EE441- Programming Assignment 2

Due Date: 02.12.2024, 23:59

For your questions: Preferably use the ODTUClass Forum
Doğu Erkan Arkadaş – arkadas@metu.edu.tr

This assignment consists of one part. You are going to create a makefile project for this part. You can also upload a PDF together with your code in your .zip file if you have special considerations for your code. Do not forget to write comments to your code as they are also graded. **Use the given code templates from ODTUClass, otherwise your answers will not be graded. You are NOT allowed to use existing data structures provided by the standard library such as std::vector or std::list, and algorithms such as std::sort.**

Simplified Mahjong [100 points]

Design and implement a simplified version of the Mahjong game using dynamic memory programming. The purpose of this homework assignment is to get you to familiarize yourself with dynamic memory management, pointers, and stacks.

Game Simplification and Rules:

1. Suits and Tiles
   - You will play the game using 3 suits of tiles, namely Characters, Bamboo, and Dot tiles. Suits will be abbreviated as C, B, and D.
   - Each tile will have a suit and a number assigned. For example, B5, D1, C9, etc.

		Numbers								
		1	2	3	4	5	6	7	8	9
Suits	Dots									
	Bamboos									
	Characters									

2. Tile Stack
   - Initially all the tiles will be in a stack called the **wall** (think of it the same as a deck of cards)

- Not every tile needs to be in the game, but existing tiles will have 4 copies in the stack. For example, an entire suit or some tiles of a suit (B5, C1, C9, etc.) may be missing from the stack.
- The player will draw tiles from the wall at each turn.

3. Player Hand
   - The player will hold 14 tiles in their digital hand, tiles will be kept sorted.
   - For suits assume (B<C<D). An example hand:
     B2 B2 B4 B5 C5 C6 C7 D1 D1 D5 D5 D7 D8 D9
   - After each draw if the player does not win the game, a tile must be discarded from the hand.
4. Sets
   - Only **triplets (three identical tiles)** are allowed (e.g., 3-3-3 of Circles).
   - Sets must be within the same suit; mixing suits is not allowed.
   - No sequences (ordered sets) or other combinations are used.
5. Win Condition
   - To win, the player must form 4 triplets (three identical tiles) and 1 pair (two identical tiles) after a draw.

You are going to implement 3 classes for this game: Tiles, Stack, and Hand. **Use the given templates for the classes uploaded on ODTUClass and follow the given steps to complete the implementations**. Use `<stdexcept>` library for the exceptions.

A parser is also given in the ODTUClass for reading tiles from a.txt file into the Stack you will implement to start the game.

Tile [10 points]

Use the provided "Tile.hpp" file as a template, only fill in the functions and do not modify them. Tile class has 2 variables, `m_suit` and `m_number`.

Notice that you have a "friend" function that overloads the printing operator. Your friend allows you to print your tiles as such:

std**::**cout **<< tile;**

1) **[5 points]** Implement the default and parameterized constructor for the class

2) **[5 points]** Overload all the given comparison operators (==,!=,<,<=,>,=>) such that it first looks at the suits using (B<C<D) and if both Tiles have the same suit it compares the numbers.
   Hint: In ASCII B=66, C=67, and D=68.

Stack [35 points]

Use the provided "Stack.hpp" file as the template, only fill in the functions, do not modify the prototypes, and do not add new members. Stack class has 3 variables, `m_capacity` for the Stack's current capacity, `m_size` to keep track of how many elements are in the stack, and `m_data` pointer to allocate memory for the elements. **Update the variables accordingly in every function.**

You have another friend function to print the whole stack, this time it is just for your debugging purposes.

1) **[7 points]** Implement the default and parameterized constructors, and the destructor. The default `m_capacity` is 0 and `m_data` is **`nullptr`**.

2) **[2 points]** Implement the `is_empty` function which returns **`true`** if the stack is empty.
3) **[7 points]** Implement the `reserve` function which allocates a new array with the size `new_capacity` and copies the elements of the old array to the new array.
Note that the new array can be smaller in which case the new array will hold only a portion of the old array.

4) **[7 points]** Implement the `push_back` function which will push an element to the top of the stack. If the `m_capacity` is reached function should reserve double the `m_capacity` and then add the element.
Hint: If `m_capacity` is 0 rather than double 0 and get 0 the function should make it 1

5) **[7 points]** Implement the `pop_back` function which will pop an element from the top of the stack. If the stack is empty function should throw `std::out_of_range` error.

6) **[5 points]** Implement the `clear` function which will delete the `m_data` and assign it to **`nullptr`**. Also, reset `m_capacity` and `m_size`.

Hand [50 points]

Use the provided "Mahjong_Hand.hpp" file as the template, only fill in the functions, do not modify the prototypes, and do not add new members. `Hand` class has 2 variables, `m_tiles` will hold a static array of pointers for `Tiles` currently in hand, and `m_handSize` holds the currently held number of `Tiles`.

1) **[5 points]** Implement the default constructor and the destructor. The default `m_handSize` is 0.
Note: The destructor should deallocate memory for all the tiles pointed to in the `m_tiles`.

2) **[5 points]** Implement the `insert_before` private function. This function adds a new `Tile` to the `tiles` before the given index. This function throws `std::out_of_range` error if the `m_handSize` exceeds `MAX_HAND_SIZE` macro with the insertion. Example:
Hand: B1,B2,B4,C2,C5, nullptr, … -> `insert_before` (3, C1) -> B1,B2,B4,C1,C2,C5, nullptr, …

3) **[5 points]** Implement the `add_tile` function. This function adds a new tile into the hand such that the tiles held are ordered after the insertion. Use the `insert_before` function you designed in the previous step.

4) **[5 points]** Implement the `pop` private function. This function pops the `Tile` pointer with a given index from the `m_tiles` array.
Hand: B1 B2 B4 C1 C2 C5 nullptr … -> `pop` (3, C1) -> B1 B2 B4 C2 C5 nullptr …

5) **[5 points]** Implement the `display_hand` function. This function will display all the tiles currently held in the hand. The function should print tiles as sorted as your hand should be sorted at any point. Example:

B1 B1 B2 B2 B9 C2 C6 C6 C7 C8 C8 C9 D2 D2

6) **[10 points]** Implement the `discard_tile` function. This function will discard the most useless tile in the hand using a simple heuristic approach. Remember that 4 triplets and a pair wins the game. The function will follow the steps below in order.
   I. Look for quadruplets in the hand. If a quadruplet exists the function will discard one of the tiles from the quadruplet.
   II. Look for single tiles in the hand, which are tiles with no duplicates in your hand. If a single tile exists the function will discard it.
   III. Look for pairs in the hand. If the previous two steps did not find anything the hand will have at least 3 pairs and the function will discard a tile from one of them.

7) **[10 points]** Implement the `check_win_condition` function. This function will return **`true`** if the hand has 4 triplets and a pair indicating a game win. The function will return **`false`** otherwise. **Hint:** The pair can be at the first two or the last 2 tiles held in the hand as edge cases for you to consider.

8) **[5 points]** Implement the `clear` function. This function will delete all the objects the pointers in the pointer array `tiles` point to and assign **`nullptr`** to the pointers. Also, reset the `m_handSize`.

Play The Game / Show Your Work [5 points]

1) **[5 points]** Use the given parser and main file to play 3 rounds. The Stack is initialized using the parser and the .txt files after which the initial 14 tiles are drawn. The player draws and discards tiles until no tile is left in the stack or the player wins. This ensures that the player will always win if the discard tile function works properly.

   If you were unable to complete your implementation, you can still present the progress you made in your own way. Full credit will be given for this part as long as you show that either the Hand or Stack functionality is working.