COMP2012H

STL: More Algorithms
Example: STL Algorithm – `count_if()`

- Here we count the number of elements that are larger than 10:

```cpp
#include <vector>
#include <algorithm>
#include "greater_than.hpp"
#include "init.cpp"
using namespace std;

int main()
{
    vector<int> x;
    my_initialization(x);
    int num = count_if(x.begin(), x.end(), Greater_Than(10));
}
```
Example: STL Algorithm – Using `for_each()` for Summation

```cpp
#include <iostream>
#include <list>
#include <algorithm>
#include "init.cpp"
using namespace std;

class Sum {
    private:
        int sum;
    public:
        Sum() : sum(0) { }
        void operator()(int value) { sum += value; }
        int result() const { return sum; }
    };

int main()
{
    list<int> x; my_initialization(x);
    Sum s = for_each( x.begin(), x.end(), Sum() );
    cout << "Sum = " << s.result() << endl;
}
```
Example: STL Algorithm –
Using `for_each()` for Summation (cont)

• In the code

```c++
for_each( x.begin(), x.end(), Sum() );
```

the `Sum()` is a constructor that creates an unnamed `local` object of class `Sum`.

• Pretend this unnamed local function object is called `c`.
  - Then the `for_each` runs `c(x[j])` for each `j`.
  - So at the end, `c.sum` contains the value of the sum of all of the items in `x`.

• But as soon as the statement finishes executing, the local object `c` goes out of scope and is destructed!
  So how do we avoid losing the sum we just computed?
Example: STL Algorithm –
Using `for_each()` for Summation (cont)

- Recall STL’s template definition of `for_each`:

  ```cpp
  template<class IteratorT, class FunctionT>
  FunctionT for_each(IteratorT first, IteratorT last, FunctionT g)
  {
    for ( ; first != last; ++first) {
      g(*first);
    }
    return g; // return-by-value: returns a copy of the function object
  }
  ```

- So the line

  ```cpp
  Sum s = for_each( x.begin(), x.end(), Sum() );
  ```

  calls a copy constructor which makes `s` into a memberwise copy of `c`. Thus `s.sum` becomes the value of the sum of all of the items in `x`. 
Example: STL Algorithm –
Using \texttt{for\_each()} for Summation (cont)

• To confuse matters, beware that the code

```cpp
int main()
{
    list<int> x; my_initialization(x);
    Sum s;
    for\_each(x.begin(), x.end(), s);
    cout << "Sum = " << s.result() << endl;
}
```

would return 0! The reason that it doesn't return the expected value is that \texttt{for\_each} calls its arguments \textbf{by value}. This means that the total sum is stored in a \textbf{local copy} of \texttt{s} and not in \texttt{s} itself. Therefore \texttt{s} itself never changes from its initially constructed value so \texttt{s.sum}=0.
STL Algorithms – `transform()`

```cpp
template<class Iterator1T, class Iterator2T, class FunctionT>
Iterator2T transform(Iterator1T first, Iterator1T last, Iterator2T result, FunctionT g)
{
    for ( ; first != last; ++first, ++result) {
        *result = g(*first);
    }
    return result;
}
```

- `transform` will apply function `g()` to all of the items in the sequence between `first` and `last`. The resulting sequence is written to the location ‘pointed’ to by `result`. 
Example: STL Algorithm – 
Using `transform()` to Add

```cpp
// File: "Add.hpp"
#include <iostream>
#include <list>
#include <vector>
#include <algorithm>
using namespace std;

class Add {
private:
    int data;
public:
    Add(int i) : data(i) { }
    int operator()(int value) { return value + data; }
};

void print (int val) { cout << val << " "; }
```
Example: STL Algorithm – Using `transform()` to Add (cont)

```cpp
#include "Add.hpp"

int main()
{
    list<int> x;
    for(int i = 1; i < 10; ++i) {
        x.push_back(i);
    }
    vector<int> y(x.size());

    transform( x.begin(), x.end(), y.begin(), Add(10) );
    for_each( y.begin(), y.end(), print );
    cout << endl;
}
```
Many Other Algorithms in the STL

- STL contains many other algorithms, including for example:
  - min_element and max_element
  - equal
  - generate (to replace elements by applying a function object)
  - remove, remove_if (to remove elements)
  - reverse, rotate (to rearrange sequence)
  - random_shuffle
  - binary_search
  - sort (using a function object to compare two elements)
  - merge, unique
  - set_union, set_intersection, set_difference

- Good documentation for SGI's extended STL implementation can be found at [http://www.sgi.com/Technology/STL/doc_introduction.html](http://www.sgi.com/Technology/STL/doc_introduction.html) but note that this is different from the Standard C++ Library, and is no longer maintained!

- See the textbook for more details.