Standard Template Library (STL)

- Collection of data types and algorithms
- Not native or primitive, programmer defined
- Data structures in STL are
  - Containers: classes to store and organize data
  - Iterators: objects that behave like a pointer
    - Associated with containers

Containers in STL

- Sequence: data is in a sequential fashion
  - vector
  - deque
  - list
- Associative: organize data with keys
  - set
  - multiset
  - map
  - multimap

vector Container

- A vector
  - Holds a sequence of elements
  - Stores elements in contiguous memory locations
  - Can use the array subscript operator [ ]
- Advantages to an array
  - Do not need to declare the size
  - Can add an element to a full vector
  - Vectors can report their size

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main()
{
  vector<int> numbers (5);

  for (int i = 0 ; i < 5; i++)
    cin >> numbers[i];

  for (int i = 0 ; i < 5; i++)
    cout << numbers[i] << endl;

  numbers.push_back(25);

  for (int i = 0; i < numbers.size(); i++)
    cout << numbers[i] << endl;

  return 0;
}
```

Vector Functions

- .at(i) – returns the value of element at index i
- .capacity() – returns the maximum number of elements that can be stored without additional memory
- .clear() – removes all the elements from the vector
- .empty() – returns true if the vector is empty

Vector Functions

- .pop_back() – removes the last element from the vector
- .push_back(value) – stores value in the last element of the vector
- .reverse() – reverses the order of the elements in the vector

Iterator

- An iterator may be used to access the containers member functions

```
vector<int> numbers;
vector<int>::iterator iter;

for (int count = 0; count < 10; count++)
{
  numbers.push_back(count);
}

for (iter = numbers.begin(); iter < numbers.end(); iter++)
{
  cout << *iter << " ";
}
```

Iterator Functions

- .begin() – returns an iterator pointing to the vector's first element
- .end() – returns an iterator pointing to the vector's last element
- .erase(iter1, iter2) – removes all elements from iter1 to iter2

Virtual Destructors

```
class Base
{
   public:
   Base(){ cout << "Constructing Base";}

   ~Base(){ cout << "Destroying Base";}
};

class Derive: public Base
{
   public:
   Derive(){ cout << "Constructing Derive";}

   ~Derive(){ cout << "Destroying Derive";}
 };

void main()
{
   Base* basePtr = new Derive();

   delete basePtr;
}
```

Sorting Assignment

- Study the Class Diagram to understand the inheritance structure
  - Highlight all of the .h files, right click, View Class Diagram
  - Right click the class diagram, Layout Diagram

Questions

- What is an interface?
- What is an abstract class?
- How do we make a class abstract?
- What is a virtual function?
- What is a pure virtual function?
- What does each **const** mean?

Questions

- Where is the vector created?
- What type of data is stored in the vector? int? something else?