STL
(Standard Template Library)

ens-lal

2013

Introduction

STL is the standard Library for C++

- part of the definition of the language
- it is a `header` -only library (nothing to link against)
- all the public components are defined and folded under the std:: namespace
- built on the concept of templates allowing to parametrize (and possibly optimize) code on the type of the elements being used
- rather efficient

STL concepts

The STL is composed of mainly 3 types of elements and concepts:

- containers: hold data
- iterators: manipulate and iterate over the containers' data. fetch data and go to the next item.
- algorithms: manipulate and modify data from containers, thru the iterators

Example:

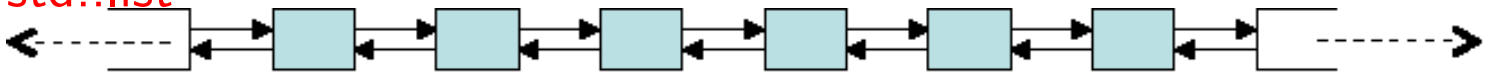
- std::string: a container of characters (with copy/value semantics), onto which algorithms such as std::rotate, std::remove_if can operate.

Containers

- an algorithmic structure providing a means to:
  - organize a set of data with the same type in a sequence
  - traverse these data
- *Examples:*
  - doubly-linked lists std::list
  - arrays std::vector
  - sorted lists std::set
  - association container std::map
- the definition (*ie:* the blueprint) of a container is independant of the type of objects being contained
- one specifies what it contains when one creates the new variable
- memory is dynamically and automatically allocated.

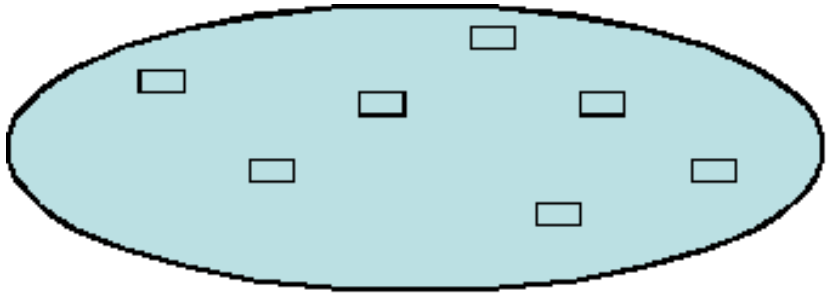
Containers (cont'd)

- Sequence containers (position of the item independant of its value)
  - std::list

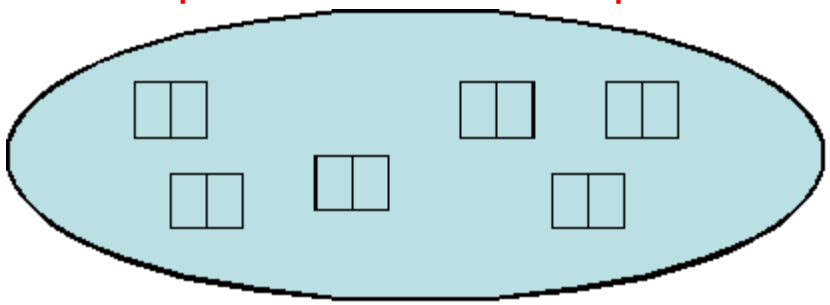


  - std::vector

- Associative containers (position of the item dependant of its value)
  - std::set and std::multiset



  - std::map and std::multimap



List

```cpp
#include <iostream>
#include <list>       // get the definition of std::list
int main(int argc, char **argv) {
  std::list<int> l;  // list is under the std:: namespace

  l.push_front(3);   // insert an element (in front)
  l.push_front(2);   // ditto
  l.push_back(4);    // insert an element (at the back)

  l.pop_front();     // remove first element
  l.pop_back();      // remove last element

  std::cout << l.front() << std::endl; // access first element
  std::cout << l.back()  << std::endl; // access last element
  return 0;
}
```

Vector

```cpp
#include <iostream>
#include <vector>       // get the definition of std::vector
int main(int argc, char **argv) {
  std::vector<int> v;

  v.push_back(3);       // insert an element (at the back)
  v.push_back(4);

  std::cout << v[0] << std::endl;      // access first element

  v.at(1) = 5;                         // modify 2nd element
  std::cout << v.back() << std::endl; // access last element

  v.clear();                           // remove all elements
  return 0;
}
```

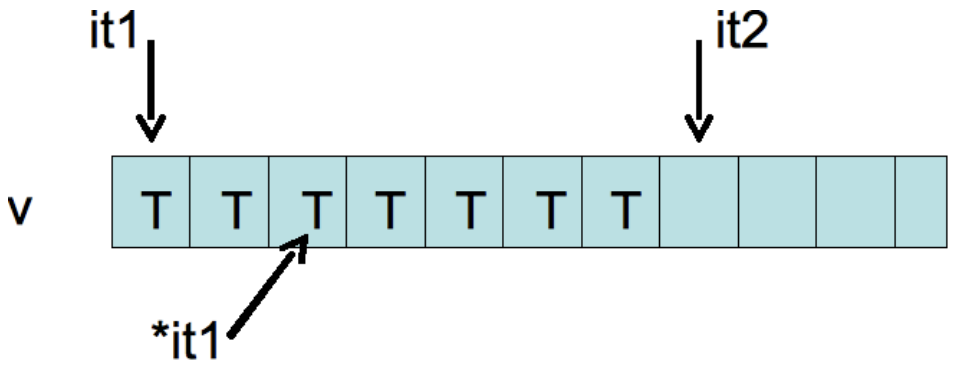
A few common methods

- clear the container
  - void clear();
  - calls the destructor for each contained object in the container
  - deallocate memory used by the container
- get the number of elements
  - int size();
- test if a container is empty
  - bool empty();

Iterators

- a generalization over the notion of pointers
- provide a means to traverse the elements of a container, w/o knowing *a priori* the type of the container

```
#include <vector>
{
  std::vector<int> v = get_some_vector();
  std::vector<int>::iterator it1 = v.begin();
  std::vector<int>::iterator it2 = v.end()
}
```



Iterators (cont'd)

```
#include <iostream>
#include <list>
int main(int argc, char **argv) {
  std::list<int> lst;

  for (unsigned int i = 0; i < 10; i++) {
    lst.push_back(i);
  }

  std::list<int>::iterator itr;  // declaration
  for (itr = lst.begin(); itr != lst.end(); ++it) {
   int i = *itr;          // dereference: access to element
   std::cout << "i = " << i << std::endl;
  }
  return (0);
}
```

Choosing the right container: depends on the job

- std::list
  - insertions or suppressions in the middle of the container
  - constant time insertions
  - access to *nth* element by iteration from the *1st* one
- std::vector
  - constant time access to *nth* element
  - insertions and suppressions can be costy
  - very efficient to traverse a container
- std::map
  - associative container
  - items sorted when inserted

Questions ?

Questions ?