Introduction and purpose

The goal is to familiarize yourself with how basic sorting works in Python and C++ respectively, and methods for this.

Task

The task is to implement:

• A method of sorting a deck • Necessary
tests

You can choose to implement in Python or C++ (or both if you want).

Python

deck.py

• Your task is to implement the function **insert(card_list, card).** This takes a list of cards (card_list) and must insert a card (card) in the right place in this list. Then the list must be returned. • The method will be called 52 times, once for each card in a deck. The sort() method takes a card from the old deck and puts it into the new deck using insert().

The main function in the current code only calls the method and prints to the console. To pass, it is enough to insert a few tests in the main function that test your function (Deck.insert). You can use assert as tests. What might be appropriate tests? (Size, test positions etc). For bonus points, see below

test_deck.py

Shell for implementing tests (see bonus points)

C++

The structure is structured as you often see in C++ programs with an include directory and a main file. There is a CMake file to be able to build with CMake. (Instructions in README file)

main.cpp

the main function creates a deck and runs the sort function and prints the deck in the prompt.

If you want to do the minimum possible for approved, you can insert assertions in main function as tests. If you implement test cases instead, you don't need it.

deck.cpp

You must implement the **static** function:

```
void Deck::insert(vector<Card> &cardlist, Card card)
```

- Input is a list of cards (cardlist of type vector<Card>) and must insert a card in the right place in this list. The list is sent as a reference, so nothing needs to be returned.

- The method will be called 52 times, once for each card in a deck. The sort() method takes a card from the old deck and puts it into the new deck using insert().

include/deck.h

Here you will find the definition of the Card and Deck class.

test_deck.cpp

Shell for implementing tests. I have set up the structure to use Googletest the framework.
If you would rather use another framework (Boost.Test, CTest, CUnit, CppUTest), that's ok. In that case, you get to set up the structure for that framework.

Bonus data

Bonus task 1:

(A bonus point before the exam)

Write test cases for all methods. Either in python or C++.

Python - test_deck.py

• In addition to the sorting algorithm, you must also write tests for the entire code base. • Implement appropriate tests for your sorting algorithm • - Deck.insert(card_list, card)) • Also implement tests for the other methods: • - Card.__init__() • - Card.__eq__() • - Card.__lt__() • - Card.__gt__() • - Deck.__len__() • - Deck.sort()

• - Deck.take()
• - Deck.put() •
Write a short justification for your tests

C++ - test_deck.cpp

For C++, I set up the structure using GoogleTest. You can use another unit test framework if you want (then you can change the structure yourself).

• Deck.insert(vector<Card> &cardlist, Card card) • Constructor
• Card == operator • Card < - operator • Card > - operator •
Deck.size() • Deck.sort() • Deck.take() • Deck.put(Card card)

Bonus task 2:

(A bonus point before the exam)

The following must be implemented (either in C++ or python):

• Add three jokers to the deck (to be added when the deck is created) • Add at least three of the following features:
o sort_by_suit – Sort so that cards of the same suit (suit) are grouped together. Within the color they must be sorted in order
o sort_by_value – Sort the cards in numerical order. Note: The sorting must always be the same, so you must also take into account the color (suit) o pick_by_random – Select (and remove) a card randomly from the deck. o deal(int n) – Deal the deck to n number of people. Here you have to think about how to implement this, so that you get an even number of card piles. (Can be separate / static function) o remove_duplicates – Find and remove cards that are duplicates o remove_jokers – Remove the jokers

Accounting

The source code must be pushed to a separate repository on Codeberg or GitHub. Write name in README.md in repot's main-branch (master).

The submission takes place by sending a link to the gitrepot into Omniway.