Overview of the Standard Template Library

Exercise 1: STL Containers

One of the features of STL are the containers. In this exercise you will practice with the different STL containers. In the main program:

- Create a *list* of doubles and add some data. Use the *front()* and *back()* functions to access the first and last element.
- Create a *vector* of doubles and add some data. Then use the index operator to access some elements. Also make the vector grow.
- Create a *map* that maps strings to doubles. Fill the map and access elements using the square bracket operator.

Exercise 2: STL Iterators

Using iterators you can iterate a STL container without knowing which container it is. In this exercise you create a function that calculates the sum of a container with doubles.

- Create a template function called *Sum()* that accepts the template argument *T* as input and returns a *double*. The template argument will be a container.
- In the implementation get an iterator (`T::const_iterator`) for the end. Then create a loop that iterates the container *T* and adds all values. Finally return the sum.
- In the main program, call the *Sum()* function for the different container from the previous exercise.

The *Sum()* function created calculates the sum of the complete container. Also create a *Sum()* function that calculates the sum between two iterators. The function then uses the template argument for the iterator type and accepts two iterators, the start and end iterator.

Exercise 3: STL Algorithms

STL already contains many algorithms that work with containers. Use the *count_if* algorithm to count the number of elements smaller than a certain number. The *count_if* function accepts a functor. Thus pass it a global function that checks the double input is smaller than a certain value.

Replace the global checking function, by a function object. This is a class that overload the round bracket operator that in this case has the same signature and functionality as the global function you created previously. Only in this case the value to check for should not be a ‘literal’ value, but taken from a data member that was set in the constructor of the function object.