

# C++ Standard Library Introduction

# Administrative

- Program 3 assigned tonight
- Due Wed 10/8 at midnight
- Examples in scholar! Please review!
- Week 5 Readings (Vectors)

# Standard library

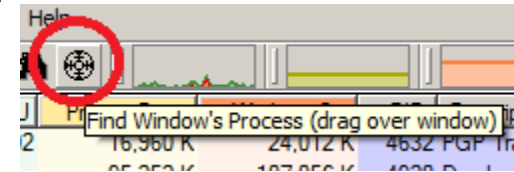
- How to use it
- Where is it
- Why use it
- What's in it
- Choosing data structures
- Vectors
- Sorting

# How to use it

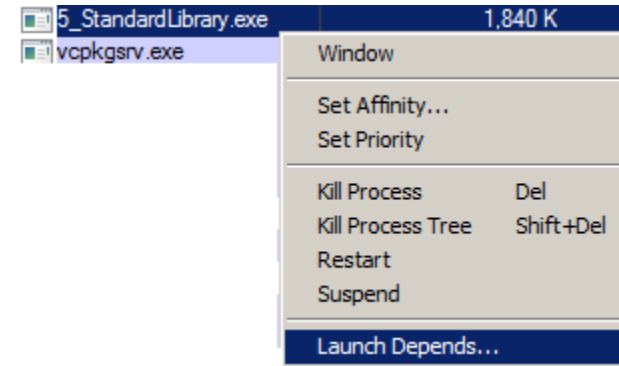
- Just include appropriate headers and use std library constructs
- You already have used
  - `std::cin`
  - `std::cout`
  - `std::string`
- And have had to include
  - `#include <string>`
  - `#include <iostream>`
- Automatically linked for you by compiler

# Where is it? (advanced technique)

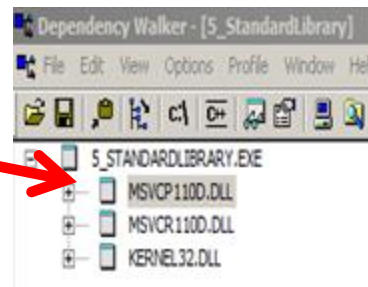
- Create project, use `std::string` and `std::cout`
- Find with “Sysinternals Process Explorer”



- Open process in Dependency Walker  
<http://www.dependencywalker.com/>



- And here it is for Visual C++



- Other compilers have other implementations

## Why use it?

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

# What is in it

## Algorithms

Sort

Find

80+ others, also

Complex Numbers,

Random number

Generators, Ratios,

Regular Expressions

Swap, move

:

**Upshot: Before you  
implement an  
Algorithm check the  
Standard Library.**

## Iterators

provide a means for  
accessing data  
stored in container

## Containers

string

vector

list

### Also

map

deque

set

Slist

rope

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- Which to Use

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
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

## Pointers (again)

- 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;
```

Address	Value	Variable Name
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>

# Iterators

- 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

# 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. Incidentally (\*itr).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 itr pointing to next element
        }
        else
            ++itr;
    }
}
```

**failstudentData**


**allstudentData**


# Revisit Grade Calculation program

## 4\_vector\_studentGrades

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> &failstudentData
    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 itr pointing to next element
        }
        else
            ++itr;
    }

```

## What difference does this really make?

<u>File Size</u>	<u>List</u>	<u>Vector</u>
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