Program Design, testing, and maintenance

through text based adventure game development

Made for CIE AS level at Musashi International School Tokyo

Goals of this course

- **Describe** and **construct** structure charts.
- **Derive** pseudocode from structure charts.
- **Understand** and **apply** testing strategies:
  - **normal**
  - **boundary/extreme**
  - **abnormal**
- **Understand** maintenance needs:
  - **perfective**
  - **adaptive**
  - **corrective**

How will we achieve this?

- By **designing** and **developing** a text based adventure game in Python.
- By **documenting** the design process using structure charts and pseudocode.
- By **testing** the program regularly using testing strategies.
- By discussing eventual **maintenance** activities.

Game specifications

- Text based adventure game.
- Player can move between different locations.
- Player can collect items.
- Player can use items to solve puzzles.
- Game has a clear objective to achieve.

Step 1: Improving specifications

Let's agree on more detailed specifications for our game.

These are high level considerations. They should describe user experience, not technical details.

Some questions to consider:

- What is the objective of the game?
- How many locations will there be?
- What kind of items can the player collect?
- What kind of puzzles will the player need to solve?
- How will the player interact with the game (commands)?
- What is the general flow of the game?

Fill in the specifications on your worksheet.

Step 2: Structure charts

Based on the specifications, we can start designing the structure of our program using structure charts.

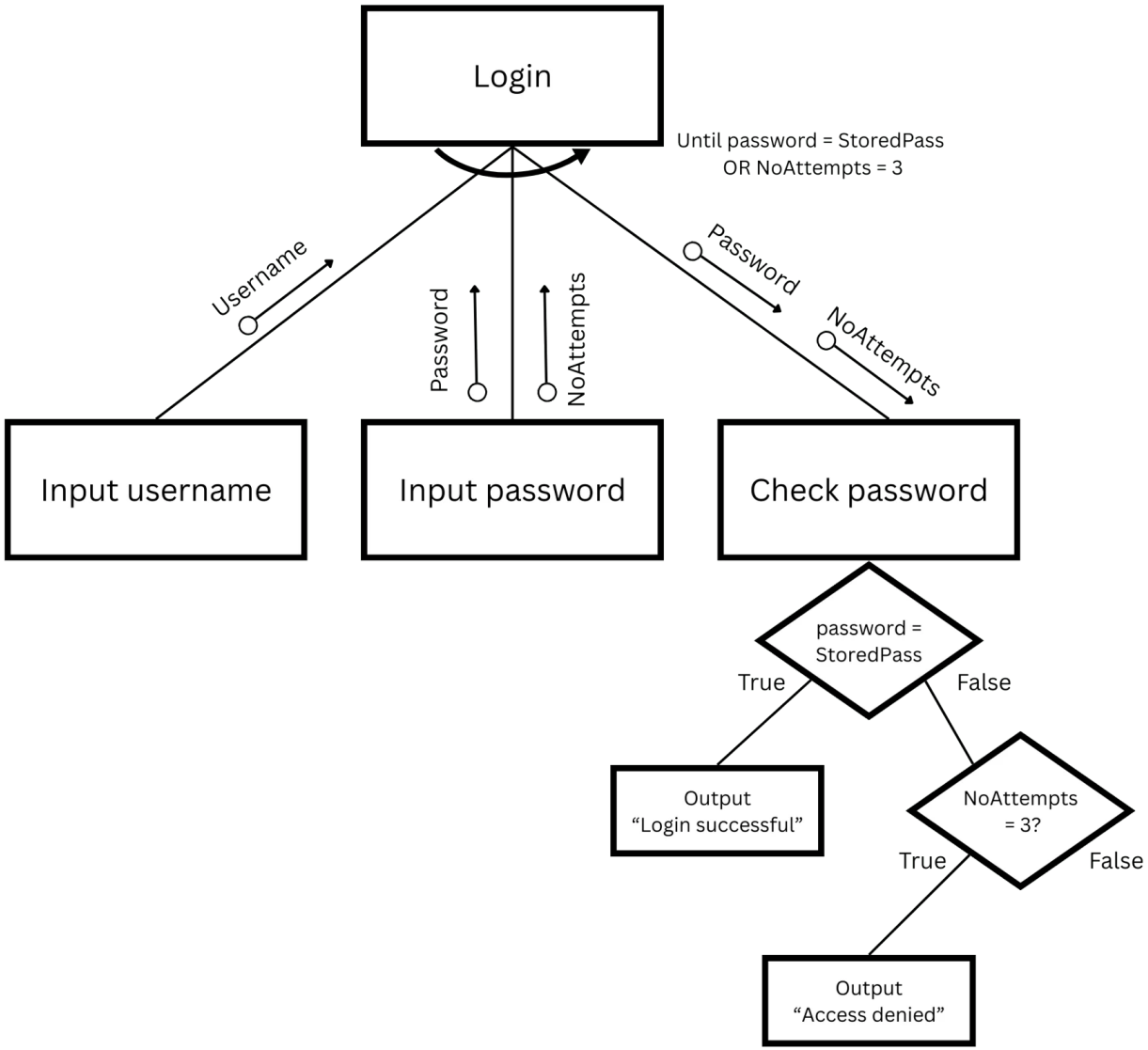
A structure chart is a hierarchical diagram that shows the different modules of a program and how they interact with each other.

Some questions to consider:

- What are the main modules of the program?
- How do these modules interact with each other?
- What are the inputs and outputs of each module?

Let's take a look at an example structure chart on the next slide.

Example structure chart



Now, it's your turn to create a structure chart for your game based on the specifications you developed earlier.

Make it using the software of your choice, or draw it on paper and take a picture, and add it to your worksheet.

Remember to consider the main modules of your program and how they interact with each other.

Step 3: Pseudocode

Once you have your structure chart, you can start writing pseudocode for each module.

Remember to use Cambridge's pseudocode^{TM} conventions:

- Use `MODULE` and `ENDMODULE` to define modules.
- Use `PROCEDURE` and `ENDPROCEDURE` for procedures.
- Use `FUNCTION` and `ENDFUNCTION` for functions.
- Use `IF`, `ELSEIF`, `ELSE`, and `ENDIF` for conditional statements.
- Use `FOR`, `NEXT`, `WHILE`, and `ENDWHILE` for loops.
- Use `CALL` to call other modules.

Let's look at an example pseudocode on the next slide.

Example pseudocode

```
MODULE Login
  DECLARE Username : STRING
  DECLARE Password : STRING
  DECLARE StoredPass : STRING
  DECLARE Attempts : INTEGER
  Attempts <- 0
  Username <- Input_Username
  WHILE NOT password = StoredPass and Attempts < 3
    CALL Input_Password(Password, Attempts)
    CALL Check_Password(Password, Attempts)
  ENDWHILE
ENDMODULE

\\ Other modules...
```

Your turn: Pseudocode

Now, it's your turn to write pseudocode for a few modules of your program based on the structure chart you created earlier.

Write pseudocode for at least two modules, considering their inputs, outputs, and internal logic.

Check with your teacher for feedback on your pseudocode.

Once you have completed your pseudocode, copy it to your worksheet.

Implementation and testing

Let's start implementation. For this project, I want you to focus on writing clean, well-documented code that follows best practices.

This includes:

- Using meaningful variable and function names.
- Writing comments to explain your code.
- Writing docstrings for your functions and modules.
- Split responsibilities clearly between different functions and modules.

Complete flow

The development process should follow this flow. After each step, call me to check on your progress before moving on:

- ☐ Clarify specification for the next thing to implement.
- ☐ Draw a structure chart for it if it's bigger than a simple function.
- ☐ Write pseudocode for the function/module.
- ☐ Implement the function/module.
- ☐ Test the function/module thoroughly (see next slide).
- ☐ When you get the all clear from me, move on to the next piece.

To do list

Everytime you complete a function or module, check this to do list before moving on:

- ☐ Does this piece of code has a clear purpose?
- ☐ Does it do things that should be something else's responsibility?
- ☐ Is it well documented with comments and docstrings?
- ☐ Have I tested it using normal, boundary, and abnormal test cases?
  - ☐ Normal: typical inputs that the program is expected to handle.
  - ☐ Boundary: inputs at the edge of acceptable ranges.
  - ☐ Abnormal: unexpected or invalid inputs to test error handling.