Standard Template Library (STL) II

Numerical Algorithms

UNIVERSITY of HOUSTON
DIVISION OF RESEARCH
HEWLETT PACKARD ENTERPRISE DATA SCIENCE INSTITUTE

Learning Objectives

- Random numbers
- Lambda function and function objects
  - Nameless function
  - functors
- Accumulation/Reduction
- transform and transform_reduce
- Parallel STL
  - Execution policies
    - Sequential, SIMD, Thread, Thread+SIMD
  - foreach, transform, reduce and transform+reduce
- Generic Algorithms
  - Sequence, set, and sorting algorithms

Random Number Generation

- Custom random number structures
  - You can create your own random numbers generators
    - Random and Pseudo-random numbers
  - Several random number distribution standard data structures exist
  - Make sense to have a standard portable implementations of them!
- Standard Template Library (STL) Numerical Algorithms
  - Includes libraries for all such data structures
    - Like random, numeric, iomanip, classes

Random Number Generation

- C library for pseudo-random numbers generation existed

  Included: rand, srand, RAND_MAX

Excerpt from: randomC_library.cpp

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
   srand(time(nullptr)); // use current time as SEED for random generator

   auto random_variable = rand();
   cout << "Random value on [0 " << RAND_MAX << "]: "  << random_variable << '\n';

   cout << "generate 6 random ints\n";
   for(int i=0; i<6;i++) cout <<rand() << ' ';

   cout<<"\n\ngenerate 6 random floats\n";

   for(int i=0; i<6;i++) cout << float(rand())/(RAND_MAX) <<' ';
   cout<< endl;
}
```

```
#OUTPUT

Random value on [0 2147483647]: 847527493
generate 6 random ints
1997235334 1221448561 154969909 2129726074 1418142480 1319447346
generate 6 random floats
0.212854 0.524828 0.413379 0.598311 0.988006 0.708397
```

Random Number Generation

- Contributions from std c++
  - Sample from distributions using STL Algorithms
    - Support for real random number generator
    - *random_device* class for both true and psuedo random number generator
      - uniform, normal, binomial, etc distributions non-deterministic random numbers

Example of true non-deterministic random generation, within a range: range_random.cpp

```
#include <iostream>
#include <random>

using namespace std;

int main()
{
   std::random_device  mydevice;  //   Initialize the random number engine
   std::uniform_int_distribution<> random_int(1, 100); // sampling from uniform distribution

   // Use random_int to transform the random unsigned int
   // generated by mydevice into an int in [1, 100]
   for (int n = 0;  n != 6; ++n) cout << random_int(mydevice) << ' ';
   cout << '\n';
}
```

```
#possible output
 61 66 76 97 75 15
```

Random Number Generation

- Contributions from std c++
  - Sample from distributions using STL Algorithms
    - Support for pseudo-random number generator
      - uniform, normal, binomial, etc distributions

Example of pseudo non-deterministic random generation, within a range: pseudo_range_random.cpp

```
#include <iostream>
#include <random>

using namespace std;

int main()
{
   std::random_device  mydevice;  //   random number engine
   std::mt19937 gen(mydevice()); // Standard mersenne_twister_engine seeded with mydevice()
   std::uniform_int_distribution<> random_int(1, 100); // sampling from uniform distribution

   // Use random_int to transform the random unsigned int
  // generated by mydevice into an int in [1, 100]
   for (int n = 0;  n != 6; ++n) cout << random_int(mydevice) << ' ';
   cout << '\n';
}
```

```
#possible output
 61 66 76 97 75 15
```

Random Number Generation

- Contributions from std c++
  - Sample from real distributions using STL Algorithms
    - Support several random number distributions
      - *default_random_engine* class for **deterministic** random numbers

Excerpt from: reproducible_range_random.cpp

```
#include <iostream>
#include <random>
using namespace std;

int main()
{
  int seed=19937; // fixed random seed
  std::default_random_engine  mydevice(seed); // seeding with a fixed seed
  uniform_int_distribution<> random_int(1, 100); // sampling from uniform distribution

   // Use random_int to transform the random unsigned int
  // generated by mydevice into an int in [1, 100]
  for (int n = 0;  n != 6; ++n) cout << random_int(mydevice) << ' ';
  cout << '\n';
}
```

```
#possible output
16 47 51 11 79 39
```

Random Number Generation

- Sample from real distributions using STL Algorithms
  - Support several random number distributions
    - Uniform, normal, binomial, etc distributions

Excerpt from: float_real_range_random.cpp

```
#include <iostream>
#include <random>
using namespace std;

int main()
{
  random_device mydevice; //   random number engine
  uniform_real_distribution<> random_float(0,1);  // sampling from uniform real distribution

  for (int n = 0;  n != 6; ++n) cout << random_float(mydevice) << ' ';
  cout << '\n';
}
```

```
#possible output
0.385541 0.420061 0.00424491 0.450918 0.838626 0.218411
```

Random Number Generation

- Sample from normal distribution
  - User provides *mean, and stddev* to get desired distribution
  - Non-deterministic

```
#include <iostream>
#include <random>


using namespace std;
int main()
{
   random_device  mydevice;
   normal_distribution<> d{5, 2}; //mean=5; stddev =2;
   for (int n = 0; n <6; n++) cout << d(mydevice) <<" ";
   Cout << endl;
}
```

```
#possible output
1.37403 5.9097 1.88431 4.8685 4.34088 5.03269 5.38202 6.9686 2.78879 8.46331
```

Lambda expression or nameless functions

- C++ provides support for lambda expression

Excerpt from: lambda_expression1.cpp

```
#include <iostream>

using namespace std;
int main()
{

 auto welcome = []() { cout << "Welcome to world of CXX"<<endl;};

    welcome();
}
```

Lambda expression or nameless functions

- C++ provides support for lambda expression

Excerpt from: lambda_expression2.cpp

```
#include <iostream>
#include <string>

using namespace std;
int main()
{

string s1;
auto welcome = [](string name ) { cout << name + ", Welcome to world of CXX"<<endl;};

cout << "Enter your name"<<endl;
getline(cin,s1);

  welcome(s1);


}
```

Lambda expression or nameless functions

- C++ provides support for lambda expression
  - Capture existing variables

Excerpt from: lambda_expression3.cpp

```
#include <iostream>
#include <string>

using namespace std;
int main()
{

string s1;
string s2 = "Welcome to world of CXX";
auto welcome = [&](string name ) { cout << name + ", " +  s2 <<endl;};

cout << "Enter your name"<<endl;
getline(cin,s1);

   welcome(s1);


}
```

Lambda expression or nameless functions

- C++ provides support for array loading algorithms
  - Apply to loading vectors

Excerpt from: lambda_expression4.cpp

```
#include <vector>
#include <iostream>
#include <random>


using namespace std;
int main()
{

 random_device mydevice;
 uniform_real_distribution<> random_float(0,1);

 auto generate_float= [&]() {  return random_float(mydevice) ; };

       vector<float> v(10);

       for (auto & item: v)   item = generate_float();

       for (auto x : v)         cout << x << endl;

}
```

Array filling algorithms

- C++ provides support for array loading algorithms
  - *fill and generate*
  - Apply to loading vectors

Excerpt from: fill.cpp

```
#include <vector>
#include <iostream>
#include <algorithm>

using namespace std;
int main()
{
	vector<float> v(10);


	fill(v.begin(), v.end(), 11);

	for (auto x : v)	cout << x << endl;


}
```

Array filling algorithms

- C++ provides support for array loading algorithms
  - *fill and generate*
  - Apply to loading vectors

Excerpt from: generate.cpp

```
#include <vector>
#include <iostream>
#include <algorithm>
#include <random>

using namespace std;
int main()
{
 random_device mydevice;
 uniform_real_distribution<> random_float(0,1);

 vector<float> v(10);

 generate(v.begin(), v.end(), [&]() {      return random_float(mydevice) ; });

 for (auto x : v)       cout << x << endl;

}
```

Reduction algorithms

- C++ provides support for reduction expression
  - Apply to reducing vector to scaler value

Excerpt from: reduce.cpp

```
#include <vector>
#include <iostream>
#include <algorithm>
#include <random>

using namespace std;
int main()
{

  random_device mydevice;
  uniform_real_distribution<> random_float(0,1);

  vector<float> v(1000);

  generate(v.begin(), v.end(), [&]() { return random_float(mydevice) ; });

  auto x = reduce (v.begin(), v.end(), 0.0f, plus<float>());

  cout << x/v.size() << endl;


}
```

Transformation algorithms

- Support for transformation statement
  - Apply to multiple vectors
    - Vector addtion V3=V1+V2

Excerpt from: transform.cpp

```
#include <vector>
#include <iostream>
#include <algorithm>
#include <transform>

using namespace std;
int main()
{

 random_device mydevice;
 uniform_real_distribution<> random_float(0,1);

 vector<float> v1(1000) , v2(1000), v3(1000);

 generate(v1.begin(), v1.end(), [&]() {      return random_float(mydevice) ; });
 generate(v2.begin(), v2.end(), [&]() {      return random_float(mydevice) ; });

 transform (v1.begin(), v1.end(), v2.begin(), v3.begin(), [](float a, float b) { return  a+b;} );

for (int  i =0; i <6; i++) cout << v3[i] << endl;

}
```

Transformation + reduction algorithms

- Support for transform followed by reduction
- Apply to multiple vectors
  - Vector addtion a= sum (V1 * V2)

Excerpt from: transform_reduce.cpp

```
#include <vector>
#include <iostream>
#include <algorithm>
#include <transform>

using namespace std;
int main()
{

 random_device mydevice;
 uniform_real_distribution<> random_float(0,1);

 vector<float> v1(1000) , v2(1000), v3(1000);

 generate(v1.begin(), v1.end(), [&]() {      return random_float(mydevice) ; });
 generate(v2.begin(), v2.end(), [&]() {      return random_float(mydevice) ; });


auto x = transform_reduce (v1.begin(), v1.end(), v2.begin(), 0.0f, plus<float>(),  [](float a, float b) { return  a*b;}  );

  cout << x/v.size() << endl;

}
```

for_each algorithm

- C++ provides support for for_each expression
  - Apply your choice algorithms to each item in range

Excerpt from: for_each.cpp

```
#include <vector>
#include <iostream>
#include <algorithm>
#include <random>
#include <cmath>

using namespace std;
int main()
{

  random_device mydevice;
  uniform_real_distribution<> random_float(0,1);

  vector<float> v(1000);

  generate(v.begin(), v.end(), [&]() { return random_float(mydevice) ; });

   for_each (v.begin(), v.end(),   [&] (float & x) {  x = tan(x) + log(x); });

   for (int  i =0; i <6; i++) cout << v[i] << endl;


}
```