SIT102 Introduction to Programming

Pass Task 7.1: Arrays and Structs

Overview

In this task you will continue to work on the knight database to help Camelot keep track of all of their knights. We can now add a **kingdom** struct to help work with and manage all of the knights in the kingdom.

Submission Details

Use the instructions on the following pages to create a small program that explores a more complex use of arrays and structures.

Submit the following files to OnTrack.

- Your program code
- A screen shot of your program running
- Answers to the accompanying questions

The focus of this task is on the tools and approaches to working with more complex data using structs and dynamic arrays, with matching functions and procedures to help ease the coding efforts.

Instructions

To get started:

1. Watch this week's video on updating the Kingdom Database code with additional types and dynamic arrays, and make the changes shown there. See the Update Kingdom Database video for details.

   Changes include:

   - A Kingdom struct with a vector of knights
   - An **add_knight** procedure to add a knight into the Kingdom
   - A **write_kingdom** procedure to output the details of the Kingdom and its knights
   - A **delete_knight** procedure to delete a knight from the kingdom

2. Now make the following additional changes:

   1. Add a **select knight** function.

      - Implement **int select_knight(const kingdom_data &kingdom)**
      - This will:
        - Show the user a list of the names of the knights, and their index (from 1 to n)
        - Ask the user to enter the index of the knight they want to select (or 0 for no knight)
        - Ensure the user has entered a valid index (i.e. an integer from 0 to n)
        - Return the index of a knight in the array (i.e. return the user supplied value - 1)

      This function will be useful as a tool to select an individual knight from our array within the kingdom. You will then be able to pass this knight into the relevant functions/procedures where the actual work is performed. For example, to write a selected knight you could use:

      ```
      int i;
      i = select_knight(my_kingdom); // assuming we have a `my_kingdom` var
      if ( i > 0 )
      {
          write_knight(my_kingdom.knights[i]);
      }
      ```

   2. Create a new `update_kingdom` procedure which shows a menu to allow the user to choose between the different system actions: add a knight, query a knight, delete a knight, update a knight, and displaying kingdom details

      - Adding will use `add knight`
      - Query knight is a new procedure that will get the user to select a knight, and then write that knight's details to the terminal
      - Deleting will need to be a new procedure - it can call **select_knight** and if the result is not -1, it can then use this in a call to `delete_knight`
      - Displaying will show the kingdom data - i.e. the name and all of the knights

   3. Build capability for **add weapon** and **delete weapon** to each knight.

> **Note:**
>
> For this step you will need to add a number of additional functions and procedures that are not explicitly listed here. Think about how you can best implement these features, and add appropriate functions and procedures to help with this.

- Add a dynamic array of strings called **weapons** to the Knight.
- Change `update_knight` to include the ability to **add a weapon**, and **delete a weapon**.
    - In *add weapon* have the user enter the details of the new weapon and add this to the knight's current weapons.
        - Store all items of **weapons** in uppercase only.
    - In *delete weapon* have the user select a weapon from a list, and remove the selection from the knights weapons.
        - If there is no weapon for the knight, tell the user by displying **No weapon is found** and exit *delete weapon*.
        - Else provide the weapons list when the user asks to delete a weapon.
- Change **write_knight** to output the up-to-date details of the knight's weapons in a list.

3. Answer the questions from the resources associated with this task.
4. Submit your program code along with a screenshot of it working.