

C++ Sans CS106B

Demystifying `genlib.h`

Simplified genlib.h Contents

```
#ifndef _genlib_h
#define _genlib_h

#include <string>
using namespace std;

void Error(string str);

#endif
```

Simplified genlib.h Contents

```
#ifndef _genlib_h
#define _genlib_h

#include <string>
using namespace std;

void Error(string str);

#endif
```

Simplified genlib.h Contents

```
#ifndef _genlib_h
#define _genlib_h

#include <string>
using namespace std;

void Error(string str);

#endif
```

Simplified genlib.h Contents

```
#ifndef _genlib_h
#define _genlib_h

#include <string>
using namespace std;

void Error(string str);

#endif
```

```
#include <iostream>
```

```
int main()
```

```
{
```

```
    cout << "C++ FTW!" << endl;
```

```
    return 0;
```

```
}
```

The Standard Namespace (namespace std)

cout

endl

cin

string

```
#include <iostream>
```

```
int main()
```

```
{
```

```
    cout << "C++ FTW!" << endl;
```

```
    return 0;
```

```
}
```

The Standard Namespace (namespace std)

cout

endl

cin

string

```
#include <iostream>
```

```
int main()  
{
```

```
    cout << "C++ FTW!" << endl;  
    return 0;
```

```
}
```

**Impenetrable
barrier!**



The Standard Namespace (namespace std)

cout

endl

cin

string

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

```
int main()  
{  
    cout << "C++ FTW!" << endl;  
    return 0;  
}
```

↑
Impenetrable
barrier!

Summary of `genlib.h`

- Includes the standard string type through `#include <string>`
- Prototypes `Error`.
- Imports standard names with `using namespace std;`

The Standard Template Library (STL)

STL Hierarchy

STL Hierarchy

CONTAINERS

STL Hierarchy

ITERATORS

CONTAINERS

STL Hierarchy

ALGORITHMS

ITERATORS

CONTAINERS

STL Hierarchy

FUNCTORS

ALGORITHMS

ITERATORS

ALLOCATORS

CONTAINERS

ADAPTERS

STL Containers

- Standard C++ version of the CS106 ADTs.
- Nothing fundamentally new – skills from CS106 transferable with some name changes.
- No `Grid`, sorry...

CS106 `vector` to **STL** `vector`

CS106 Vector to STL vector

```
#include "vector.h"
#include <iostream>
using namespace std;

const int NUM_INTS = 10;
int main()
{
    Vector<int> v;
    for(int i = 0; i < NUM_INTS; i++)
        v.add(i);
    for(int i = 0; i < v.size(); i++)
        cout << v[i] << endl;
}
```

CS106 Vector to STL vector

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

const int NUM_INTS = 10;
int main()
{
    Vector<int> v;
    for(int i = 0; i < NUM_INTS; i++)
        v.add(i);
    for(int i = 0; i < v.size(); i++)
        cout << v[i] << endl;
}
```

CS106 Vector to STL vector

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

const int NUM_INTS = 10;
int main()
{
    vector<int> v;
    for(int i = 0; i < NUM_INTS; i++)
        v.add(i);
    for(int i = 0; i < v.size(); i++)
        cout << v[i] << endl;
}
```

CS106 Vector to STL vector

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

const int NUM_INTS = 10;
int main()
{
    vector<int> v;
    for(int i = 0; i < NUM_INTS; i++)
        v.push_back(i);
    for(int i = 0; i < v.size(); i++)
        cout << v[i] << endl;
}
```

foreach is a CS106B-specific invention.

That means that there is no **foreach** in the STL.

Instead, the STL uses **iterators**.

STL Iterators

- Generalization of a pointer.
- Used to iterate over containers.
- More verbose than **foreach**, but more powerful.

Basic Iterator Patterns

Basic Iterator Patterns

```
Vector<int> myVector;  
foreach(int x in myVector)  
    cout << x << endl;
```

Basic Iterator Patterns

```
Vector<int> myVector;  
foreach(int x in myVector)  
    cout << x << endl;
```

```
vector<int> stlVec;  
for(vector<int>::iterator itr = stlVec.begin();  
    itr != stlVec.end(); ++itr)  
    cout << *itr << endl;
```

Basic Iterator Patterns

```
Vector<int> myVector;  
foreach(int x in myVector)  
    cout << x << endl;
```

```
vector<int> stlVec;  
for(vector<int>::iterator itr = stlVec.begin();  
    itr != stlVec.end(); ++itr)  
    cout << *itr << endl;
```

Basic Iterator Patterns

```
Vector<int> myVector;  
foreach(int x in myVector)  
    cout << x << endl;
```

```
vector<int> stlVec;  
for(vector<int>::iterator itr = stlVec.begin();  
    itr != stlVec.end(); ++itr)  
    cout << *itr << endl;
```

Basic Iterator Patterns

```
Vector<int> myVector;  
foreach(int x in myVector)  
    cout << x << endl;
```

```
vector<int> stlVec;  
for(vector<int>::iterator itr = stlVec.begin();  
    itr != stlVec.end(); ++itr)  
    cout << *itr << endl;
```

Basic Iterator Patterns

```
Vector<int> myVector;  
foreach(int x in myVector)  
    cout << x << endl;
```

```
vector<int> stlVec;  
for(vector<int>::iterator itr = stlVec.begin();  
    itr != stlVec.end(); ++itr)  
    cout << *itr << endl;
```

Basic Iterator Patterns

```
Vector<int> myVector;  
foreach(int x in myVector)  
    cout << x << endl;
```

```
vector<int> stlVec;  
for(vector<int>::iterator itr = stlVec.begin();  
    itr != stlVec.end(); ++itr)  
    cout << *itr << endl;
```

Basic Iterator Patterns

```
Vector<int> myVector;  
foreach(int x in myVector)  
    cout << x << endl;
```

```
vector<int> stlVec;  
for(vector<int>::iterator itr = stlVec.begin();  
    itr != stlVec.end(); ++itr)  
    cout << *itr << endl;
```

```
for(vector<int>::iterator itr = stlVec.begin();  
    itr != stlVec.end(); ++itr)  
    *itr = 137;
```

Basic Iterator Patterns

```
Vector<int> myVector;  
foreach(int x in myVector)  
    cout << x << endl;
```

```
vector<int> stlVec;  
for(vector<int>::iterator itr = stlVec.begin();  
    itr != stlVec.end(); ++itr)  
    cout << *itr << endl;
```

```
for(vector<int>::iterator itr = stlVec.begin();  
    itr != stlVec.end(); ++itr)  
    *itr = 137;
```

```
set<int> stlSet;  
for(set<int>::iterator it = stlSet.lower_bound(3);  
    it != stlVec.upper_bound(137); ++it)  
    cout << *it << endl;
```

STL Algorithms

- No parallel in the CS106 libraries.
- 75 generic algorithms to apply to ranges of data.
- Usually operate on iterator ranges.

STL Algorithms Example

```
vector<int> v;  
const int NUM_INTS = 200;  
  
v.resize(NUM_INTS);  
generate(v.begin(), v.end(), rand);  
sort(v.begin(), v.end());  
  
if(binary_search(v.begin(), v.end(), 137))  
    cout << "Found 137!" << endl;  
else  
    cout << "Awww..." << endl;
```

STL Algorithms Example

```
vector<int> v;  
const int NUM_INTS = 200;  
  
v.resize(NUM_INTS);  
generate(v.begin(), v.end(), rand);  
sort(v.begin(), v.end());  
  
if(binary_search(v.begin(), v.end(), 137))  
    cout << "Found 137!" << endl;  
else  
    cout << "Awww..." << endl;
```

More STL Algorithms

```
bool IsPalindrome(string str)
```

More STL Algorithms

```
bool IsPalindrome(string str)
{
    return equal(str.begin(), str.end(),
                str.rbegin());
}
```

Even More STL Algorithms

```
string ConvertToUpperCase(string str)
```

Even More STL Algorithms

```
string ConvertToUpperCase(string str)
{
    transform(str.begin(), str.end(),
             str.begin(), ::toupper);
    return str;
}
```

More STL Algorithms Than That

```
vector<vector<int> >  
AllPermutations (vector<int> input)
```

More STL Algorithms Than That

```
vector<vector<int> >  
AllPermutations(vector<int> input)  
{  
    vector<vector<int> > result;  
    sort(input) ;  
  
    do  
        result.push_back(input) ;  
    while (next_permutation(input.begin() ,  
                            input.end())) ;  
  
    return result ;  
}
```

Summary of the STL

Containers **store data**.

Iterators define ranges.

Algorithms let you sleep **earlier**.