C++ Standard Library Introduction

Outline Standard library

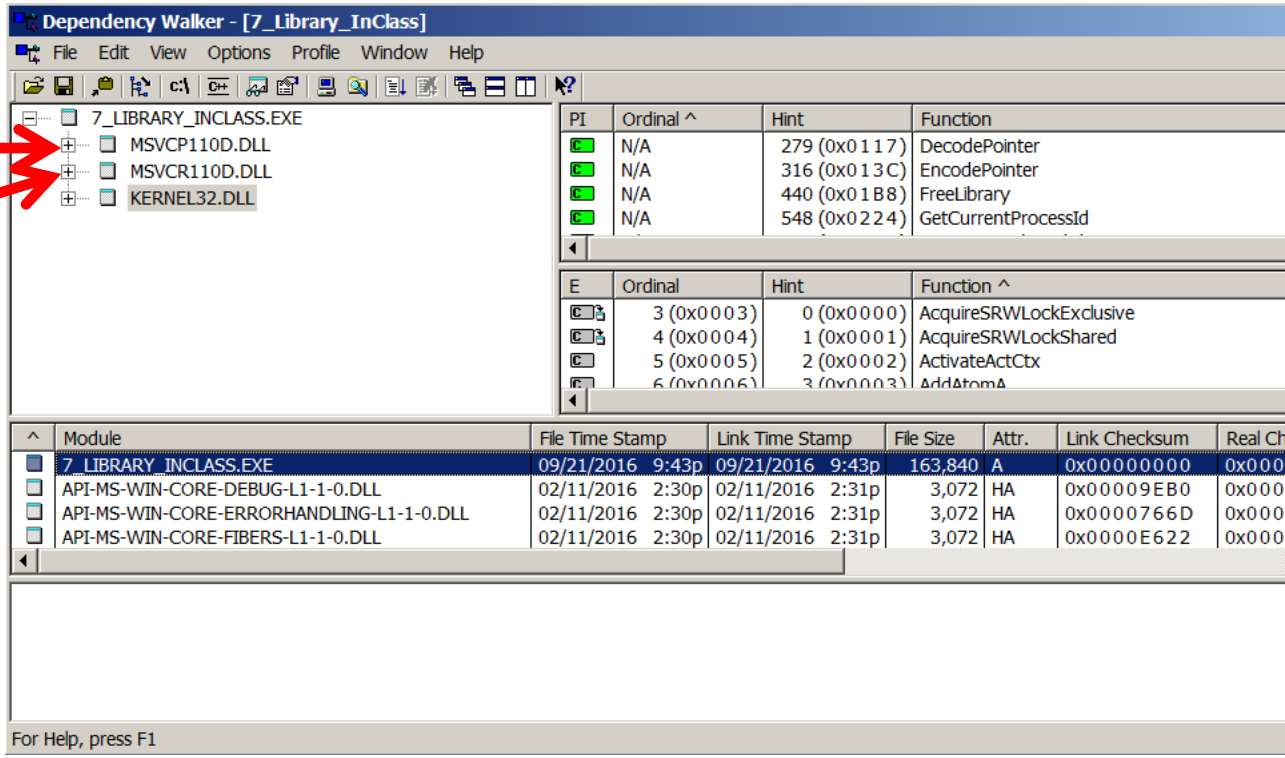
- Where is it?
- Why use it?
- What's in it?
- Choosing data structures
- Iterators

Where is it?

- Dependency Walker http://www.dependencywalker.com/

C++ library

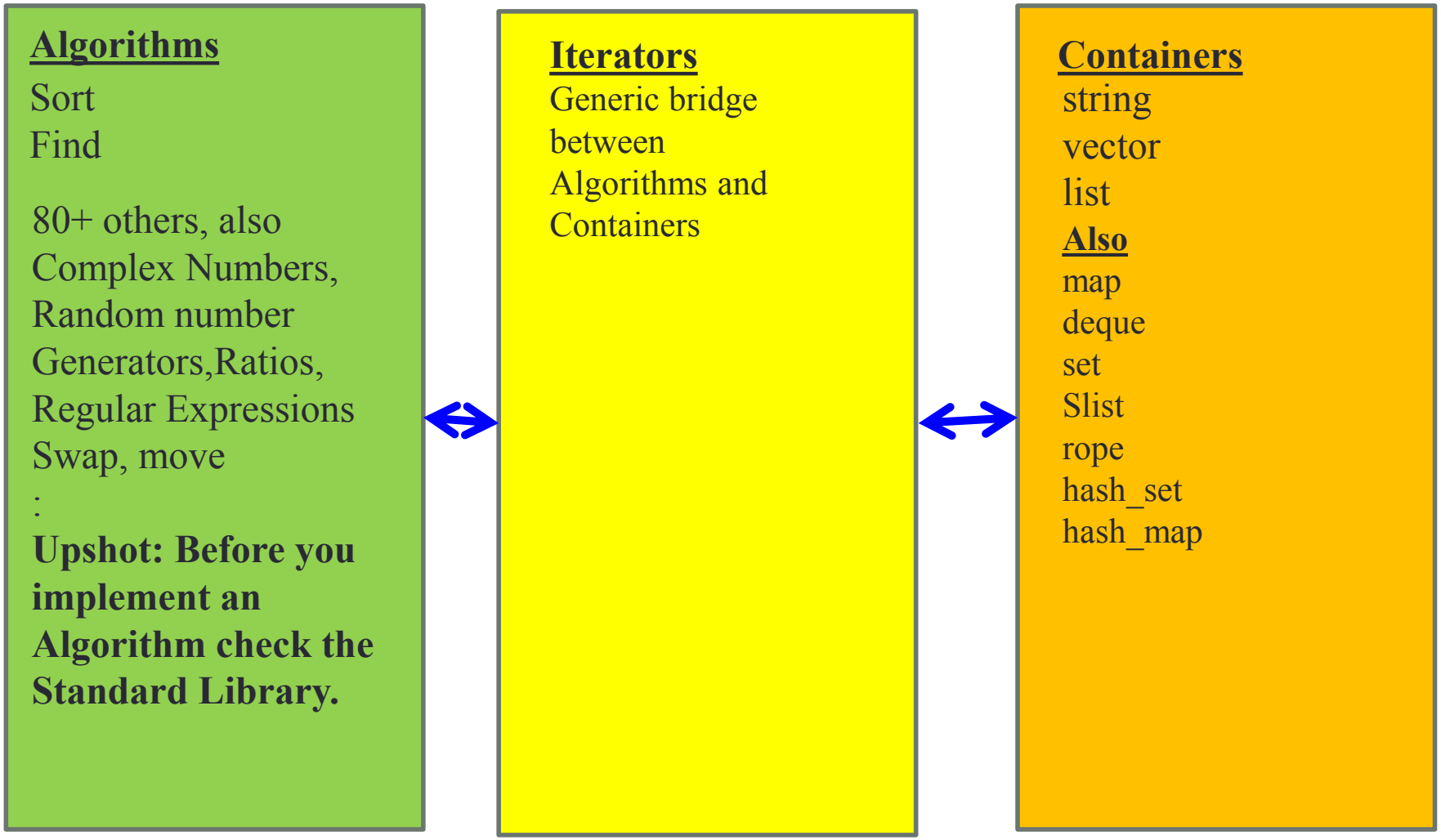
C library



Why use Standard Library

- Code Reuse (never reinvent the wheel)
- Fast efficient
- WELL DEBUGGED
- Terse Readable code
- Guaranteed available with C++ compiler
- Standardized

What is in Standard library



Containers

1. vector, string, deque…
2. list
3. set, map, hash_set, hash_map …

Containers

- Written by Experts
- Designed for specific situations
- Guaranteed performance (remember Big O?)
- ALWAYS Choose container based on your particular application.
- How?...

Containers- Simplified Rules

1. Need random access? – vector
2. Need to insert/delete from middle? - list
3. Lookup speed critical – hash_map, sorted vector …
4. Need to insert/delete from begginning/end? deque

5. Are you lazy – (sigh…) just choose vector

See http://stackoverflow.com/questions/10699265/how-can-i-efficiently-select-a-standard-library-container-in-c11

Also Effective STL, Scott Meyers

Example- student grades

- Problem: Bunch of students, with name, midterm and final grades. Want to calculate their class grade and then sift out people who failed.

- Datastructure?

```
const double UNINITIALIZED = -1.0;
struct studentData{
	std::string name;
	double midterm,final;
	double classgrade;
	void clear(){name.clear();midterm=final=classgrade=UNINITIALIZED;}
};
```

- Top down design

Iterators

- Sequential NOT random access
- Used by containers to move between and examine each element
- Each container defines its own iterator
- Example vector and list iterators

```
//iterator for list
std::list<studentData>::iterator itr1;

//iterator for vector
std::vector<studentData>::iterator itr;
```

Iterators - Using

The [] way, does not work with most containers

```
for ( int i = 0; i != myData.size()-1; ++i ){
    myData[i].classgrade = 0.4 * myData[i].midterm  + 0.6 * m
}
```

The iterator way, works with all containers

```
std::vector<studentData>::iterator itr;
for ( itr = myData.begin(); itr != myData.end(); ++itr ){
    (*itr).classgrade = 0.4 * (*itr).midterm    + 0.6 * (*itr).fin
}
```

Pointers again

Pointers (will see again in memory allocation)

- Represents a memory address
- Refers to the location where an object resides in the computer's memory
- Initialize

```
//initialize to 0 (0 or NULL)
//unless setting it equal to an address
int     *ip     = NULL;
double  *dp     = 0;
char    *chp    = 0;
```

- Size of all pointers is the same (large enough to hold memory address)
- Setting pointer address

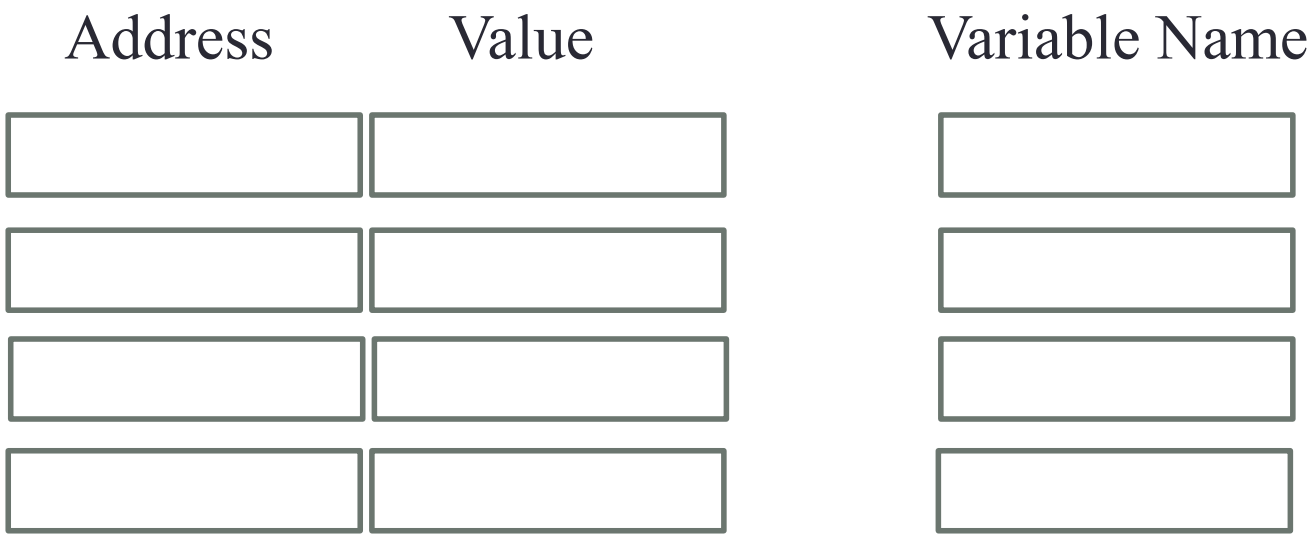
```
ip      = &myint;
```

- Dereference it to get the stored value

```
int NEWint          = *ip;
```

Pointers – Reminder

```
int myint       = 3;
int *ip         = NULL;
ip              = &myint;
int NEWmyint    = *ip;
int NEWip       = ip;
```



Pointers and References

- Pointer can initially point to one object and later be made to point to another object
- References, once initialized, must always point to same thing
- Thus when declared references must be initialized.

```
//references
int& myintref    = myint;     //must initialize at declaration
myintref         = &NEWint;   //
```

- References cannot be null (0), pointers can and often are null.

```
const int MP_WAS_NULL = -1;
int myFunc(int& myint, int* mp){
    //dont have to check myint for null
    //MUST check pointer mp
    if (!mp)
        return MP_WAS_NULL;
```

Iterators - Using

The [] way, does not work with most containers

```
for ( int i = 0; i != myData.size()-1; ++i ){
    myData[i].classgrade = 0.4 * myData[i].midterm  + 0.6 * m
}
```

The iterator way, does work with most containers

```
std::vector<studentData>::iterator itr;
for ( itr = myData.begin(); itr != myData.end(); ++itr ){
    (*itr).classgrade = 0.4 * (*itr).midterm    + 0.6 * (*itr).fin
}
```

This is a pointer that's dereferenced to view the underlying object. In this case a studentData Struct. Incidently (*iter).classgrade Is the same as iter->classgrade

Iterators - Using

```
void extractFailingStudents(vector<studentData> &allstudentData, vector<studentData> &failstudentData){
    const double FAILGRADE = 60.0;

    //iterate over allstudentData
    //using iterators
    std::vector<studentData>::iterator itr = allstudentData.begin();
    while (itr != allstudentData.end()){
        if ((*itr).classgrade<FAILGRADE){
            failstudentData.push_back(*itr);
            itr = allstudentData.erase(itr);    //erase returns updated iter pointing to next element
        }
        else
            ++itr;
    }
}
```

failstudentData

allstudentData

Revisit container selection

extractFailingStudents() deleted from middle of vector allstudentData, so What is a good datastructure?

1. Need random access? – vector
2. Need to insert/delete from middle? - list
3. Lookup speed critical – hash_map, sorted vector …
4. Need to insert/delete from beginning/end? Deque

From Rule 2, choose List

Revise part of 4_vector_studentGrades

```
 void extractFailingStudents(vector<studentData> &allstudentData, list<studentData> &failstudentData
//void extractFailingStudents(vector<studentData> &allstudentData, vector<studentData> &failstudentDa
    const double FAILGRADE = 60.0;

   //iterate over allstudentData
   std::list<studentData>::iterator    itr = allstudentData.begin();   //list
 //std::vector<studentData>::iterator  itr = allstudentData.begin();   //vector

    while (itr != allstudentData.end()){
        if ((*itr).classgrade<FAILGRADE){
            failstudentData.push_back(*itr);
            itr = allstudentData.erase(itr);    //erase returns updated iter pointing to next element
        }
        else
            ++itr;
    }
```

What difference does this really make?

File Size	List	Vector
735	0.1	0.1
7350	0.8	6.7
73500	8.8	597.1

Can you swap one container for another?

- Usually - No
- Only sequence containers support push_front or push_back (array, vector, deque,list,forward_list)
- Only associative containers support count and lower_bound (set, multiset, map, multimap)
- Contiguous-memory containers offer random-access iterators (vector, string, deque)
- node-based containers offer bidirectional iterators (list, set, map, hash_set, hash_map …)

Summary

- Don’t Reinvent the wheel. The standard library is your first stop when designing a project.
  - Choose data structure (container) based on which one performs best for your needs
  - Look in Algorithms before you write anything
- Iterators are a standardized way to move through containers, element by element