Copyright Warning

ViSLAB
High-Performance Visualization, Communication, Computing

Object Oriented Design

Week 11-2

STL : Algorithm

- STL containers provided convenient storages.
  - dynamically change sizes, etc.
  - generic programming
  - easy to change the containers

- Searching, sorting elements in the container
- other frequently used processes

find

- search an element from a container

```
#include <list>
#include <algorithm>
#include <iostream>

int main() {
   std::list<int> lst;
   std::list<int>::iterator it;

   lst.push_back(5);
   lst.push_back(4);
   lst.push_back(3);
   lst.push_back(2);
   lst.push_back(1);

   it = std::find(lst.begin(), lst.end(), 3);
   if (it != end())
      std::cout << "foud " << *it << std::endl;
   return 0;
}
```

- #include <algorithm>
- first 2 args : search region
- returns an iterator
- or the it == end
- can apply to vector, too
- multiple search?

```
#include <list>
#include <algorithm>
#include <iostream>

int main() {
   std::list<int> lst;
   std::list<int>::iterator it;

   lst.push_back(5);
   lst.push_back(4);
   lst.push_back(3);
   lst.push_back(2);
   lst.push_back(1);
   lst.push_back(2);

   for (it = lst.begin(); it != end(); ) {
      it = std::find(it, lst.end(), 2);
      if (it != lst.end()) {
         std::cout << "found " << *it << std::endl;
         ++it;
      }
   }
   return 0;
}
```

- search until fails
- lst.end() is not covered.
- lst.end() is empty
- [lst.begin(), lst.end())

vector<int>....

- STL containers are templates..
- yes, you can do vector<int>...., but
- you want to store your own class:
  - vector<MyClass>

Custom Class

```
#include <iostream>

class MyData {
    int data1;
    char* data2;
    friend std::ostream& operator<<(std::ostream& os, MyData& a) {
        return os << a.data1 << ", " << a.data2;
    }
    void init_data2(const char* d2) {
        if (d2) {
            this->data2 = new char[strlen(d2)+1];
            strcpy(this->data2, d2);
        }
    }
public:
    MyData(int d1, char* d2) : data1(d1), data2(0) {
        init_data2(d2);
    }
    MyData(const MyData& src) : data1(src.data1), data2(0) {
        init_data2(src.data2);
    }
};
```

Custom Class

```
int main() {
std::list<MyData> lst;
std::list<MyData>::iterator it;

lst.push_back(MyData(3, "data3"));
lst.push_back(MyData(5, "data5"));
lst.push_back(MyData(2, "data2"));
lst.push_back(MyData(1, "data1"));
lst.push_back(MyData(6, "data6"));
lst.push_back(MyData(4, "data4"));

   for (it = lst.begin(); it != lst.end(); ) {
      it = std::find(it, lst.end(), MyData(1, "test1"));
      if (it != lst.end()) {
         std::cout << "found " << *it << std::endl;
         ++i;
      }
   }
   return 0;
}
```

- MyData obj-> local
- std::list : data -> heap
- MyData copy const.

Custom Class

```
int main() {
std::list<MyData> lst;
std::list<MyData>::iterator it;

lst.push_back(MyData(3, "data3"));
lst.push_back(MyData(5, "data5"));
lst.push_back(MyData(2, "data2"));
lst.push_back(MyData(1, "data1"));
lst.push_back(MyData(2, "data6"));
lst.push_back(MyData(4, "data4"));

   for (it = lst.begin(); it != lst.end(); ) {
      it = std::find(it, lst.end(), MyData(1, "test1"));
      if (it != lst.end()) {
         std::cout << "found " << *it << std::endl;
         ++i;
      }
   }
   return 0;
}
```

- compile error?!?
- operator==

Custom Class

```
iterator find(iterator begin, iterator end, T value) {
iterator it;
for (it = begin; it != end; ++it)
   if (*it == value)
      return it;
return end;
}
```

- inside “find”
- == to compare values (int, double are ok, but..)
- you need to overload “operator==”

Custom Class

```
class MyData {
int data1;
char* data2;
friend std::ostream& operator<<(std::ostream& os, MyData& a) {
   return os << a.data1 << ", " << a.data2;
}
friend bool operator==(const MyData &left, const MyData &right) {
   return left.data1 == right.data1;
}
void init_data2(const char* d2) {
   if (d2) {
      this->data2 = new char[strlen(d2)+1];
      strcpy(this->data2, d2);
   }
}
public:
   MyData(int d1, char* d2) : data1(d1), data2(0) {
      init_data2(d2);
   }
   MyData(const MyData& src) : data1(src.data1), data2(0) {
      init_data2(src.data2);
   }
};
```

overloading operator

- find ....."=="
- sort ....."<",">"

- Cannot find "operatorXX"
  - then overload it.
  - use "const" for the arg

Pointer and Iterator

- Pointer works just like Iterator
- Arithmetic operation
- can be used to access a container

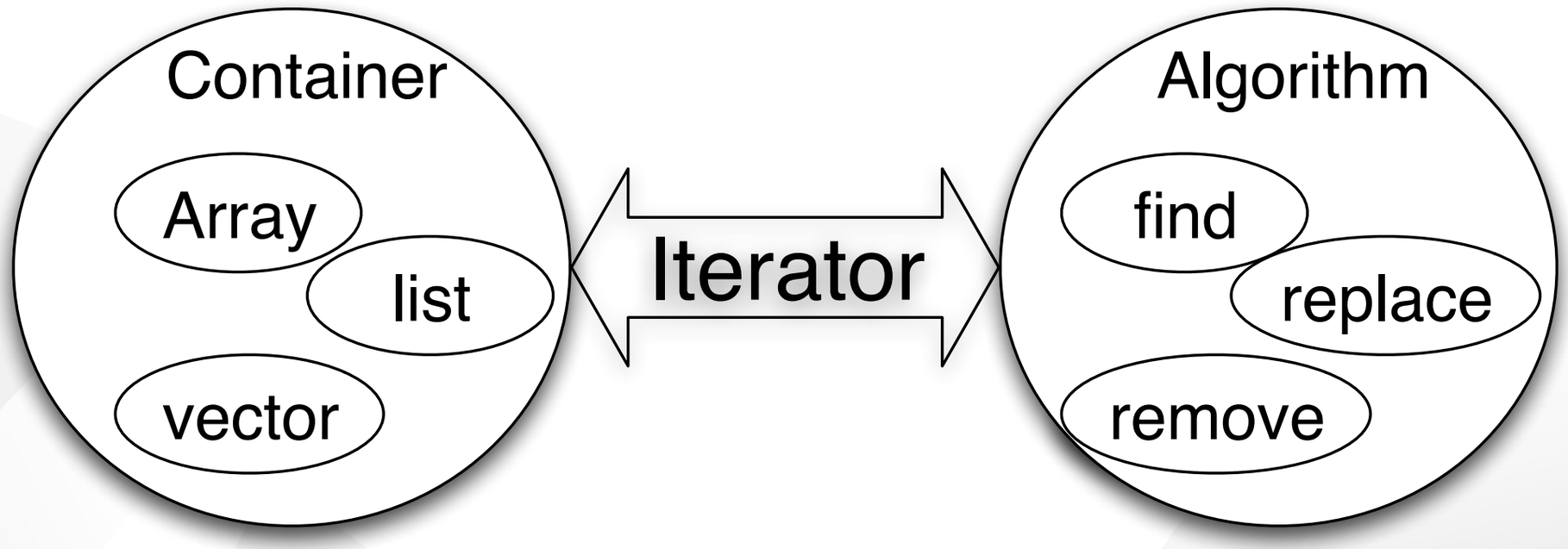
std::copy

- you can copy using
  - operator=
  - copy constructor, but..
- std::copy also copy elements one by one.

```
int main() {
std::vector<int> v;
std::list<int> l;

v.push_back(3);
v.push_back(5);
v.push_back(2);
v.push_back(1);

   l.push_back(0);
   l.push_back(0);
   l.push_back(0);
   l.push_back(0);

   std::copy(v.begin(), v.end(), l.begin());
   std::list<int>::iterator it;
   for (it = l.begin(); it != l.end(); ++it)
      std::cout << *it < std::endl;
   return 0;
}
```

```
int main() {
int a[] = {3, 5, 2, 1};
std::list<int> l;

l.push_back(0);
l.push_back(0);
l.push_back(0);
l.push_back(0);

   std::copy(a, a + 5, l.begin());
   std::list<int>::iterator it;
   for (it = l.begin(); it != l.end(); ++it)
      std::cout << *it < std::endl;
   return 0;
}
```

- Array + pointer works with std::copy
- why iterator?



stream iterator

- 3rd arg ...iterator
- indirectly points to the destination container
- needs to be move to the next element

- use cout in C++ rather than printf (tutorial)
- stream object .... container

stream iterator

- ostream_iterator (output)
- istream_iterator (input)

stream iterator

```
int main() {
int a[] = {3, 5, 2, 1};
std::list<int> l;

l.push_back(0);
l.push_back(0);
l.push_back(0);
l.push_back(0);

std::copy(a, a + 5, l.begin());
std::copy(l.begin(), l.end(), std::ostream_iterator<int>(std::cout, " "));
return 0;
}
```

- More in tute.

string

- basic_string is a container

```
typedef basic_string<char> string;
typedef basic_string<wchar_t> wstring;
```

- string is also a container
- you can use std::find, std::copy for string