

# MODULE 1 / UNIT 5

## USING STL CONTAINERS



# Today

- Abstract data types
- Container types in C++ template libraries
  - `std::vector<T>`
  - `std::pair<T1, T2>`
  - `std::map<T1, T2>`
- Applications
  - Counting most frequent k-mers
  - Making an word unscramble game

# Abstract Data Types (ADT)

- Mathematical **model** for data types
- Its behavior (**semantics**) is defined by a set of values and operations.
- Theoretical concept that allows algorithms to **separate** from a particular method of implementation.

# Example **Container** ADTs

*Sensitive to input orders*

- Stack
- Queue
- List
- Priority queue

*Insensitive to input orders*

- Set
- Map

# Example Container ADTs

## *Sensitive to input orders*

- Stack : **LIFO**

*push, pop*

- Queue : **FIFO**

*enqueue, dequeue*

- List : **random access**

*front, back, next,  
insert, remove*

- Priority queue

*insert\_with\_priority,  
pop\_highest\_priority*

## *Insensitive to input orders*

- Set : **key only**

*insert, remove, has\_key  
front, back, next*

- Map : **(key, value) pair**

*insert, remove  
has\_key, get\_value  
front, back, next*

# Data structure

- A specific way to **organize** the data in a computer.
- Key factors : **correct** and **efficient** algorithms
  - .. to store values and perform operations
- STL containers include several data structures implementing ADTs

# Container data structure in C++ STL

## *Sensitive to input orders*

- Stack : `std::stack`
- Queue : `std::queue`  
`std::deque`
- List : `std::list`  
`std::vector`
- Priority queue  
`std::priority_queue`

## *Insensitive to input orders*

- Set : `std::set`  
`std::unordered_set`
- Map : `std::map`  
`std::unordered_map`

# Using `std::vector<T>`

- STL vector is a flexible-sized array that can contain an arbitrary type.
- Because it is using “template”, the data type of the elements must be specified in definition

```
std::vector<std::string> example_str_array;  
std::vector<double> example_dbl_array(10, 0);
```

- Similar to C-style array, elements can be access using `operator[ ]`
- A new element can be appended using `push_back()` function
- The size of array can be changed using `resize()` function
- See more at <http://www.cplusplus.com/reference/vector/vector/>



# An example code

```
#include <Rcpp.h>
#include <fstream> // need to use std::ifstream
#include <iostream> // need to use std::cout
#include <vector> // need to use std::vector
using namespace Rcpp;
using namespace std;

// [[Rcpp::export]]
void loadWords(string filename) {
    ifstream ifs(filename);
    string s;
    vector<string> vecstr; // a vector of string
    while ( ifs >> s )
        vecstr.push_back(s);
    ifs.close();
    cout << "Finished loading " << vecstr.size() << " words" << endl;
    cout << "The first word is " << vecstr[0] << endl;
    cout << "The last word is " << vecstr[vecstr.size()-1] << endl;
    cout << "The word in the middle is " << vecstr[vecstr.size()/2] << endl;
}
```

# An example output

```
loadWords('dolch.314.txt')
```

---

```
Finished loading 314 words  
The first word is a  
The last word is your  
The word in the middle is like
```

---

```
loadWords('common.2198.words.txt')
```

---

```
Finished loading 2198 words  
The first word is a  
The last word is zone  
The word in the middle is likely
```

---

```
loadWords('mit.10000.words.txt')
```

---

```
Finished loading 10000 words  
The first word is a  
The last word is zus  
The word in the middle is lanka
```

---

**DIY R Notebook**  
**bios615\_1\_5.Rmd**  
**section #1**

# Using `std::pair<T1, T2>`

- STL pair is a simple container that can contain a pair of elements with different data types.
- The data types of the elements must be specified in definition  
`std::pair<std::string, int> name2int;`
- Each element can be accessed using `.first` and `.second` member variables.
- See more at <http://www.cplusplus.com/reference/utility/pair/pair/>

# An example code & an example output

```
#include <Rcpp.h>
#include <utility> // needed for std::pair
#include <string>   // needed for std::string
#include <iostream> // needed for std::cout

using namespace Rcpp;
using namespace std;

// [[Rcpp::export]]
void pairTest(string uniqname, int umid) {
    // constructing a std::pair object
    pair<string,int> name2id(uniqname, umid);
    cout << "Name : " << name2id.first << endl;
    cout << "UMID : " << name2id.second << endl;
}
```

```
pairTest("hmkang",12345678)
```

```
Name : hmkang
```

```
UMID : 12345678
```

**DIY R Notebook**  
**bios615\_1\_5.Rmd**  
**section #2**

# Using `std::map<T1, T2>`

- STL map is..
  - An associative container insensitive to the input order.
- **Contains a set of (key,value) pairs with different data types.**
  - The data types of the elements must be specified in definition  
`std::map<std::string, int> name2id;`
  - Key must be unique within the container, but values do not have to.

# Using `std::map<T1,T2>::iterator`

- To represent a specific position within the container, iterator is used

```
map<string,int>::iterator it1 = name2id.begin();  
map<string,int>::iterator it2 = name2id.end();
```

- Iterator can be used to enumerate each element

```
for(it=name2id.begin(); it!=name2id.end(); ++i) {  
    cout << "Name : " << it->first;  
    cout << "UMID : " << it->second << endl;  
}
```





# Key features in `std::map<T1, T2>`

- $O(\log n)$  time complexity for search, insert, erase.
- `find()` member function can tell whether a key exist in the container

```
if ( name2id.find("hmkang") == name2id.end() )  
    cout << "Username hmkang does not exist" << endl;
```
- `operator[]` can be used to retrieve/assign values associated with keys.

```
name2id["hmkang"] = 12345678;
```
- Inexact queries are possible using `lower_bound()`, `upper_bound()`;
- See <http://www.cplusplus.com/reference/map/map/> for further details.

# Application : finding most frequent substrings

- **Given**

- A list of alphabetical words
- $k$  : the length of substring of the words

- **Goal**

- Which  $k$ -mer substrings appear most frequently across the list of words?
  - Output all  $k$ -mers if there are ties
- What is the number of appearance of the most frequent  $k$ -mer substrings?

# Building a map of k-mer counts

```
#include <Rcpp.h>
#include <string>
#include <map>
#include <fstream>
#include <iostream>
using namespace Rcpp;
using namespace std;
//[[Rcpp::export]]
void topkmer(string filename, int k) {
    ifstream ifs(filename); // read a file
    // return with error msg if file fails to open
    if ( !ifs.is_open() ) {
        cerr << "Cannot open file " << filename << endl;
        return;
    }
    string s;
    map<string,int> counts; // keeps count of each kmer
    while ( ifs >> s ) { // read a word from the file
        for(int i=0; i < (int)s.size()-k+1; ++i)
            ++counts[s.substr(i,k)];
    }
}
```

**counts** variable  
contains a map  
between each k-  
mer and its  
number of  
occurrences

# Identifying most frequent k-mers

```
vector<string> top; // keeps most frequent kmers
int topcount = 0; // keeps the count of top kmers
map<string,int>::iterator it;
for(it=counts.begin(); it != counts.end(); ++it) {
    if ( it->second > topcount ) {
        topcount = it->second; // update topcount
        top.clear(); // clear if new topcount comes in
    }
    if ( it->second >= topcount )
        top.push_back(it->first); // add to the top list
}
cout << "These " << k << "-mers appeared " << topcount;
cout << " times" << endl;
for(int i=0; i < (int)top.size(); ++i)
    cout << top[i] << endl;
}
```

# Running examples

```
topkmer("dolch.314.txt",2)
```

These 2-mers appeared 22 times  
er

```
topkmer("dolch.314.txt",3)
```

These 3-mers appeared 11 times  
the

```
topkmer("dolch.314.txt",4)
```

These 4-mers appeared 5 times  
ther

```
topkmer("common.2198.words.txt",4)
```

These 4-mers appeared 84 times  
tion

```
topkmer("common.2198.words.txt",5)
```

These 5-mers appeared 37 times  
ation

```
topkmer("common.2198.words.txt",6)
```

These 6-mers appeared 9 times  
ection

```
topkmer("common.2198.words.txt",7)
```

These 7-mers appeared 4 times  
ference  
lection  
present

**DIY R Notebook**  
**bios615\_1\_5.Rmd**  
**section #3**

# A few more notes on STLs..

- Sorting a container is as simple as..

```
#include <algorithm>
std::sort( v.begin(), v.end() )
```

- Shuffling container is as simple as..

```
#include <utility>
std::random_shuffle(v.begin(), v.end() )
```

- Mixing STL containers are working seamlessly

- A vector of maps
- A vector of pairs
- A map between a string and a vector
- A map of maps

# Using R function in C++ with **Rcpp**

- **Function** type can represent an arbitrary R function.
- Example from <http://gallery.rcpp.org/articles/r-function-from-c++/>

```
// [[Rcpp::export]]  
NumericVector callFunction(NumericVector x, Function f) {  
    NumericVector res = f(x);  
    return res;  
}
```

- Or find function by name and **Environment** type

```
Environment base = Environment("package:base");  
Function readline = base["readline"];  
Function as_character = base["as.character"];
```

- Mind that this might be very **SLOW!**



# Making an word **unscrambler** game



sprbreiresa

\_\_\_\_\_



bnaaan

\_\_\_\_\_



ycehrr

\_\_\_\_\_



altwof otruyg

\_\_\_\_\_

*Image from chefsolus.com*

# Word unscrambler **specs**

1. Read a **file** containing a list of words
2. Pick a **random** word
3. **Shuffle** the letters of the chosen word
4. Ask users to **type** their guess
5. **Respond** “Correct!” or “Sorry!” based on correctness
  - Mind that there could be multiple answers
  - Need to display all correct answers when responding.
6. **Repeat** steps 2-5 until the user types ‘q’ to quit

# A running example

```
> unscramble("common.2198.words.txt")
```

```
Welcome to WORD UNSCRAMBLE 615!
```

```
Unscramble erlhesewe (or type q) : elsewhere
```

```
Correct!
```

```
Here are all possible answers: elsewhere
```

```
Unscramble igrame (or type q) : image
```

```
Correct!
```

```
Here are all possible answers: image
```

```
Unscramble noxtinsee (or type q) : extension
```

```
Correct!
```

```
Here are all possible answers: extension
```

```
Unscramble cta (or type q) : act
```

```
Correct!
```

```
Here are all possible answers: act cat
```

```
Unscramble gid (or type q) : q
```

# Strategy 1

1. **Read** the input file and **store** all set of words in a map
2. **Pick** a random word, **shuffle** it, **ask** a question, and **get** input
3. Enumerate all possible **permutations** of the question words
4. For each permutation, **check** if it can be found from the map
5. Add all permutations that are **valid** words to possible answers
6. **Check** if input word is in one of the possible answers.
7. **Repeat** steps 2-6 until q is typed in step 2

# Strategy 1

1. **Read** the input file and **store** all set of words in a map
2. **Pick** a random word, **shuffle** it, **ask** a question, and **get** input
3. Enumerate all possible **permutations** of the question words
4. For each permutation, **check** if it can be found from the map
5. Add all permutations that are **valid** words to possible answers
6. **Check** if input word is in one of the possible answers.
7. **Repeat** steps 2-6 until q is typed in step 2

$$T(n, l) = O(l! \log n)$$

# Strategy 2

1. **Read** the input file and **construct** the following map
  - [sorted list of letters] => [vector of corresponding words]
2. **Pick** a random word, **shuffle** it, **ask** a question, and **get** input
3. **Look** up the map to find all possible answers
4. **Check** if input word is in one of the possible answers.
5. **Repeat** steps 2-4 until q is typed in step 2

# Strategy 2

1. **Read** the input file and **construct** the following map
  - [sorted list of letters] => [vector of corresponding words]
2. **Pick** a random word, **shuffle** it, **ask** a question, and **get** input
3. **Look** up the map to find all possible answers
4. **Check** if input word is in one of the possible answers.
5. **Repeat** steps 2-4 until q is typed in step 2

$$T(n, l) = O(\log n)$$

# Implementation - setting up

```
#include <Rcpp.h>
#include <string>
#include <map>
#include <iostream>
#include <fstream>
#include <algorithm>
using namespace Rcpp;
using namespace std;
// [[Rcpp::export]]
void unscramble(string filename) {
    ifstream ifs(filename); // read file
    // return if file fails to open
    if ( !ifs.is_open() ) {
        cerr << "Cannot open file " << filename << endl;
        return;
    }
}
```



# Constructing a map

```
string s;
map<string,vector<string> > bag2words; // string -> vector of strings
vector<string> words;
while ( ifs >> s ) { // read a word from the file
    string b = s;      // clone the string
    sort(b.begin(), b.end()); // ignore orderings of letters
    bag2words[b].push_back(s); // map answer -> problems
    words.push_back(s);
}
Environment base = Environment("package:base");
Function readline = base["readline"];
Function as_character = base["as.character"];

cout << "Welcome to WORD UNSCRAMBLE 615!" << endl;
```

```

while(true) {
    string q = words[rand() % words.size()]; // pick a word
    sort(q.begin(), q.end()); // get the key value
    vector<string> answers = bag2words[q]; // obtain all answers
    random_shuffle(q.begin(), q.end()); // create a quiz
    string input = as<string>(as_character(
        readline("Unscramble " + q + " (or type q) : ")));
    if ( input == "q" ) break;
    bool correct = false;
    for(int i=0; i < (int)answers.size(); ++i) {
        if ( input == answers[i] )
            correct = true;
    }
    if ( correct ) { cout << "Correct!" << endl; }
    else { cout << "Sorry!" << endl; }
    cout << "Here are all possible answers:";
    for(int i=0; i < (int)answers.size(); ++i) {
        cout << " " << answers[i];
    }
    cout << endl;
}
}

```

**DIY R Notebook**  
**bios615\_1\_5.Rmd**  
**section #4**

# Summary

- `std::vector<T>` in `<vector>`
- `std::pair<T1,T2>` in `<utility>`
- `std::map<T1,T2>` in `<map>`
- `std::sort` in `<algorithm>`
- Using R function in **Rcpp**

# Reading Material

- **Eubank & Kupresanin : Chapter 9**