Week 12 – Session 1: Templates

In this laboratory session you are going to explore generic programming and parametric polymorphism with C++ using templates. After completing all the challenges, you will develop a simplified version of the card game Bridge. Let's get on with it.

Challenge 1: The smart vector class

In this challenge, we will use the power of C++ template to create our own version of smart vector class. Our smart vector class will include additional user-defined functions as compared to the standard library defined vector class.

Before you begin, please clone the github repository of this project:
`https://github.com/eiramlan/Bridge`

Task 1 Complete the smart vector class template - duration 15 minutes

Complete the smart vector class template. Similar to the `std::vector`, the smart vector class will accept any data type (e.g., `integer`, `double`, `float`, `char`, `string`, `struct` and even `object`). You need to define a constructor to initialise your vector. We are using the `std::vector` object `SmartVector_` as data attribute. There will be multiple versions available in the class definition.

We have two constructors.

- Default constructor `SmartVector(const std::vector<T>& t)` that accepts the std::vector template, and

- `SmartVector(const T t[], int s)` that accepts an array template and the array size.

In addition to the constructor, we also need to define some operators for our SmartVector class.

The smart vector class will include these functions:

```
(1) Push – push an array of object inside the vector
(2) Sum – Add all elements inside vector
(3) Sort – Sort all elements inside vector in ascending order
(4) Shuffle – Shuffle all elements inside vector – use random shuffle
(5) Search – Search all elements matching the search argument
(6) At – Return the element at index (similar to std::vector.at())
(7) Size – Return the size of the vector (similar to std::vector.size())
(8) Range – Return a range of element from vector
(9) Erase – Delete an element inside vector given index
```

Details about these functions will be given in the next task.

Task 2 Function template for Push operation - duration 5 minutes

The push function accepts an array template and array size as arguments. The function will `push_back()` the elements inside the array template into the `smartVector` attribute, similar to `std::vector.push_back()`.

Task 3 Function template for Sum operation - duration 5 minutes

The sum function will add all elements inside the vector and return the total amount. For char and string, the total summation will be a word (in the case of char), and sentences (in the case of string).

Task 4 Function template for Sort - duration 5 minutes

The sort function will sort the elements inside the vector in ascending order.

Task 5 Function template for Shuffle - duration 5 minutes

Use `random_shuffle` to shuffle the all the elements inside your vector. You should refer this documentation for details:

https://en.cppreference.com/w/cpp/algorithm/random_shuffle

Task 6 Function template for Search - duration 5 minutes

Given a value as argument, this function will return all the indexes of the elements matching the argument value

Task 7 Function template for At - duration 5 minutes

Return the element given index. This function is similar to the `std::vector.at() const`

Task 8 Function template for Size - duration 5 minutes

Return the size of `smartVector_`. This function is similar to the `std::vector.size() const`

Task 9 Function template for Range - duration 5 minutes

Return a range of elements from the vector. The function will accept to arguments indicating the start and end index of the range.

Task 10 Function template for Erase - duration 5 minutes

Erase the element in the vector given index.

Once you have completed the smart vector class, build your codes and test the function implementation. We will use our smart vector class to construct A game of bridge in the next challenge.

Challenge 2: A game of bridge

In this challenge, we are going to build a C++ version of the card Bridge using our `SmartVector` class. The simplified version will not take into account Bridge contract using bidding.

Bridge uses a pack of 52 playing cards. There are 4 suits (spades, hearts, diamonds, clubs). The 13 cards are Ace, 2 to 10, Jack, Queen, and King. The Ace is the highest card, followed by the King, Queen, down to the 2. In bridge there are 2 teams of 2 players (North & South) and (East & West). Before moving on the gameplay, let us focus on the essentials.

Task 1 Create a struct for Card - duration 15 minutes

Create a struct called Card to represent a card. It will need 2 fields: one to store the card's value (`value_: int`) and one to store the suit (`suit_: string`). Struct is a data structure (similar to class), therefore we should be able to assign `<struct>` Card using our `smartVector`. However, in order to make use of the function in `smartVector` that involves operators (e.g., `==, >`) we need to overload these operators, so the compiler knows how to handle them.

You also need to write the definition for the operator overloading (less than) `bool operator<(const Card& c) const` and the operator equality, `bool operator==(const Card& c) const`.

The operator less- than will compare the `value_` of the Card, and the equality operator will compare the `suit_` value of the Card.

Task 2 Create a class for Player - duration 15 minutes

You need to create a class player with these attributes, `hand_` of the type `SmartVector<Card>` and `played_` of the type `Card`. The `hand_` data is the random 13 cards that each player has during the game. The data `played_` is the card that will be played during each player turn.

Player has a custom constructor that will accept a `SmartVector<Card>` argument. Use initialiser to assign the data attribute to the argument.

You also need to define these public functions:

```
(01) PlayCard – Play card action for each player
(02) GetTrumpSuit – Return the suit for the trump mode
(03) GetHandTotal – Calculate the total values for player hand
```

```
(04) RemoveCard – Remove card from hand_
(05) GetPlayed – Return played_ (Getters)
```

We need to write the function definition using the pseudocodes provided in the skeleton codes.

Task 3 Definition for `PlayCard` - duration 20 minutes

In bridge the teams try to win tricks (a set of 4 cards, one put down by each player sequentially). Players should try to "follow suit" - e.g. if the first card of a trick is a Spade, subsequent players must put down a Spade if they have one. In our game the winner of the trick is the person who puts down the highest card with the suit of the first card.

A function to play a card will need to know what cards are available to play, i.e., the best card put down so far. Write a function to implement this with the following prototype

```
bool PlayCard(Card bestCard, std::string trump, int turn);
```

Make it return true if the card put down is the best card so far. First find out if the hand has any cards the same suit as the first card. Then find out whether the best card in that suit is better than the best card so far played. If this is true, set `played_` to the best card with the same suit for that player. Then return `true`.

If a player is unable to follow suit, then check if that player can play a trump card. In bridge a suit is often chosen to be the trump suit. If a player can't follow suit they are allowed to "trump" the card by putting down a card that's a trump. If subsequent players also can't follow suit they are allowed to "overtrump" - put a higher trump card down. A trump card carries higher values than any suit cards (currently in play). You should return `true` when this is the case.

If a player is unable to follow suit, and cannot play a trump card, then the player should play the lowest value card that he/she has. You should return `false`.

Task 4 Definition for `GetTrumpSuit` - duration 15 minutes

Write a function that given a hand, return the highest number of suits. For instance, if a player is given (6 spades, 3 hearts, 2 clubs and 2 diamonds), the function will return the value of “Spade”. This will determine the trump suit for the round.

Task 5 Definition for `GetHandTotal` - duration 10 minutes

In bridge people often use an evaluation system where an Ace is worth 4 points, a King 3, a Queen 2 and a Jack 1. Write a function that given a hand of cards evaluates it, returning the number of points it's worth. To check whether it's working you could see whether the sum of all the hands' points is 40.

Task 6 Definition for `RemoveCard` - duration 10 minutes

The card it puts down will have to be removed from the hand. Write a function to remove the card that has been played.

Task 7 Write the game play - duration 30 minutes

In order to begin, declare 4 objects representative of the 13 cards and suits that we will use in this game. You should write a function `Card* PopulateDeck(string suit)` that will generate all the 13 cards for each suit, and return these cards are array of Cards.

Create a `SmartVector` with the type Card, that will store all 52 cards generated using `PopulateDeck`. Shuffle the deck using the `SmartVector::Shuffle` function and then distribute 13 cards to each player (you can use the `SmartVector::Range` function to do this).

Write a function `int* FindMax(int a, int b, int c, int d)` that will find the best hand value. In our Bridge version, the player with the best hand value will start the game and will determine the trump suit.

We'll use the following gameplay, player with the most hand total puts down his best card. Subsequent players will try to follow suit. If they can't, they can play a trump card or any random card. Playing trump card will have priority. If they can follow suit then they put down the lowest card they have if they can't beat the currently best card, otherwise they put down their best card.

Add some code to print out how many tricks each team wins (players 0 and 2 are one team, players 1 and 3 the other). You can do this by using the function `string DisplayCard(int winner, Card& c)`. Check that a total of 13 tricks are won. The team with the most number of tricks wins the game.