your partner by combining your student IDs (e.g., 2022400XXX-2022400YY) or join the team that your partner has already set up.

Submission to GitHub allows you to automatically test your code within the provided

Docker image environment. Each repository is allocated approximately **10 minutes** of GitHub Actions runtime. Although each individual workflow run is limited to a maximum of **10 minutes**, triggering the workflow more than **3 times** may consume your entire allowance.

Our organization has a total limit of **3000 GitHub Actions minutes**.
Please be cautious to avoid depleting your repository's minutes and preventing both yourself and others from testing their submissions.
To conserve minutes, you may manually trigger the grading workflow by navigating to the repository's `Actions` tab and selecting `Workflows → grade`.

Late Submission

If the project is submitted late, the following penalties will be applied:

Hours late	Penalty
0 < hours late <= 24	25%
24 < hours late <= 48	50%
hours late > 48	100%

7 Grading

Your project will be graded according to the following criteria:

- **Code Comments (10%):** Write code comments for discrete code behavior and method comments. This sub-grading evaluates the quality and quantity of comments included in the code. Clear, purposeful comments are essential for understanding not just what the code does, but why and how. Key expectations for comments:
  - **Function/Class-Level Comments:** At the top of each class, method, or function, include a short description of its role, parameters, return value, and side effects. Use proper Doxygen-style formatting if possible (e.g., ///, /** ... **/).
  - **Inline comments:** Add comments next to or above complex expressions, important control flow decisions, or operations with non-obvious purpose. Focus on clarifying why something is done, rather than repeating what is written in code.
  - **Code structure clarity:** Use comments to mark logical sections of the code to improve readability.

  Proper documentation ensures that the code is easily readable and maintainable for future developers. Refer to the associated repository for an brief example demonstrating how to use Doxygen and automate documentation.

- **Documentation (15%):** A written document describing how you implemented your project. This sub-grading assesses the quality and completeness of the written documentation accompanying the project. Good documentation should effectively communicate the project's purpose, design, and implementation details, ensuring that others can understand and build upon your work. Key expectations for documentation:
  - **Project Overview:** Clearly state the problem definition, objectives, and motivation behind the project.
  - **Design and Implementation:** Write structured paragraphs (not just bullet points) to describe:
    - **Class and Object Design:** Identify key classes and their responsibilities. Explain how they interact.
    - **Data Structures:** Discuss your use of arrays, STL containers, or custom structs/classes.
    - **Execution Flow:** Describe how the program processes input to produce output. Flowcharts or diagrams are encouraged, but must be explained in the text.
    - **Key Snippets (if used):** Include only when necessary and provide clear context.
  - **Challenges and Solutions:** Explain technical difficulties you encountered and how you overcame them.
  - **Usage Guide:** Provide compilation and execution instructions. Show example inputs and expected outputs. Screenshots are welcome but should be explained.
  - **Code Structure Summary:** Outline the organization of your project. Briefly describe the purpose of main files or modules and how they contribute to the overall functionality.

  Students should aim for concise, well-organized, and readable documentation. For formatting, follow the provided project report template in LaTeX, which demonstrates the expected structure.

  Reminder

  Visual elements (e.g., UML diagrams, code snippets) should support your narrative, not replace it.

- **Implementation and Tests (75%):** The submitted project should be implemented following the guidelines in this description and should pass testing. This sub-grading assesses the quality and correctness of the implementation.

  Grading the test cases will be straightforward: your score from the test cases will be determined by the formula:

  Total Correct Answers to Lines / Total Lines Given

8 Warnings

- You can submit this project individually or as a group of two.
- All source codes are checked automatically for similarity with other submissions and exercises from previous years. Make sure you write and submit your own code. Any sign of cheating will be penalized with an F grade.
- Project documentation should be structured in a proper manner. The documentation must be submitted as a .pdf file, as well as in raw text format.
- Make sure you document your code with necessary inline comments and use meaningful variable names. Do not over-comment, or make your variable names unnecessarily long. This is very important for partial grading.
- Do not start coding right away. Think about the structure of your program and the possible complications resulting from it before coding.
- Questions about the project should be sent through the discussion forum on Piazza.
- There will be a time limit of 30 seconds for your program execution. Program execution consists of the total amount of time for all queries. This is not a strict time limit and the execution times may vary for each run. Thus, objections regarding time limits will be considered if necessary.

9 AI Assistants

If AI assistants were used during this project, the report must explicitly describe their usage. Clearly specify the extent of AI involvement to ensure transparency and maintain academic integrity.

The use of AI assistants (such as ChatGPT, Copilot, or similar tools) is permitted for limited purposes, including:

- Grammar and spelling correction in documentation.
- Debugging assistance (e.g., understanding error messages, suggesting fixes).
- Clarifications on C programming concepts and best practices.

However, students must adhere to the following restrictions:

- **Do not use AI-generated content directly in your code or documentation.** Doing so will be viewed as plagiarism and thoroughly investigated.
- Ensure all submitted work is your own. AI tools should be used as an aid, not as a replacement for understanding or implementation.
- Clearly document any AI usage in your report.

Failure to comply with these guidelines may result in academic penalties.