CS213/293 Data Structure and Algorithms 2023

Lecture 2: Containers in C++

Instructor: Ashutosh Gupta

IITB India

Compile date: 2023-07-31

What are containers?

A collection of C++ objects

- `int a[10]; //array`
- `vector<int> b;`

Exercise 2.1

Why the use of the word 'containers'?

More container examples

- `array`
- `vector<T>`
- `set<T>`
- `map<T,T>`
- `unordered_set<T>`
- `unordered_map<T,T>`

In math, sets are unordered?

Set in C++ $\neq$ Mathematical set

Why do we need containers?

Collections are everywhere

- CPUs in a machine
- Incoming service requests
- Food items on a menu
- Shopping cart on a shopping website

Not all collections are the same

Example: using a container

Source: http://www.cplusplus.com/reference/set

```
#include <iostream>
#include <set>
int main () {
  std::set<int> s;
  for(int i=5; i>=1; i--)   // s: {50,40,30,20,10}
    s.insert(i*10);
  s.insert(20);     // no new element inserted
  s.erase(20);     // s: {50,40,30,10}

  if( s.contains(40) )
    std::cout << "s has 40!\n";

  for( int i : s )  // printing elements of a container
    std::cout << i << '\n';
  return 0;
}
```

Why do we need many kinds of containers?

- Expected properties and usage patterns define the container

  For example,

  - Unique elements in the collection
  - Arrival/pre-defined order among elements
  - Random access vs. sequential access
  - Only few additions(small collection) and many membership checks
  - Many additions (large collection) and a few sporadic deletes

Different containers are

efficient to use/run

in varied usage patterns

Choose a container

Exercise 2.2

Which container should we use for the following collections?

- *CPUs in a machine*
- *Incoming service requests*
- *Food items on a menu*
- *Shopping cart on a shopping website*

Some examples of containers

`set<T>`

- Unique element
- insert/erase/contains interface
- collection has implicit ordering among elements

`map<T,T>`

- Unique key-value pairs
- insert/erase interface
- collection has implicit ordering among keys
- Finding a key-value pair is not the same as accessing it
- Throws an exception if accessed using a non-existent key

Containers are abstract data types

The containers do not tell us the implementation details. They provide an interface with guarantees.

In computer science, we call the libraries abstract data types. The guarantees are called axioms of abstract data type.

Example 2.1

Axioms of abstract data type set.

- `std::set<int> s; s.contains(v) = false`
- `s.insert(v); s.contains(v) = true`
- `x = s.contains(u); s.insert(v); s.contains(`$\overset{u}{\underline{\texttt{v}}}$`)` $\overset{==}{\underline{=}}$ `x`, where $u != v$.
- `s.erase(v); s.contains(v) = false`
- `x = s.contains(u); s.erase(v); s.contains(`$\overset{u}{\underline{\texttt{v}}}$`)` $\overset{==}{\underline{=}}$ `x`, where $u != v$.

Example: map<T,T>

Source: http://www.cplusplus.com/reference/map

```
#include <iostream>
#include <string>
#include <map>
int main () {
  std::map<std::string,int> cart;
  // set some initial values:
  cart["soap"] = 2;
  cart["salt"] = 1;
  cart.insert( std::make_pair( "pen", 10 ) );
  cart.erase("salt");
  //access elements
  std::cout << "Soap: " << cart["soap"] << "\n";
  std::cout << "Hat: " << cart["hat"] << "\n";
  std::cout << "Hat: " << cart.at("hat") << "\n";
}
```

Exercise 2.3 *What will happen at the last two calls?*

Exceptions in Containers (abstract data types)

Containers must be used under certain conditions.

Example 2.2

Read operation `cart.at("shoe")` must not be called if the cart does not value for key `"shoe"` .

Since containers cannot return an appropriate value, they throw exceptions in the situations.

Callers must be ready to catch the exceptions and respond accordingly.

Please ask in tutorial session, if you need explanations related to exceptions.

STL: container libraries with unified interfaces

Since the containers are similar

`http://www.cplusplus.com/reference`

C++ in flux

Once C++ was set in stone. Now, modern languages have made a dent!

Three major revisions in history!!

- c++98
- c++11
- c++17
- c++20 (we will use this compiler!)

Topic 2.1

Array vs. Vector

Vector

- Variable length
- Primarily stack-like access
- Allows random access
- Difficult to search
- Overhead of memory management

Array

- Fixed length
- Random access
- Difficult to search
- Low overhead

Let us create a test to compare the performance

```
#include <iostream>
#include <vector>
#include "rdtsc.h"
using namespace std; // unclear!! STOP ME!
int local_vector(size_t N) {
  vector<int> bigarray; //initially empty vector
  // fill vector upto length N
  for(unsigned int k = 0; k<N; ++k)
    bigarray.push_back(k);
  // find the max value in the vector
  int max = 0;
  for(unsigned int k = 0; k<N; ++k) {
    if( bigarray[k] > max )
      max = bigarray[k];
  }
  return max;
} // 3N memory operations
```

Let us create a test to compare the performance (2)

```
// call local_vector M times
int test_local_vector( size_t M, size_t N ) {
  unsigned sum = 0;
  for(unsigned int j = 0; j < M; ++j ) {
    sum = sum + local_vector( N );
  }
  return sum;
}
//In total, 3MN memory operations
```

Let us create a test to compare the performance (3)

```
// assumes the 64-bit machine
int main() {
  ClockCounter t; // counts elapsed cycles
  size_t MN = 4*32*32*32*32*16;
  size_t N = 4;
  while( N <= MN  ) {
    t.start();
    test_local_vector( MN/N , N );
    double diff = t.stop();
    //print average time for 3 memory operations
    std::cout << "N = " << N << " : "<< (diff/MN);
    N = N*32;
  }
}
```

Exercise 2.4

Write the same test for arrays.

Topic 2.2

Problem

Exercise: What is the difference between at and ..[..] accesses?

Exercise 2.5

What is the difference between "at" and "..[..]" accesses in C++ maps?

Exercise: smart pointers

Exercise 2.6

C++ does not provide active memory management. However, smart pointers in C++ allow us the capability of a garbage collector. The smart pointer classes in C++ are

- `auto_ptr`
- `unique_ptr`
- `shared_ptr`
- `weak_ptr`

Write programs that illustrate the differences among the above smart pointers.

Exercise: named requirements

Exercise 2.7

Some of the containers have named requirements in their description. For example, "std::vector (for T other than bool) meets the requirements of Container, AllocatorAwareContainer (since C++11), SequenceContainer, ContiguousContainer (since C++17), and ReversibleContainer.".

What are these? Can you describe the meaning of these? How these conditions are checked?

End of Lecture 2