# Copyright Warning

ViSLAB
High-Performance Visualization, Communication, Computing

# Object Oriented Design

## Week 10-2

# STL (Standard Template Library)

- Template
  - same process for different data types
- good for "writing less code!"

- list structure,
- alg. for 2-tree search
- etc.

# STL

- Container
- Algorithm
- Iterator
- Functor/Function Object

# STL : Container

- vector
- deque
- list
- set/multiset
- map/multimap
- queue/stack

# vector

- can store many elements
- varaible length
  - internally “malloc”, “realloc”
  - getting close to full....resize/copy

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
   vector<int> v;

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

   for (int i = 0; i < v.size(); i++)
      cout << v[i] << endl;
   return 0;
}
```

- <int>
- push_back, etc.
- operator[].....no idx check
  - but fast
- at .... safer but bit slow

```
#include <iostream>
#include <vector>
#include <stdexept>
using namespace std;
int main() {
   try {
      vector<int> v;
      v.at(34) = 20;
   } catch (std::out_of_range &ex) {
      cout << "out fo range" << endl;
   }
   return 0;
}
```

- <int>
- push_back, etc.
- operator[].....no idx check
  - but fast
- at .... safer but bit slow

# vector : iterator and delete

```
#include <vector>
using namespace std;
int main() {
   vector<int> v;
   vector<int>::iterator it;
   ...
   return 0;
}
```

- “erase” to delete
- “iterator” to specify the location
- iterator : internal class
  - needs <int>

# vector : iterator and delete

```
#include <vector>
using namespace std;
int main() {
   vector<int> v;
   vector<int>::iterator it;
   for (it = v.begin(); it != v.end(); ++it) {
      ...
   }
   ...
   return 0;
}
```

- v.begin() ... first iterator
- v.end() ... last iterator

- it != v.end() rather than
  - it < v.end()

# vector : iterator and delete

```
#include <vector>
using namespace std;
int main() {
   vector<int> v;
   v.push_back(5);
   v.push_back(4);
   v.push_back(3);

   vector<int>::iterator it;
   for (it = v.begin(); it != v.end(); ++it) {
      cout << *it << endl;
   }
   return 0;
}
```

- use * to access the element
- not a pointer!
- operator* is defined

- -> (member access) can be used for
  - class or struct

# iterator

```
#include <vector>
using namespace std;
int main() {
   vector<int> v;
   v.push_back(5);
   v.push_back(4);
   v.push_back(3);

   vector<int>::iterator it = v.begin() + 1;
   v.erace(it);
   for (it = v.begin(); it != v.end(); ++it) {
      cout << *it << endl;
   }
   return 0;
}
```

- arithmetic operations

# iterator

```
#include <vector>
using namespace std;
int main() {
   vector<int> v;
   v.push_back(5);
   v.push_back(4);
   v.push_back(3);
   vector<int>::iterator it;
   for (it = v.begin(); it != v.end(); ++it) {
      if (*it == 4)
          v.erase(it);
      cout << *it << endl;
   }
   return 0;
}
```

- be careful when you “erase”
- the corresponding iterator is no longer valid.

# iterator

```
#include <vector>
using namespace std;
int main() {
   vector<int> v;
   v.push_back(5);
   v.push_back(4);
   v.push_back(3);

   vector<int>::iterator it;
   for (it = v.begin(); it != v.end(); ) {
      if (*it == 4)
          v.erase(it);
      else
          ++it;
      cout << *it << endl;
   }
   return 0;
}
```

- be careful when you “erase”
- the corresponding iterator is no longer valid.

# list

- vector is
  - continuous
    - slow ....frequent add/delete
    - re-arrange elements
- list is
  - double linked list
  - efficient in add/delete operations

# list

```
#include <iostream>
#include <list>
using namespace std;
int main() {
   list<int> l;
   l.push_back(5);
   l.push_back(4);
   l.push_back(3);
   l.push_back(2);

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

- use of list is “almost” the same as vector
- no l[i]
- no l.begin() + 2
- just it++, it--