

# Containers Workshop

# Preface

- This workshop will give you practice in using containers
- If you need to look up details of how to use them, I recommend the C++ Reference Site
  - <https://en.cppreference.com/w/cpp/container>
  - (This website is also available in Chinese, French, German, Italian, Japanese, Portuguese, Russian and Spanish)

# URL Container

- Define a class to hold an ordered history of URLs (as might be used in a web browser). Use an STL container to store the URLs
- Ensure that a URL is added to the history only if it is not in the history already. If it is, it should be moved to the beginning of the history
- Write a program to exercise your class

# String input

- Write a program that reads strings entered by the user, stores them in a vector and prints out the strings in the order they were entered
- Change your program to use a list instead of a vector
- Change your program to use a deque instead of a list
- Change the list and deque versions to store the strings in reverse order, without performing any manipulation on their elements

# Container choice

- Which sequential container would you use for:
  - A databank of historical securities prices, in chronological order. Numerical processing will be repeatedly performed on the container and this must be done efficiently. The prices may occasionally be altered (to correct bad data) but will never be added to or removed
  - A queue of people entering a venue. People join at the back of the queue and leave, in arrival order, from the front. There should be provision for "VIP's" to go straight to the front of the queue
  - Storing users who are connected to a very busy website. The users will be added when they log on and removed when they log off (or are timed out)
- Explain your choices

# Sets

- Write a program that reads strings entered by the user, stores them in a set and prints out the strings
  - What happens if the user enters duplicate strings?
- Write a program that displays the number of distinct words in some text using a set

# Maps

- Write a program that reads words from input
- Store the value and length of each word in a suitable `std::pair`
- Store each pair in
  - A vector
  - A map whose key and value have the appropriate types
- Print out all the elements of the vector and the map

# Maps contd

- Write a Family class which has
  - A vector of children's names as a member
  - A member function to add a child
  - A member function to print out all the children in the family
- Define a map whose key is a family name and whose value is a Family
- Write a program which populates this map and prints each family's name followed by its children's names



# Multimaps

- Define a multimap whose key is a family name and whose value is a child's name (i.e., one element per child)
- Write a program which populates this multimap and prints each family's name followed by its children's names
- Modify this program to use an `unordered_multimap`