Vector Constructors

The `vector` template provides several constructors:

```
vector<T> V;           //empty vector
vector<T> V(n,value); //vector with n copies of value
vector<T> V(n);       //vector with n copies of default for T
```

The `vector` template also provides a suitable deep copy constructor and assignment overload.

STL Ve[redacted]or Iterator Example

```
string [redacted]igitString = "45658228458720501289";
vector[redacted]nt> BigInt;

for (i[redacted]. = 0; i < DigitString.length
```

io

strin

Obtain reference to target of iterator.

Advance iterator to next element.

Insert() Member Function

An element may be inserted at an arbitrary position in a `vector` by using an iterator and the `insert()` member function:

```
vector<int> Y;
for (int m = 0; m < 100; m++) {

   Y.insert(Y.begin(), m);

   cout << setw(3) << m
        << setw(5) << Y.capacity()
        << endl;
}
```

```
  m  Y.capacity()
  0    1
  1    2
  2    4
  3    4
  4    8
  . . .
  8   16
  . . .
 15   16
 16   32
  . . .
 31   32
 33   64
 63   64
  . . .
 64  128
```

This is the worst case; insertion is always at the beginning of the sequence and that maximizes the amount of shifting.

There are overloadings of `insert()` for inserting an arbitrary number of copies of a data value and for inserting a sequence from another `vector` object.

The `resize()` allows the growth of the vector to be controlled explicitly.

[redacted]

Relational Comparison Example

```
void ivecPrint(const vector<int> V, ostream& Out);
void StringToVector(vector<int
```

Not random acces

Some differences in methods from `vector`and `deque`(e.g., no `operator[]`)

Co

Set Example

```
void EmpsetPrint(const set<Employee>& S, ostream& Out) {

   int Count;
   set<Employee>::const_iterator It;

   for (It = S.begin(), Count = 0; It != S.end();
                                        It++, Count++)
      PrintEmployee(*It, cout);
}
```

Maps and Multimaps

Associative "arrays" indexed on a given Key type.

`map` requires unique Keys (by def of order)

allows duplicate Keys `multimap`

A `map` is somewhat like a `set` that holds key-value pairs, which are only ordered on the keys.

A map element can be addressed with the usual array syntax: map1[k] = v

However: the semantics are different!

Ty

___ of items: pair<const Key, T>Once a pair has been inserted, you ca

mapuse pair constructor:

pairobject to insert into a

second.To create a

______ member fields first2 d

00 -0001");pair<const string, Employee>(Homer.getID(), Homer)

HourlyEmployeeHomer("Homer", "Simpson", "000-

Map Example

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <      >
#include
#include <map>
using namespace std;
#include

void EmpmapPrint(const map<const       , Employee*> S,
                                           ostream& Out);
void Employee               toPrint, ostream& Out);

void main() {
   Employee Ben("Ben",         , "000-00-0000");
   Employee Bill("Bill", "McQuain", "111-11-1111");
   Employee Dwight("Dwight", "Barnette", "888-88-8888");

   map<const       , Employee*> S;
// . . . continues . . .
```