CSCI 390 – Special Topics in C++

Lecture 18 (10/18/18)

Time To Turn Off Cell Phones

Some std::string Langiappe

```
#include <string>
#include <stdarg.h>  // For va_start, etc.
std::string StringFormat(const std::string &Format, ...)
{
    int Size = ((int)Format.size()) * 2 + 50;
    std::string TempString;
    va_list ap;
    while (true)
    {     // Maximum two passes on a POSIX system...
        TempString.resize(Size);
        va_start(ap, Format);
        int nChars = vsnprintf((char *)TempString.data(), 
Size, Format.c_str(), ap);
        va_end(ap);
        if (nChars > -1 && nChars < Size) {  // Everything 
worked
            TempString.resize(nChars);
            return TempString;
        }
        if (nChars > -1)
            Size = nChars + 1;
        else
          Size *= 2;
    }

    return TempString;
}
#include <iostream>
int main(void)
{
  std::cout << StringFormat("Pi: %9.7f", 3.141592653589793) 
<< std::endl;
}
```

```
Console Log:
Pi: 3.1415927
```

std::ostringstream

- Header: #include <sstring>
- Purpose: Implements stream output on a string.

std::ostringstream Example

```
#include <iostream>
using std::cout; using std::endl;

#include <sstream>
using std::ostringstream;

int main(void)
{
  ostringstream oss;
  oss << "You can do anything" << endl;
  oss << "You can do with cout!" << endl;

  cout << "Here is an important message:" << endl;
  cout << oss.str() << endl;

  return 0;
}
```

```
Console Log:
Here is an important message:
You can do anything
You can do with cout!
```

std::istringstream

- Header: #include <sstring>
- Purpose: Implements stream input on a string.
- Handy for converting a string to a number.

std::ostringstream Example

```cpp
#include <iostream>
using std::cout; using std::endl;

#include <sstream>
using std::istringstream;

#include <string>

template<typename T>
T ToNumber(const std::string &str)
{
  istringstream iss(str);
  T Number;
  iss >> Number;
  return Number;
}

int main(void)
{
  auto Pi = ToNumber<double>("3.141592653589793");

  cout << "Pi: " << Pi << endl;

  return 0;
}
```

```
Console Log:
Pi: 3.14159
```

C++ STL std::vector

- Header: #include <vector>
- Iterator: Random Access
- Purpose: Implements arrays
- See: https://en.cppreference.com/w/cpp/container/vector

std::vector Member Summary

Element access

at access specified element with bounds checking

operator[] access specified element

front access the first element

back access the last element

data direct access to the underlying array

Iterators

begin cbegin returns an iterator to the beginning

end cend returns an iterator to the end

rbegin crbegin returns a reverse iterator to the beginning

rend crend returns a reverse iterator to the end

std::vector Member Summary

Capacity

empty checks whether the container is empty

size returns the number of elements

max_size returns the maximum possible number of elements

reserve reserves storage

capacity returns the number of elements that can be held in currently allocated storage

shrink_to_fit reduces memory usage by freeing unused memory

Modifiers

clear clears the contents

insert inserts elements

emplace constructs element in-place

erase erases elements

push_back adds an element to the end

emplace_back constructs an element in-place at the end

pop_back removes the last element

resize changes the number of elements stored

swap swaps the contents

std::vector Example

```
#include <iostream>
using std::cout;  using std::endl;

#include <vector>
using std::vector;

int main(void)
{
  vector<unsigned int> SomePrimes{2, 3, 5, 7, 11,
13, 17, 19};

  for (auto Prime : SomePrimes)
  {
    cout << "Prime: " << Prime << endl;
  }

  return 0;
}
```

```
Console Log:
Prime: 2
Prime: 3
Prime: 5
Prime: 7
Prime: 11
Prime: 13
Prime: 17
Prime: 19
```

std::vector Example

```
#include <iostream>
using std::cout;  using std::endl;

#include <vector>
using std::vector;

int main(void)
{
  vector<unsigned int> SomePrimes{2, 3, 5, 7, 11,
13, 17, 19};

  for (auto it = SomePrimes.rbegin(); it !=
SomePrimes.rend(); ++it)
  {
    cout << "Prime: " << *it << endl;
  }

  return 0;
}
```

```
Console Log:
Prime: 19
Prime: 17
Prime: 13
Prime: 11
Prime: 7
Prime: 5
Prime: 3
Prime: 2
```

std::vector Example

```
#include <iostream>
using std::cout;  using std::endl;

#include <vector>
using std::vector;

int main(void)
{
  vector<unsigned int> SomePrimes{2, 3, 5, 7, 11,
13, 17, 19};

  for (auto i = 0u; i < SomePrimes.size(); ++i)
  {
    cout << "Prime: " << SomePrimes[i] << endl;
  }

  return 0;
}
```

```
Console Log:
Prime: 2
Prime: 3
Prime: 5
Prime: 7
Prime: 11
Prime: 13
Prime: 17
Prime: 19
```

std::vector Example

```
#include <iostream>
using std::cout;  using std::endl;

#include <vector>
using std::vector;

#include <string>
using std::string;

int main(void)
{
  vector<string> CarMakers{"Ford", "Chevy",
"Dodge"};

  for (auto CarMaker : CarMakers)
  {
    cout << "CarMaker: " << CarMaker << endl;
  }

  return 0;
}
```

```
Console Log:
CarMaker: Ford
CarMaker: Chevy
CarMaker: Dodge
```

std::vector Example

```
#include <iostream>
using std::cout;  using std::endl;
#include <vector>
using std::vector;
#include <string>
using std::string;
#include <algorithm>
using std::sort;

int main(void)
{
  vector<string> CarMakers{"Ford", "Chevy", "Dodge"};

  sort(CarMakers.begin(), CarMakers.end());

  for (auto CarMaker : CarMakers)
  {
    cout << "CarMaker: " << CarMaker << endl;
  }

  sort(CarMakers.begin(), CarMakers.end(),
    [](const string &Left, const string &Right){ return Left > Right;});

  for (auto CarMaker : CarMakers)
  {
    cout << "CarMaker: " << CarMaker << endl;
  }

  return 0;
}
```

```
Console Log:
CarMaker: Chevy
CarMaker: Dodge
CarMaker: Ford
CarMaker: Ford
CarMaker: Dodge
CarMaker: Chevy
```

std::vector Example

```
#include <iostream>
using std::cout;  using std::endl;
#include <vector>
using std::vector;
#include <string>
using std::string;
#include <algorithm>
using std::sort;
struct sName{string First; string Last;};
bool operator<(const sName &Left, const sName &Right)
    { return (Left.Last < Right.Last) ||
    ((Left.Last == Right.Last) && (Left.First < Right.First));}
int main(void)
{
  vector<sName> Names{{"Wolfgang", "Mozart"}, {"Ludwig", "Beethoven"},
{"Johann", "Strauss"}};

  for (auto Composer : Names)
  {
    cout << "Composer: " << Composer.Last << ", " << Composer.First <<
endl;
  }

  sort(Names.begin(), Names.end());
  for (auto Composer : Names)
  {
    cout << "Composer: " << Composer.Last << ", " << Composer.First <<
endl;
  }

  return 0;
}
```

```
Console Log:
Composer: Mozart, Wolfgang
Composer: Beethoven, Ludwig
Composer: Strauss, Johann
Composer: Beethoven, Ludwig
Composer: Mozart, Wolfgang
Composer: Strauss, Johann
```

std::vector Example

```
#include <iostream>
using std::cout;  using std::endl;
#include <vector>
using std::vector;
#include <string>
using std::string;
#include <algorithm>
using std::sort;
struct sName{string First; string Last;};
int main(void)
{
  vector<sName> Names{{"Wolfgang", "Mozart"}, {"Ludwig", "Beethoven"},
{"Johann", "Strauss"}};

  for (auto Composer : Names)
  {
    cout << "Composer: " << Composer.Last << ", " << Composer.First <<
endl;
  }
  sort(Names.begin(), Names.end(),
    [](const sName &Left, const sName &Right)
    { return (Left.Last < Right.Last) ||
    ((Left.Last == Right.Last) && (Left.First < Right.First));});

  for (auto Composer : Names)
  {
    cout << "Composer: " << Composer.Last << ", " << Composer.First <<
endl;
  }

  return 0;
}
```

```
Console Log:
Composer: Mozart, Wolfgang
Composer: Beethoven, Ludwig
Composer: Strauss, Johann
Composer: Beethoven, Ludwig
Composer: Mozart, Wolfgang
Composer: Strauss, Johann
```

std::vector Ad Hoc Examples

- Vector Concatenate
- Erase elements
- Vector Add
- Vector Insert
- Algorithms:
  See: https://en.cppreference.com/w/cpp/algorithm

std::vector Ad Hoc Example

```
#include <iostream>
using std::cout;  using std::endl;
#include <ostream>
#include <vector>
using std::vector;
#include <algorithm>
#include <string>
std::ostream &operator<<(std::ostream &os, const
vector<int> &v)
{ std::string sep{""}; os << "{";
  for (const auto &i:v) {os << sep << i; sep=",";}
  os << "}"; return os; }

int main(void)
{
  vector<int> v1{2, 3, 5};
  vector<int> v2{7, 11, 13};

  cout << v1 << endl;
  cout << v2 << endl;

  return 0;
}
```

```
Console Log:
{2,3,5}
{7,11,13}
```

std::vector Ad Hoc Example

```cpp
#include <iostream>
using std::cout;  using std::endl;
#include <ostream>
#include <vector>
using std::vector;
#include <algorithm>
#include <string>
template<typename T>
std::ostream &operator<<(std::ostream &os, const
vector<T> &v)
{ std::string sep{""}; os << "{";
  for (const auto &i:v) {os << sep << i; sep=",";}
  os << "}"; return os; }

template<typename T>
vector<T> operator||(const vector<T> &lhs, const
vector<T> &rhs)
{ vector<T> v{lhs}; v.insert(v.end(), rhs.begin(),
rhs.end()); return v; }
int main(void)
{
  vector<int> v1{2, 3, 5};
  vector<int> v2{7, 11, 13};

  cout << v1 << endl;
  cout << v2 << endl;

  cout << (v1 || v2) << endl;

  return 0;
}
```

```
Console Log:
{2,3,5}
{7,11,13}
{2,3,5,7,11,13}
```