CS 485
Advanced Object Oriented Design

C++

Lambda, Function Objects,

Brace Initializers
constexpr,

Move Constructors/Rvalue references

Spring 2017

Reading

- Meyers 7,15, 31-34

Inline Functions

- Declare function body in .h file
- Can allow compiler to insert function directly at the invocation spot
  - no jump to function/faster code
  - bigger (in file size) executable

- inline keyword
  - allow the function body to be in .cpp file

```
inline int Math::sum(int x, int y)
{
  return x+y;
}
```

lambda

- Closure
  - a feature in many languages
    - C++, Python, Ruby, Scheme....
  - lexically (static) scoped name binding with first-class functions
  - first-class functions
    - pass as parameter, return from function, assign to variables, store in data structure
  - lexically (static) scoped name binding
    - bind variables at compile time
    - not dynamic (at runtime)

https://en.wikipedia.org/wiki/Closure_(computer_science)

https://martinfowler.com/bliki/Lambda.html

Binding of Available Variables

```
std::vector<int> cLocalVector;

auto list = { 1,2,3,4,5 };

// capture the cLocalVector by reference
std::for_each(list.begin(), list.end(),
  [&](auto cItem)
{
  cLocalVector.push_back (cItem);
});

// [=] capture the variables by copy

// always capture this by reference

// [&cLocalVar, list]
// capture cLocalVar by reference
// capture list by copy
```

Be careful!

```
int changeOperatorSecretRef (std::function<int (int, int)> &func)
{
  int secret = 42;
  func = [&](int param1, int param2) -> int
  {
    return param1 * param2 * secret;
  };
  return func (2, 2);
}
```

- What is the issue here?

std::function< >

- #include <functional>
- Represent a function
  - like a function pointer

```
void foo(int);

std::function<void (int)> cFunc =  foo;

cFunc(1);
```

- std::bind
  - bind a function call to a particular set of arguments

```
auto f = std::bind(&foo, 1 );

f();
auto f2 = std::bind(&Account::deposit, pcAcct, pay);
f2();
```

  - std::placeholders::

std::initializer_list<T>

- provide access to an array of objects of type const T
- immutable sequence
  - begin()
  - end()
- Allows you to create a constructor that takes a list of items
  - often a constructor

http://www.stroustrup.com/C++11FAQ.html#init-list

http://www.stroustrup.com/C++11FAQ.html#uniform-init

http://en.cppreference.com/w/cpp/utility/initializer_list

Example

```
class Example
{
   Example(std::initializer_list<int> cList);

private:
   std::vector<int> cVec;

};

Example::Example(std::initializer_list<int> cList)
{
   for_each(cList.begin(), cList.end(),
    [this](auto cItem)
    {
      cVec.push_back(cItem);
    };
}
```

Brace Initialization

- Brace Initialization
  - uniform initialization
  - can be used anywhere

```
int x = 0;
int y { 0 };
int x(0);

int x = {0};
```

```
class Example
{
 …
private:
   int mX = 0;
   int mY {1};
   int mz (0); //error
};
```

```
std::vector<int> cVec{1,2,3};
// previously
cVec.push_back(1);
cVec.push_back(2);
cVec.push_back(3);
```

Meyers, Item 7

Safety

- Prevents type narrowing

```
double x, y;
...
int sum{ x + y }; // invalid

int sum2(x+y); // valid, truncate to int

int sum3 = x+y; // valid, truncate to int
```

Most Vexing Parse

- C++ rule: anything that can be parsed as a declaration must be

```
Widget w(10); // ok
Widget f();   // ???

Widget z{};   // ok
```

https://web.archive.org/web/20160709112804/http://www.informit.com/guides/content.aspx?g=cplusplus&seqNum=439

constexpr

- Must be possible to evalute the value of the function or variable at compile time
- The function or variable can be used in constant expressions
  - evaluated at compile time
  - off load computation to compile time
    - write functions that get executed at compile time and use the results to generate data structures
- constexpr member function body must be in the header file
  - why?

http://en.cppreference.com/w/cpp/language/constexpr

RValue References

- Lvalue - left value
  - "persists beyond a single expression"[2]
  - can be assigned to
  - assigned a named location in memory[1]
- R value - right value
  - everything else is an rvalue
    - you can use an lvalue as an rvalue
  - cannot be assigned to
  - *temporary value*
  - *result of calculation*
- lhs = rhs

[1]http://eli.thegreenplace.net/2011/12/15/understanding-lvalues-and-rvalues-in-c-and-c

[2]https://msdn.microsoft.com/en-us/library/f90831hc.aspx
http://www.artima.com/cppsource/rvalue.html

&&

https://msdn.microsoft.com/en-us/library/dd293665.aspx

blog.smartbear.com/c-plus-plus/c11-tutorial-introducing-the-move-constructor-and-the-move-assignment-operator/

http://stackoverflow.com/questions/3106110/what-are-move-semantics/11540204#11540204

http://eli.thegreenplace.net/2011/12/15/understanding-lvalues-and-rvalues-in-c-and-c/

http://en.cppreference.com/w/cpp/language/move_assignment

http://www.drdobbs.com/cpp/object-swapping-part-3-swapping-and-movi/232700458

Move Constructors

```
ExampleClass(ExampleClass &&rcData);
```

- Steal data from parameter
  - rvalue means the param is a temporary value
  - leave parameter in a different state
    - not unlike CopyAndSwap assignment operator