CS 485
Advanced Object Oriented Design

C++

Lambda, Function Objects,

Brace Initializers
constexpr,

Move Constructors/Rvalue references

Spring 2019

Type Checking

Templates

- Another form of abstraction
  - no inheritance or composition

```
template<class T>
bool URLFile::readData (T & rData)
{
  bool retVal = false;
  if (mcStream >> rData)
  {
    retVal = true;
  }
  return retVal;
}
```

Functor

Functor Example

```
class LogFunctor {

  public:
    LogFunctor (std::string prefix) : mPrefix (prefix) {};

    std::string operator() (std::string msg) const
    {
      return mPrefix + ": " + msg;
    }

  private:
    std::string mPrefix;
};
```

Functor Example

```
void doWork (LogFunctor &rcLog) {

  for (int i = 0; i < 10; ++i)
  {
    std::cout << rcLog (std::to_string (i*i));
    std::cout << std::endl;
  }
}

std::vector<std::string> cVecOfValues;

LogFunctor cTestLog ("Testing");
LogFunctor cActualLog ("Actual");

doWork (cTestLog);
doWork (cActualLog);
```

Functor Example

```
std::vector<std::string> cVecOfValues;

LogFunctor cTestLog ("Testing");
LogFunctor cActualLog ("Actual");


// change each entry in the vector using the functor
std::transform (cVecOfValues.begin (), cVecOfValues.end (),
  cVecOfValues.begin (), cTestLog);


// COMPILE ERROR!
std::for_each (cVecOfValues.begin (), cVecOfValues.end (), cIntFunctor);
```

std::transform

Possible implementation

First version

```
template<class InputIt, class OutputIt, class UnaryOperation>
OutputIt transform(InputIt first1, InputIt last1, OutputIt d_first,
                   UnaryOperation unary_op)
{
    while (first1 != last1) {
        *d_first++ = unary_op(*first1++);
    }
    return d_first;
}
```

https://en.cppreference.com/w/cpp/algorithm/transform

Functor

```
class CountingLogFunctor {

  public:
    CountingLogFunctor (std::string prefix) : mPrefix (prefix) {};

    std::string operator() (std::string msg)
    {
      mBytesWritten += (mPrefix + ": " + msg).length();
      return std::to_string(mBytesWritten) + " " + mPrefix + ": " + msg;
    }

  private:
    std::string mPrefix;
    long long mBytesWritten = 0;
};
```

Abstract

```
template<class T>
void doWorkTemplate (T &rcLog) {

  for (int i = 0; i < 10; ++i)
  {
    std::cout << rcLog (std::to_string (i*i));
    std::cout << std::endl;
  }
}

 CountingLogFunctor cCountingLog ("Count");

 doWorkTemplate(cTestLog);
 doWorkTemplate(cCountingLog);
```

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

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?

(Poorly) Simulate Functor with lambda

```
int state = 0;

auto lambdaWithState = [state](auto &rStr) mutable
  {
    std::cout << "Lambda: " << state++ << " " << rStr << std::endl;
  };

// simulate functor with lambda
std::for_each (cVecOfValues.begin (), cVecOfValues.end (),
  lambdaWithState
);

// simulate functor with lambda
std::for_each (cVecOfValues.begin (), cVecOfValues.end (),
  lambdaWithState
);
```

Lambda usage

```
// store a function into a variable
auto func =
[](int param1, int param2) -> int
{
  return param1 + param2;
};

// invoke the function
std::cout << func (1, 2) << std::endl;
```

Lambda usage

```
// create an array of functions
std::function<int (int, int)> aMathOps[2] =
{
  [] (int param1, int param2) -> int
  {
    return param1 + param2;
  },
  [](int param1, int param2) -> int
  {
    return param1 - param2;
  }
};
```

Accept a Lambda

```
int takeOperator (std::function<int (int, int)> func)
{
  return func (1, 100);
}
```

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> mcVec;

};

Example::Example(std::initializer_list<int> cList)
{
   for_each(cList.begin(), cList.end(),
    [this](auto cItem)
    {
      mcVec.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
-

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

Constexpr

```
int foo ();
constexpr int constExprFoo ();
```

```
int size = 9;
```

```
// int aSize[size];  // illegal

// int aFoo[foo ()]; // illegal

int aConstexprFoo[constExprFoo ()]; // legal
```

Rvalue Reference: Motivation

```
ExampleClassWithoutMove copyBackExampleClassWithoutMove()
{
  ExampleClassWithoutMove cRetVal(1,2,3);

  return cRetVal;
}
cOriginalExampleClassWOMove = copyBackExampleClassWithoutMove();
```

What happens?

http://thbecker.net/articles/rvalue_references/section_01.html

RValue References

- Lvalue - left value
- R value - right value
- 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

Move & Perfect Forwarding