CS213/293 Data Structure and Algorithms 2024

Lecture 2: Containers in C++

Instructor: Ashutosh Gupta

IITB India

Compile date: 2024-08-04

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 provide details on the implementation. 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.

Commentary: Defining the axioms is not a simple matter. We need to answer the following questions.

Why do we need exactly these five axioms?
Are these sufficient?
Are any of them redundant, i.e., implied by others?
Do they contradict each other?

These kind of questions will be answered in CS228.

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(u) == x`, where $u! = v$.
- `s.erase(v); s.contains(v) == false`
- `x = s.contains(u); s.erase(v); s.contains(u) == 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";
}
```

Commentary: When we run `cart["hat"]`, C++ modifies the content of cart and maps "hat" to 0 (default value of int). Therefore, the run `cart.at("hat")` succeeds without exception. If we delete the second last statement containing `cart["hat"]` in the program, the last statement will throw an exception. It is a strange situation, where mere reading a data structure is modifying it and changing the behavior of the data structure.

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

Exceptions in Containers

If containers cannot return an appropriate value, they throw exceptions.

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

Example 2.2

Read operation `cart.at("shoe")` throws an exception if the cart does not value for key `"shoe"` .

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!)

Daily Quiz

```
#include <iostream>
#include <map>
int main() {
  std::map<int, std::string> responses = {  {0, "Zero value!"},
                         {-1, "Negative one!"}, {2, "Positive two!"} };
  int x;
  std::cin >> x;
  if (responses.find(x) != responses.end()) {
    std::cout << responses[x] << std::endl;
  } else {
    std::cout << "Default response." << std::endl;
  }
  return 0;
}
```

Topic 2.1

Exceptions

What to do if an unexpected event occurs

Example 2.3

- Divide by zero
- Open a non-existent file
- Network device is failed

Stop the program and throw an exception!

Exceptions: something unexpected happened!

you can't use auto in the catch block

```
#include <iostream>
using namespace std;

int foo(int x) {
  try
  {
    throw 20; // something has gone wrong!!
  }
  catch (int e) // type of e must match the type of thrown value!
  {
    cout << "An exception occurred. Exception Nr. " << e << '\n';
  }
  return 0;
}
```

Exceptions: catch matches the types!

```
int foo(int x) {
  try{
    if( x >  0 ){
      throw 20; // something has gone wrong!!
    }else{
      throw "C'est la vie!"; // Another thing has gone wrong!
    }
  }
  catch (int e){ // type of e is matched!
    cout << "An int exception occurred. " << e << '\n';
  }
  catch (string e){ // type of e is matched!
    cout << "A string exception occurred. " << e << '\n';
  }
  return 0;
}
```

Exceptions in the callee

```
int bar(){
  ...
  throw 20; // something has gone wrong!!
  ...
}

int foo(int x) {
  try{
    bar();
  }
  catch (int e){ // type of e is matched!
    cout << "An int exception occurred. " << e << '\n';
  }
}
```

Why write exceptions instead of handling the "unexpected" cases?

To avoid cumbersome code!

If no catch is written, the exception flows to the top, and the program fails.

Exceptions provide a succinct mechanism to handle all possible errors, with a few catches.

Topic 2.2

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 performances

```
#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 up to 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 2.3

Tutorial Problems

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

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

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

Exercise: const

Exercise 2.7

Why do the following three writes cause compilation errors in the C++20 compiler?

```
class Node {
public:
  Node() : value(0) { }
  const Node& foo( const Node* x) const {
    value = 3;          // Not allowed because of _______________
    x[0].value = 4;     // Not allowed because of _______________
    return x[0];
  }
  int value;
};
int main() {
  Node x[3], y;
  auto& z = y.foo(x);
  z.value = 5; // Not allowed because of _______________
}
```

It's just a way to initialise the value=0

1st node: says that the function return a const data type
2nd node: says that the parameters are const
3rd node: it says that the function is const

Topic 2.4

Problems

Exercise: named requirements

Exercise 2.8

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 are these conditions checked?

Exercise: auto in exception (2024 student suggestion!)

Exercise 2.9

Can we write auto within the catch parameter? No.

```
int foo(int x) {
  try{
    throw 20; // something has gone wrong!!
  }
  catch (auto e){ // type of e is matched!
    cout << "An int exception occurred. " << e << '\n';
  }
  return 0;
}
```

Topic 2.5

Extra slides: weak pointers

An illustrative example of weak pointer usage (continued)

```
#include <iostream>
#include <memory>
class Node {
public:
  Node(int value) : value(value) {std::cout << "Node " << value << " created." << std::endl; }
  // Functions to set/get the next node/weak ref to previous node/shared ref to previous node
  void setNext    ( std::shared_ptr<Node> next ) { nextNode = next;     }
  void setWeakPrev( std::shared_ptr<Node> next ) { prevWeakNode = next; }
  void setPrev    ( std::shared_ptr<Node> next ) { prevNode = next;     }
  std::shared_ptr<Node> getNext()     const      { return nextNode;             }
  std::shared_ptr<Node> getPrev()     const      { return prevNode;             }
  std::shared_ptr<Node> getWeakPrev() const      { return prevWeakNode.lock(); }
  // Function to display the value of the node
  void display() const { std::cout << "Node value: " << value << std::endl; }
private:
  int value;
  std::shared_ptr<Node> nextNode;
  std::shared_ptr<Node> prevNode;
  std::weak_ptr<Node>   prevWeakNode;
};
void print_list( std::weak_ptr<Node> current ) {
  for (int i = 0; i < 5; ++i) {
    auto current_ref = current.lock();
    if (current_ref) {
      current_ref->display();
      current = current_ref->getNext();
    } else {
      std::cout << "Next node is nullptr." << std::endl; break;
    }
  }
}
```

An example of weak pointer usage (2)

```
// Creating a doubly linked list via shared_ptr/weak_ptr
std::weak_ptr<Node> shared_test() {
  auto node1 = std::make_shared<Node>(1);
  auto node2 = std::make_shared<Node>(2);
  auto node3 = std::make_shared<Node>(3);
  // Create a circular reference
  node1->setNext(node2);
  node2->setNext(node3);
  node2->setPrev(node1); // shared pointer pointing to previous node is causing a reference cycle
  node3->setPrev(node2);
  return node1;
}
std::weak_ptr<Node> weak_test() {
  auto node1 = std::make_shared<Node>(1);
  auto node2 = std::make_shared<Node>(2);
  auto node3 = std::make_shared<Node>(3);
  node1->setNext(node2);
  node2->setNext(node3);
  node2->setWeakPrev(node1); // weak pointer pointing to previous node breaks cyclic reference counting
  node3->setWeakPrev(node2);
  return node1;
}
int main() {
  std::cout << "Testing shared pointer:" << std::endl;
  auto current = shared_test();
  print_list(current);
  std::cout << "Testing weak pointer:" << std::endl;
  current = weak_test();
  print_list(current);
  return 0;
}
```

End of Lecture 2