STL Vectors

CPE 212 -- Lecture 19 continued

** Notes based on
The C++ Standard Library: A Tutorial and Reference, by Niicolai M. Josuttis

Array Example

```
//
// Array Example -- writing outside of array bounds
//
#include <iostream>

using namespace std;

int main()
{
  int  values[5];
  char letters[6] = {'A', 'B', 'C', 'D', 'E', 'F'};
  int  j, k;

  for(j = 0; j < 6; j++)  // Output letters array
    cout << letters[j];
  cout << endl;

  for(k = 0; k < 1000; k++)
  {
    cout << "Set values[" <<  k << "] = "
         << k << "   Status = ";
    values[k] = k;
    cout << "done   ";

    // Attempt to output letters array
    for(j = 0; j < 6; j++)
      cout << letters[j];
    cout << endl;
  }

  return 0;
}
```

```
-bash-$ g++ array1.cpp
-bash-$ time  ./a.out
ABCDEF
Set values[0] = 0   Status = done   ABCDEF
Set values[1] = 1   Status = done   ABCDEF
Set values[2] = 2   Status = done   ABCDEF
Set values[3] = 3   Status = done   ABCDEF
Set values[4] = 4   Status = done   ABCDEF
Set values[5] = 5   Status = done   EF
Set values[6] = 6   Status = done
Set values[7] = 7   Status = done
Set values[8] = 8   Status = done
Set values[9] = 9   Status = done

…

Set values[253] = 253   Status = done
Set values[254] = 254   Status = done
Set values[255] = 255   Status = done
Segmentation fault

real    0m0.443s
user    0m0.009s
sys     0m0.019s
-bash-$
```

Vector

- STL Sequence Container
- Models behavior of dynamically allocated array
- ***Capacity***
  - Maximum number of elements that can be stored in the vector
- ***Size***
  - Number of elements currently stored in the vector
- When capacity is exceeded, additional memory will automatically be allocated

Selected Vector Operations - 1

- vector<T> someVector;
  - Creates vector with no elements
- vector<T> someVector(int someSize);
  - Creates vector with someSize elements, each created using the default constructor for type T
- vector<T> someVector(int someSize, T value);
  - Creates vector with someSize elements of type T, all initialized to value
- ~vector<T>()
  - Destructor

Selected Vector Operations - 2

- size()
  - Number of elements currently stored
- capacity()
  - Maximum number of elements that can be stored without reallocation
- empty()
  - Returns true if empty, false otherwise
- front()
  - Returns first element but does not check to see if it exists
- back()
  - Returns last element but does not check to see if it exists

Selected Vector Operations - 3

- operator [ ]
  - Index into vector as if it is an array
- push_back(T someValue)
  - Adds someValue to back of vector
- pop_back()
  - Removes last element from vector but does not return it
- at(int someIndex)
  - Returns value at position someIndex, throwing range error exception if someIndex is out of range

Selected Vector Operations - 4

- begin()
  - Returns random access **iterator** that points to first element
- end()
  - Returns random access **iterator** that points to position AFTER last element

Note:

Elements may be inserted at arbitrary positions using insert, but performance will suffer

```
//
// Vector Example1
//
#include <iostream>
#include <vector>

using namespace std;

int main()
{
  vector<int>  values(5);
  char letters[6] = {'A', 'B', 'C', 'D', 'E', 'F'};
  int  j, k;

  for(j = 0; j < 6; j++)
    cout << letters[j];
  cout << endl;

  cout << "Size = " << values.size() << endl;
  for(k = 0; k < 1000; k++)
  {
    values.push_back(k);
    cout << "Set values[" <<  k << "] = "
         << values[k] << "   Status = ";
    cout << "done   ";

    for(j = 0; j < 6; j++)
      cout << letters[j];
    cout << endl;
  }

  cout << "Size = " << values.size() << endl;

  return 0;
}
```

```
-bash-$ g++ vector1.cpp
-bash-$ time  ./a.out
ABCDEF
Size = 5
Set values[0] = 0   Status = done   ABCDEF
Set values[1] = 0   Status = done   ABCDEF
Set values[2] = 0   Status = done   ABCDEF
Set values[3] = 0   Status = done   ABCDEF
Set values[4] = 0   Status = done   ABCDEF
Set values[5] = 0   Status = done   ABCDEF
Set values[6] = 1   Status = done   ABCDEF
Set values[7] = 2   Status = done   ABCDEF
Set values[8] = 3   Status = done   ABCDEF
Set values[9] = 4   Status = done   ABCDEF
Set values[10] = 5   Status = done   ABCDEF

…

Set values[997] = 992   Status = done   ABCDEF
Set values[998] = 993   Status = done   ABCDEF
Set values[999] = 994   Status = done   ABCDEF
Size = 1005

real    0m0.250s
user    0m0.020s
sys     0m0.028s
-bash-$
```

```
//
// Vector Example2
//
#include <iostream>
#include <vector>

using namespace std;

int main()
{
  vector<int>  values;
  int  k;

  cout << "Size = " << values.size() << endl;
  cout << "Capacity = " << values.capacity() << endl;

  for(k = 0; k < 5; k++)
  {
    values.push_back(k);
    cout << "Set values[" <<  k << "] = " << values.at(k)
         << "   Status = done" << endl;;
    cout << "Capacity = " << values.capacity() << endl;
  }

  cout << "Size = " << values.size() << endl;
  cout << "Capacity = " << values.capacity() << endl;

  cout << "Pop_back()" << endl;
  values.pop_back();
  cout << "Size = " << values.size() << endl;
  cout << "Capacity = " << values.capacity() << endl;

  return 0;
}
```

```
-bash-$ g++ vector2.cpp
-bash-$ ./a.out
Size = 0
Capacity = 0
Set values[0] = 0   Status = done
Capacity = 1
Set values[1] = 1   Status = done
Capacity = 2
Set values[2] = 2   Status = done
Capacity = 4
Set values[3] = 3   Status = done
Capacity = 4
Set values[4] = 4   Status = done
Capacity = 8
Size = 5
Capacity = 8
Pop_back()
Size = 4
Capacity = 8
-bash-$
```

Iterator

- An object that can iterate (traverse) a sequence of elements

- Different categories of iterators depending upon the type of container
  - Forward iterator
  - Bidirectional iterator
  - Random Access iterator
- Notation mimics that of pointers

```
// Vector Example3
//
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

void Print(vector<int> v);

int main()
{
  vector<int>  values;
  int  k;
  vector<int>::iterator  i;

  for(k = 0; k < 10; k++)
  {
    values.push_back(rand() % 1000);
  }
  cout << "Original Order = ";
  Print(values);

  sort(values.begin(), values.end());
  cout << "  Sorted Order = ";
  Print(values);

  i = min_element(values.begin(), values.end());
  cout << " Minimum Value = " << *i << endl;
  cout << " Maximum Value = "
       << *(max_element(values.begin(), values.end())) << endl;

  return 0;
}

void Print(vector<int> v)
{
  for(int k = 0; k < v.size(); k++)
  {
    cout << v.at(k) << " ";
  }
  cout << endl;
}
```

```
-bash-$ g++ vector3.cpp
-bash-$ ./a.out
Original Order = 807 249 73 658 930 272 544 878 923 709
  Sorted Order = 73 249 272 544 658 709 807 878 923 930
 Minimum Value = 73
 Maximum Value = 930
-bash-$
```