2023

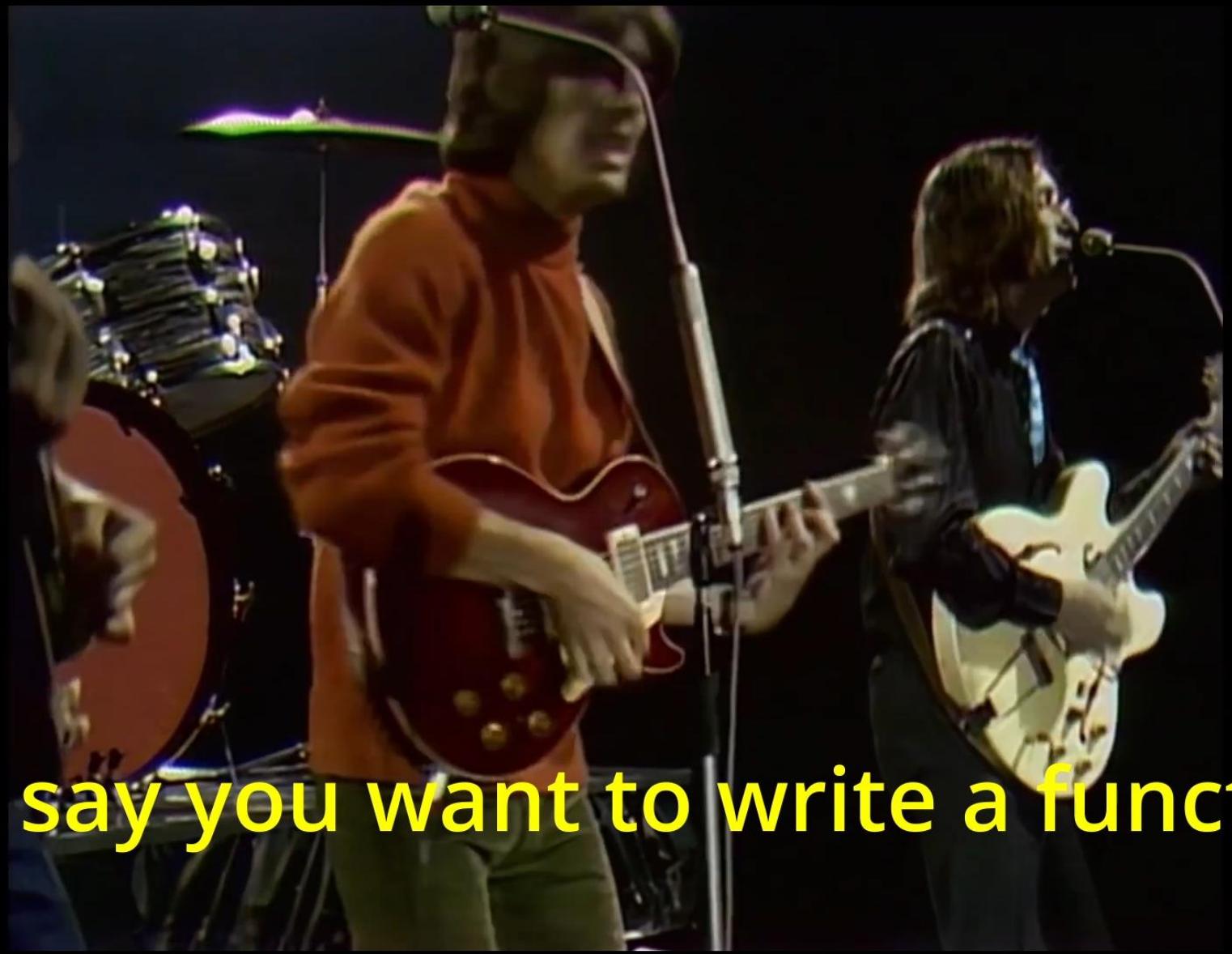
Value Oriented Programming. Part 1

You Say You Want to Write a Function

Tony Van Eerd

C++ now

Outline


You say you want to write a function

You say you want to write a function....

Value Oriented Programming, Part 1: Functions

Tony Van Eerd

C++Now 2023

CHRISTIE

A Possible Future of Software Development

Sean Parent

Principal Scientist & Manager

Software Technology Lab

~~July 25, 2007~~

May 16, 2007

BoostCon #1

Adobe

CHRISTIE

You say you want to write a function....

Value Oriented Programming, Part 1: Functions

Tony Van Eerd

C++Now 2023

CHRISTIE

A long time ago in a galaxy far, far away....

You say you want to write a function....

CHRISTIE

You say you want to write a function....

Now that is a lie!

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …
};
```

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …
};
```

- Add a feature
- Fix a bug

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // special case North
  if (???)
    …
};
```

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  if (???)
    …
};
```

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  if (proj->getPose().orientation().yaw()


};
```

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  if (proj->getPose().orientation().yaw() < 5 * M_PI / 180)
     …

};
```

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  if (proj->getPose().orientation().yaw() <   5 * M_PI / 180
   || proj->getPose().orientation().yaw() > 355 * M_PI / 180)
     …

};
```

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <   5 * M_PI / 180
   || dir.yaw() > 355 * M_PI / 180)
     …

};
```

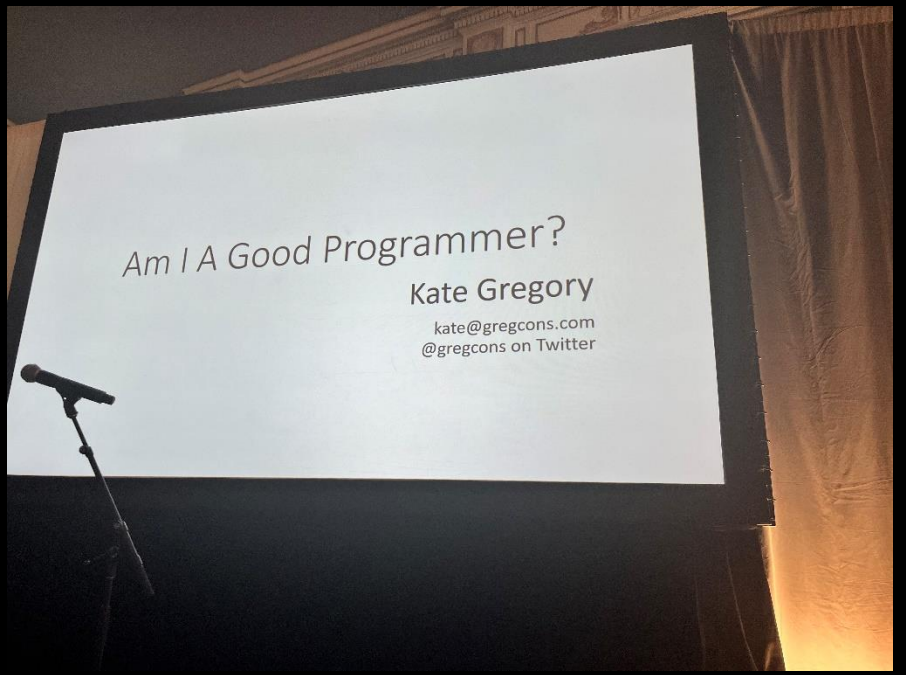
```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <       tolerance   * M_PI / 180
   || dir.yaw() > (360-tolerance) * M_PI / 180)
     …

};
```

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <       tolerance   * M_PI / 180
   || dir.yaw() > (360-tolerance) * M_PI / 180)
     …

};
```

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <       tolerance   * M_PI / 180
   || dir.yaw() > (360-tolerance) * M_PI / 180)
     …

};
```

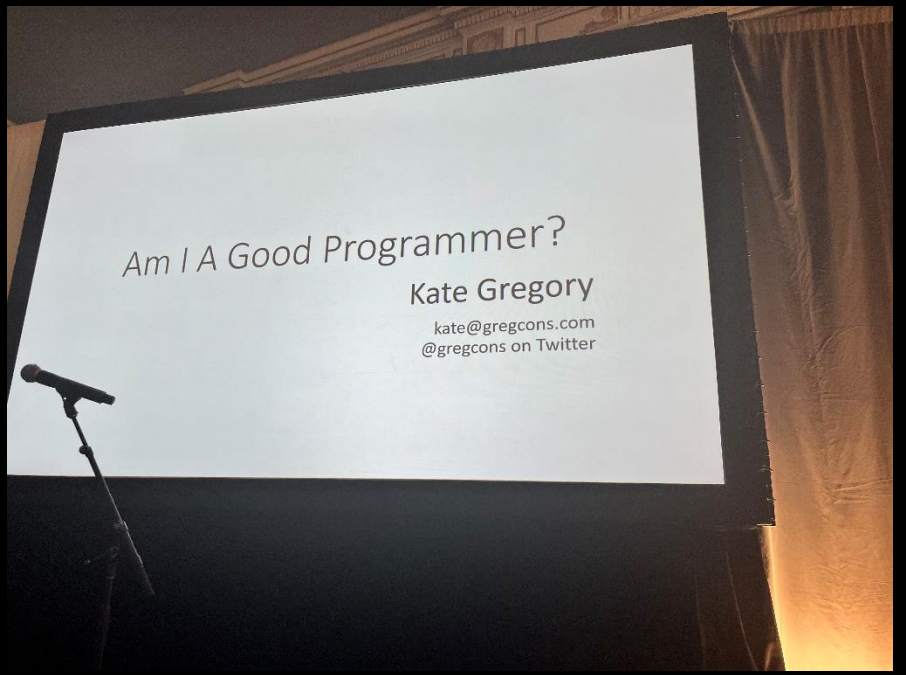


CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <       tolerance   * M_PI / 180
   || dir.yaw() > (360-tolerance) * M_PI / 180)
     …

};
```

CHRISTIE

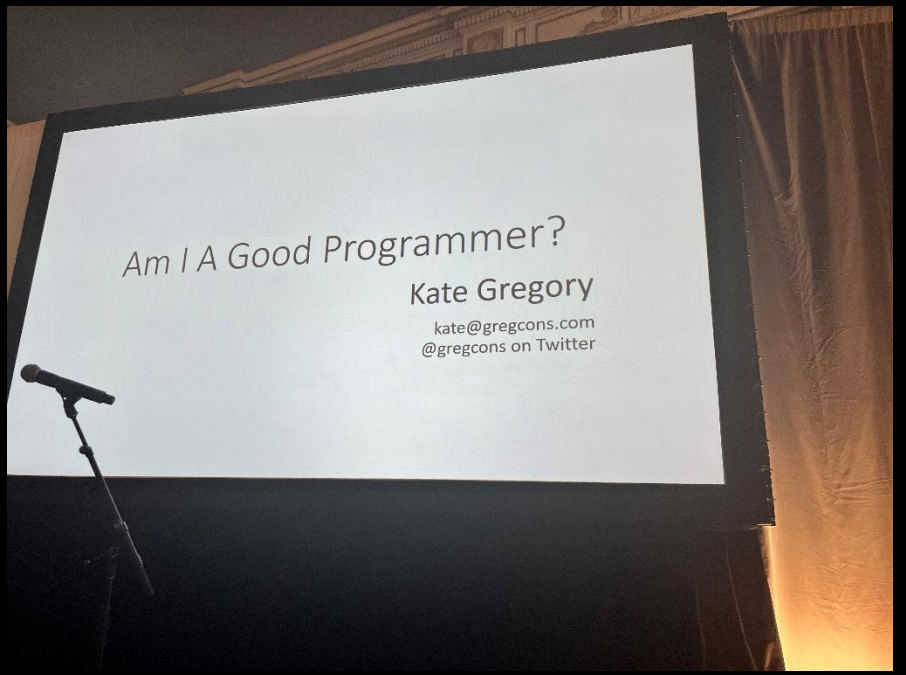
```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <       tolerance   * M_PI / 180
   || dir.yaw() > (360-tolerance) * M_PI / 180)
     …

};
```



How did we get here?

CHRISTIE

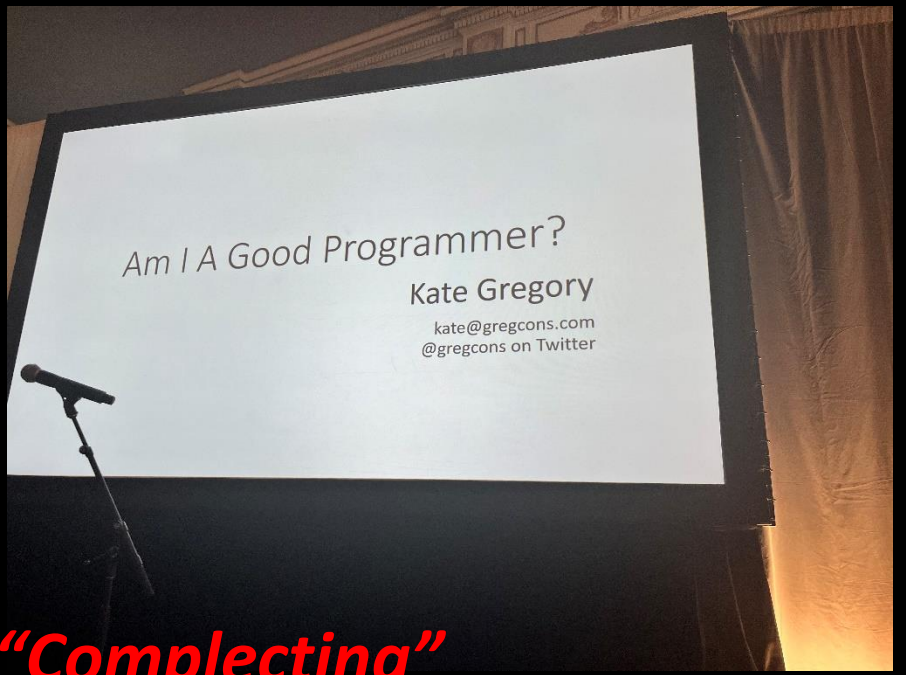
```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <       tolerance   * M_PI / 180
   || dir.yaw() > (360-tolerance) * M_PI / 180)
     …

};
```



How did we get here?
One step at a time.

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <       tolerance   * M_PI / 180
   || dir.yaw() > (360-tolerance) * M_PI / 180)
     …

};
```



"Complecting"

How did we get here?
One step at a time.

CHRISTIE

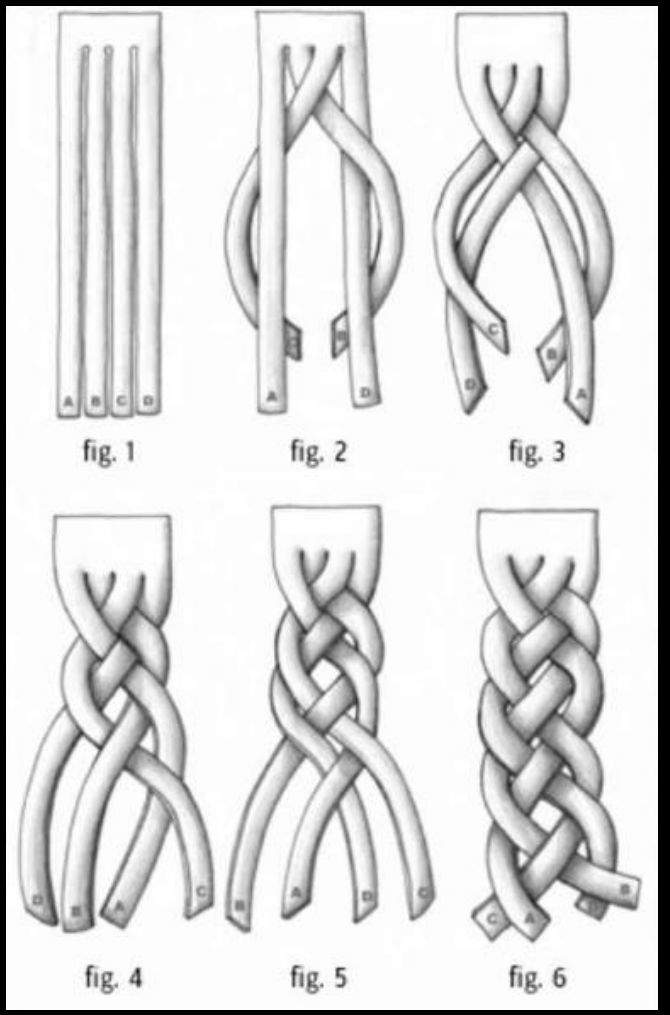
```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <       tolerance   * M_PI / 180
   || dir.yaw() > (360-tolerance) * M_PI / 180)
     …

};
```

"Complecting"

How did we get here?
One step at a time.

CHRISTIE

Rich Hickey (Clojure)
Simple Made Easy



Simple/Complex

plecto (latin) – fold / braid

Simple – one fold

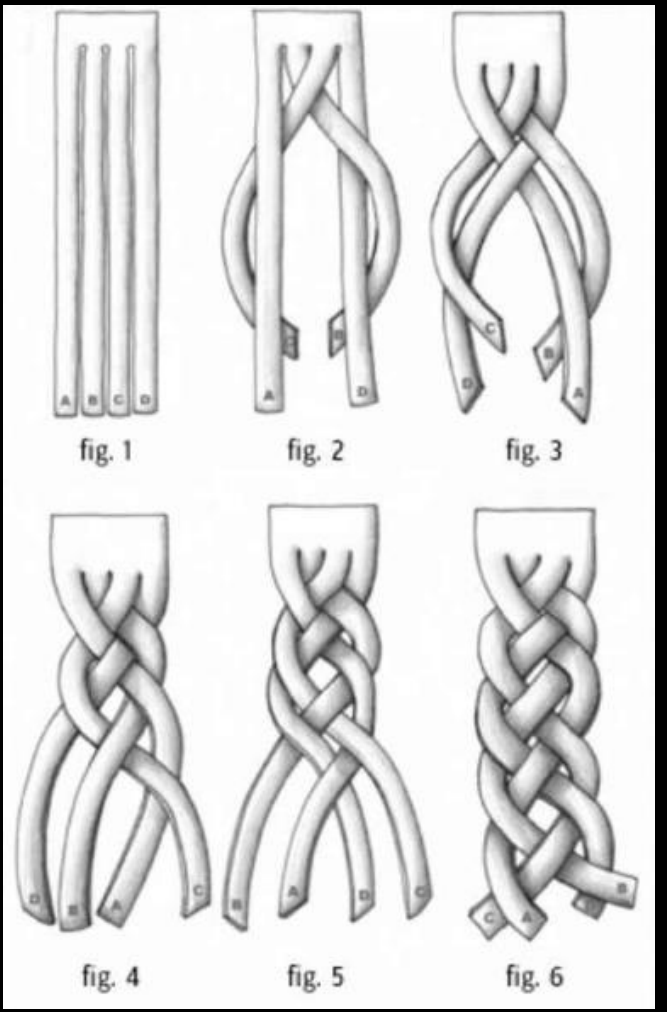
Complex – many folds

CHRISTIE

Rich Hickey (Clojure)
Simple Made Easy

Easy
adjacens (latin)
close at hand

Simple/Complex
plecto (latin) – fold / braid
Simple – one fold
Complex – many folds

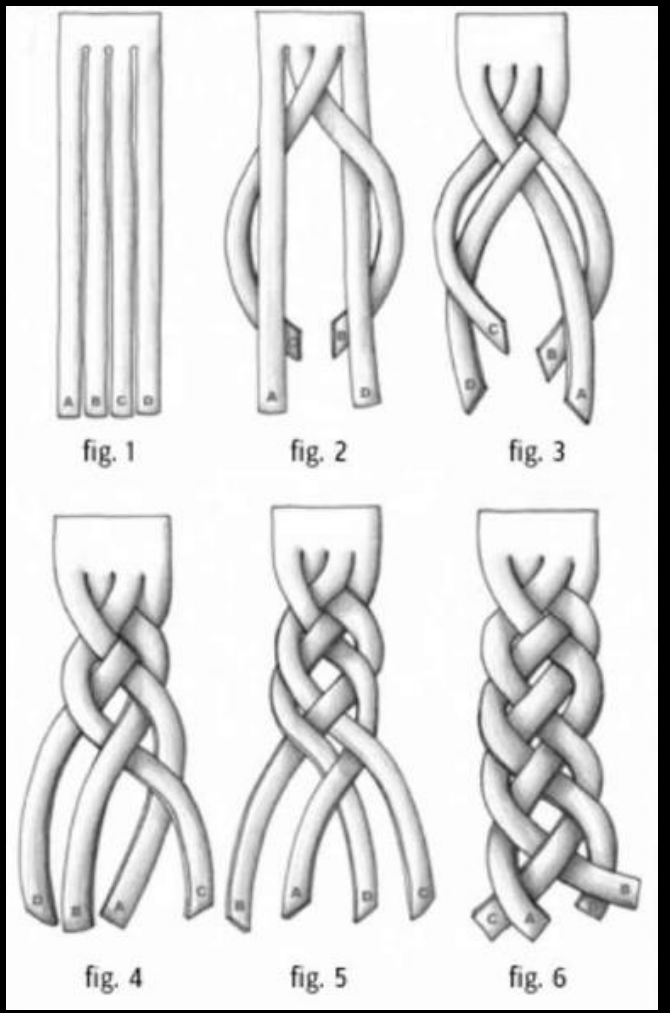


CHRISTIE

Rich Hickey (Clojure)
Simple Made Easy

Easy
adjacens (latin)
close at hand

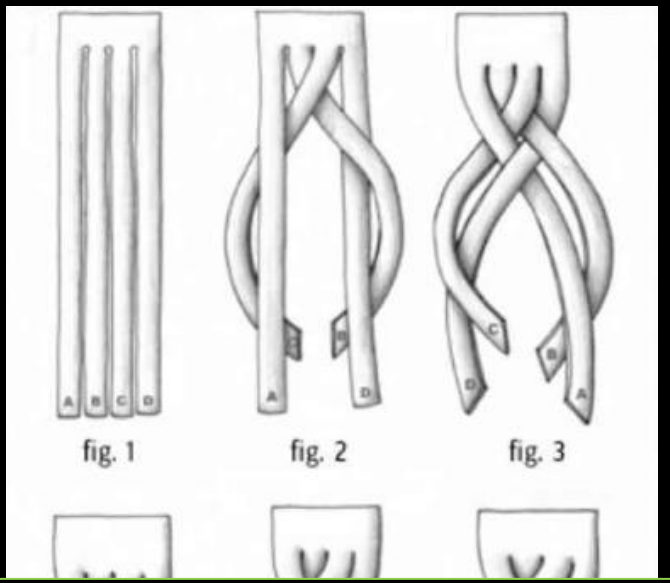
Simple/Complex
plecto (latin) – fold / braid
Simple – one fold
Complex – many folds
Complecting – interweaving



CHRISTIE

Rich Hickey (Clojure)
Simple Made Easy

Easy
adjacens (latin)
close at hand



Simple/Complex
plecto (latin) – fold / braid
Simple – one fold
Complex – many folds
Complecting – interweaving

Complecting Made Easy

Tony Van Eerd
C++Now, May 2021

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <       tolerance   * M_PI / 180
   || dir.yaw() > (360-tolerance) * M_PI / 180)
     …

};
```

"Complecting"

How did we get here?
One step at a time.

CHRISTIE

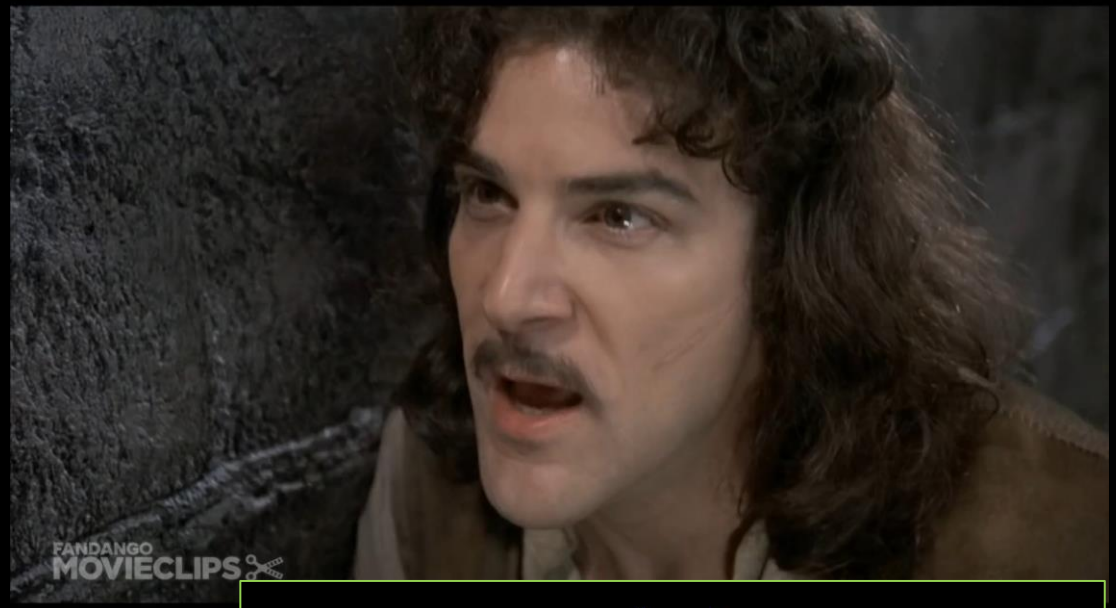
```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
 if (proj->getPose().orientation().yaw() <   5 * M_PI
  || proj->getPose().orientation().yaw() > 355 * M_PI

};
```

"Complecting"

How did we get here?
One step at a time.

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
 if (proj->getPose().orientation().yaw() <   5 * M_
  || proj->getPose().orientation().yaw() > 355 * M_

};
```



"Comp

Complecting Made Easy

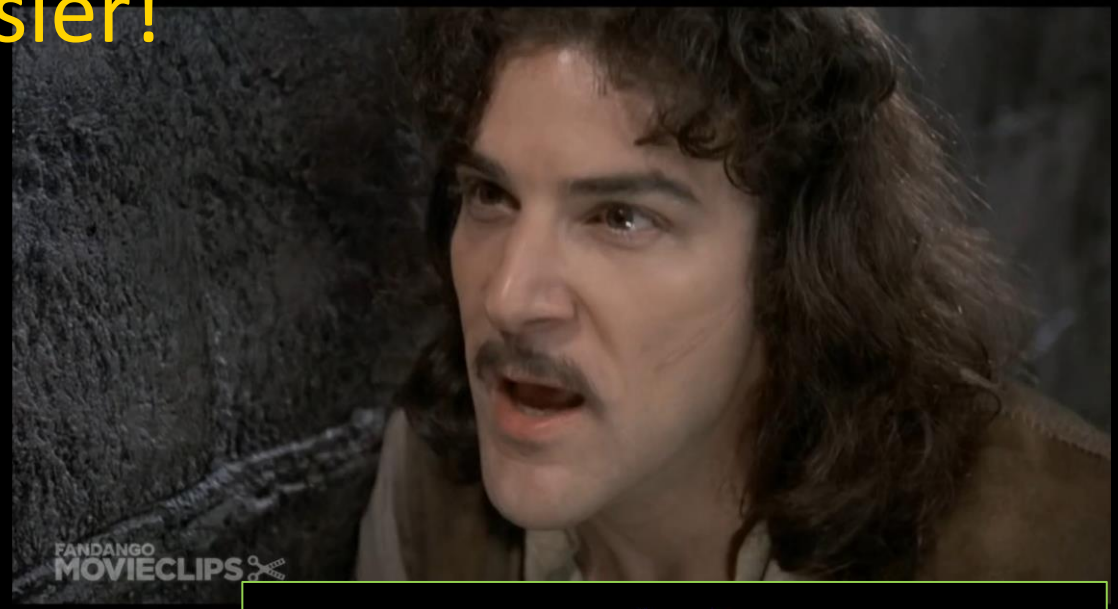
Tony Van Eerd

C++Now, May 2021

CHRISTIE

NOT writing a function is easier!

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
 if (proj->getPose().orientation().yaw() <   5 * M_
  || proj->getPose().orientation().yaw() > 355 * M_

};
```



"Comp

Complecting Made Easy

Tony Van Eerd

C++Now, May 2021

CHRISTIE

You say you want to write a function....

Now that is a lie!

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <       tolerance   * M_PI / 180
   || dir.yaw() > (360-tolerance) * M_PI / 180)
     …

};
```

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <       tolerance   * M_PI / 180
   || dir.yaw() > (360-tolerance) * M_PI / 180)
     …

};
```

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  // if projector facing North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <      tolerance  * M_PI / 180
   || dir.yaw() > (360-tolerance) * M_PI
     …
```

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  // if projector_facing_North…
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <      tolerance  * M_PI / 180
   || dir.yaw() > (360-tolerance) * M_PI
     …
```

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  // if projector_facing_North()
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <      tolerance  * M_PI / 180
   || dir.yaw() > (360-tolerance) * M_PI
     …
```

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  if (projector_facing_North(proj))
  double tolerance = 5.0;
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  Orientation dir = proj->getPose().orientation();
  if (dir.yaw() <       tolerance   * M_PI / 180
   || dir.yaw() > (360-tolerance) * M_PI
     …
```

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // if projector facing North…
  if (projector_facing_North(proj))
    …

};
```

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  if (projector_facing_North(proj))
    …

};
```

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // check for north OTHERWISE …
  if (projector_facing_North(proj))
    …

};
```

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // check for north OTHERWISE …
  if (projector_facing_North(proj))
    …

};
```

*My favourite comment word is "**otherwise**"*

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // check for north OTHERWISE
  // I don’t have a good talk example
  if (projector_facing_North(proj))
    …

};
```

*My favourite comment word is “**otherwise**”*

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // check for north OTHERWISE …
  if (projector_facing_North(proj))
  if (isFacingNorth(proj))
    …

};
```

Don’t include param types in function names

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // check for north OTHERWISE …
  if (projector_facing_North(proj))
  if (isFacingNorth(proj))
    …

};
```

Don’t include param types in function names

```
string s1 = string_from_int(17);
string s2 = to_string(17);
```

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // check for north OTHERWISE …
  if (isFacingNorth(proj))
    …

};
```

```
bool isFacingNorth(Projector proj)
{
  Degrees tolerance = Degrees(5.0);
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  auto dir = proj.getPose().orientation();
  if (dir.yaw() < Degrees(tolerance)
   || dir.yaw() > Degrees(360)-tolerance)
     return true;
   return false;
};
```

#CppKoan

CHRISTIE

#CppKoan

There's a function for that

CHRISTIE

#CppKoan

There's a function for that

What's it called?

CHRISTIE

#CppKoan

There's a function for that

What's it called?

Guess

CHRISTIE

#CppKoan

There's a function for that

What's it called?

Guess

readConfigFile()?

CHRISTIE

#CppKoan

There's a function for that

What's it called?

Guess

readConfigFile()?

Pass the filename

CHRISTIE

#CppKoan

There's a function for that

What's it called?

Guess

readConfigFile()?

Pass the filename

What's the path?

CHRISTIE

#CppKoan

There's a function for that

What's it called?

Guess

readConfigFile()?

Pass the filename

What's the path?

There's a function for that too

CHRISTIE

#CppKoan

There's a function for that

What's it called?

Guess

readConfigFile()?

Pass the filename

What's the path?

There's a function for that too

Just for the config path?

CHRISTIE

#CppKoan

There's a function for that

What's it called?

Guess

readConfigFile()?

Pass the filename

What's the path?

There's a function for that too

Just for the config path?

Yes

CHRISTIE

#CppKoan

There's a function for that

What's it called?

Guess

readConfigFile()?

Pass the filename

What's the path?

There's a function for that too

Just for the config path?

Yes

So... readConfigFile(getConfigFilepath())?

#CppKoan

There's a function for that

What's it called?

Guess

readConfigFile()?

Pass the filename

What's the path?

There's a function for that too

Just for the config path?

Yes

So... readConfigFile(getConfigFilepath())?

Yes, the code is true

#CppKoan

There's a function for that

What's it called?

Guess

readConfigFile()?

Pass the filename

What's the path?

There's a function for that too

Just for the config path?

Yes

So... readConfigFile(getConfigFilepath())?

Yes, the code is true

It didn't compile.

But Sensai was gone.

#CppKoan

Functions
are answers to questions.

There's a function for that

What's it called?

Guess

readConfigFile()?

Pass the filename

What's the path?

There's a function for that too

Just for the config path?

Yes

So... readConfigFile(getConfigFilepath())?

Yes, the code is true

It didn't compile.

But Master was gone.

#CppKoan

There's a function for that

What's it called?

Guess

readConfigFile()?

Pass the filename

What's the path?

There's a function for that too

Just for the config path?

Yes

So... readConfigFile(getConfigFilepath())?

Yes, the code is true

It didn't compile.

But Master was gone.

Functions
are answers to questions.

Write the functions
you want to see in the world.

Functions
are answers to questions.

Write the functions
you want to see in the world.

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // check for north OTHERWISE …
  if (isFacingNorth(proj))
    …

};
```

```
bool isFacingNorth(Projector proj)
{
  Degrees tolerance = Degrees(5.0);
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  auto dir = proj.getPose().orientation();
  if (dir.yaw() < Degrees(tolerance)
   || dir.yaw() > Degrees(360)-tolerance)
     return true;
   return false;
};
```

CHRISTIE

Functions
are answers to questions.

Write the functions
you want to see in the world.

*Code **top-down** on the way down*

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  proj.showPattern(delta) …
  …

  // check for north OTHERWISE …
  if (isFacingNorth(proj))
    …

};
```

```
bool isFacingNorth(Projector proj)
{
  Degrees tolerance = Degrees(5.0);
  if (proj.serialNumber() < 12345678)
     tolerance *= 2;
  auto dir = proj.getPose().orientation();
  if (dir.yaw() < Degrees(tolerance)
   || dir.yaw() > Degrees(360)-tolerance)
     return true;
   return false;
};
```

CHRISTIE

Master, I am honoured by your visit…

Master, I am honoured by your visit.

I was out walking my dog. Your function, why does it take BigCommonStruct instead of just x and y?

Master, I am honoured by your visit.

I was out walking my dog. Your function, why does it take BigCommonStruct instead of just x and y?

BCS is where we keep x and y, master.

Master, I am honoured by your visit.

I was out walking my dog. Your function, why does it take BigCommonStruct instead of just x and y?

BCS is where we keep x and y, master.

<Bark>

Master, I am honoured by your visit.

I was out walking my dog. Your function, why does it take BigCommonStruct instead of just x and y?

BCS is where we keep x and y, master.

<Bark>

He's hungry.

Master, I am honoured by your visit.

I was out walking my dog. Your function, why does it take BigCommonStruct instead of just x and y?

BCS is where we keep x and y, master.

<Bark>

He's hungry.

Shall I prepare him food?

Master, I am honoured by your visit.

I was out walking my dog. Your function, why does it take BigCommonStruct instead of just x and y?

BCS is where we keep x and y, master.

<Bark>

He's hungry.

Shall I prepare him food?

No, just open the fridge, let him take what he wants.

- Ancient C++ Koan

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj))
    …
};
```

```
bool isFacingNorth(Projector proj)
{
   auto dir = proj.orientation();
   auto tolerance = proj…

   ... math on dir …
   if (… < tolerance …)
      return true;
   return false;
};
```

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj))
    …
};
```

```
bool isFacingNorth(Projector proj)
{
   auto dir = proj.orientation();
   auto tolerance = proj…

   proj.showVideo(…)

   ... math on dir …
   if (… < tolerance …)
      return true;
   return false;
}
```

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj))
    …
};
```

```
bool isFacingNorth(Projector proj)
{
   auto dir = proj.orientation();
   auto tolerance = proj…

   proj.showVideo(…)

   ... math on dir …
   if (… < tolerance …)
      return true;
   return false;
```

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj))
    …
};
```


```
bool isFacingNorth(Projector proj)
{
   auto dir = proj.orientation();
   auto tolerance = proj…

   proj.showVideo(…)

   ... math on dir …
   if (… < tolerance …)
      return true;
   return false;
```

CLOSE. AT. HAND.

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj))
    …
};
```

```
bool isFacingNorth(Projector proj)
{
   auto dir = proj.orientation();
   auto tolerance = proj…

   proj.getMODEL()->getThing()->etc…

   ... math on dir …
   if (… < tolerance …)
      return true;
   return false;
}
```

CLOSE. AT. HAND.

CHRISTIE


AND YOU GET A MODEL POINTER!
AND YOU GET A MODEL POINTER!
imgflip.com

Complecting Made Easy

Tony Van Eerd

C++Now, May 2021

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj))
    …
};
```

```
bool isFacingNorth(Projector proj)
{
   auto dir = proj.orientation();
   auto tolerance = proj…

   proj.getMODEL()->getThing()->etc…

   ... math on dir …
   if (… < tolerance …)
      return true;
   return false;
}
```

CLOSE. AT. HAND.

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj.orientation(),5)
    …
};
```

```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
   ... math on dir …
   if (… < tolerance …)
       return true;
   return false;
};
```

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj.orientation(),5)
    …
};
```

```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
   ... math on dir …
   if (… < tolerance …)
       return true;
   return false;
};
```

Barrier to Complecting

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj.orientation(),5)
    …
};
```

```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
   ... math on dir …
   if (… < tolerance …)
       return true;
   return false;
};
```

Barrier to Complecting

CHRISTIE

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj))
    …
};
```

```
bool isFacingNorth(Projector proj)
{
   auto dir = proj.orientation();
   auto tol = getTolerance(proj);
   return isFacingNorth(dir, tol);
};
```

```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
   ... math on dir …
   if (… < tolerance …)
       return true;
   return false;
};
```

*Code **top-down** on the way down,*
*and **bottom-up** on the way back up.*

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj))
    …
};
```

```
bool isFacingNorth(Projector proj)
{
   auto dir = proj.orientation();
   auto tol = getTolerance(proj);
   return isFacingNorth(dir, tol);
};
```

```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
   ... math on dir …
   if (… < tolerance …)
       return true;
   return false;
};
```

*Code **top-down** on the way down*

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj))
    …
};
```

```
bool isFacingNorth(Projector proj)
{
   auto dir = proj.orientation();
   auto tolerance = proj…

   proj.showVideo(…)

   ... math on dir …
   if (… < tolerance …)
      return true;
   return false;
}
```

*Code **top-down** on the way down*

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj))
    …
};
```

```
bool isFacingNorth(Projector proj)
{
   auto dir = proj.orientation();
   auto tolerance = proj…

   proj.showVideo(…)

   ... math on dir …
   if (… < tolerance …)
      return true;
   return false;
```

CHRISTIE

*Code **top-down** on the way down,*
*and **bottom-up** on the way back up.*

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj))
    …
};
```

```
bool isFacingNorth(Projector proj)
{
   auto dir = proj.orientation();
   auto tol = getTolerance(proj);
   return isFacingNorth(dir, tol);
};
```

```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
   ... math on dir …
   if (… < tolerance …)
       return true;
   return false;
};
```

*Code **top-down** on the way down,*
*and **bottom-up** on the way back up.*

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj))
    …
};
```

KYSS:
Keep Your Stuff Separate

```
bool isFacingNorth(Projector proj)
{
   auto dir = proj.orientation();
   auto tol = getTolerance(proj);
   return isFacingNorth(dir, tol);
};
```

```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
   ... math on dir …
   if (… < tolerance …)
       return true;
   return false;
};
```

Value Oriented Programming, Part 1: Functions

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj))
    …
};
```

```
bool isFacingNorth(Projector * proj)
{
   auto dir = proj.orientation();
   auto tol = getTolerance(proj);
   return isFacingNorth(dir, tol);
};
```

```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
   ... math on dir …
   if (… < tolerance …)
       return true;
   return false;
};
```

Value Oriented Programming, Part 1: Functions

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj))
    …
};
```

Testability!

Safety!

```
bool isFacingNorth(Projector * proj)
{
   auto dir = proj.orientation();
   auto tol = getTolerance(proj);
   return isFacingNorth(dir, tol);
};
```

```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
   ... math on dir …
   if (… < tolerance …)
       return true;
   return false;
};
```

Value Oriented Programming, Part 1: Functions


```
bool isFacingNorth(Projector * proj)
{
    auto dir = proj.orientation();
    auto tol = getTolerance(proj);
    return isFacingNorth(dir, tol);
};
```

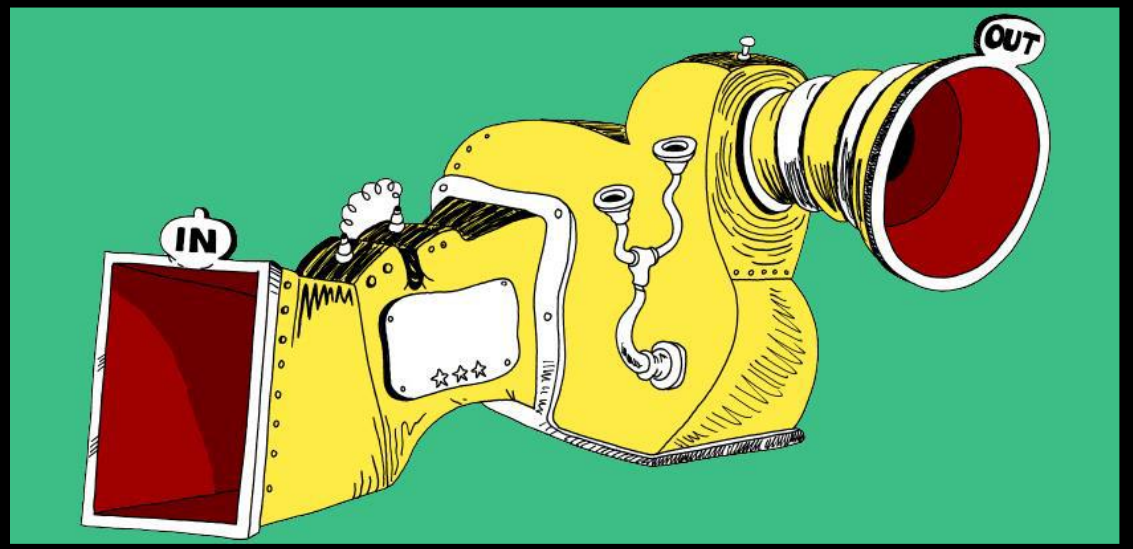
```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
    ... math on dir …
    if (… < tolerance …)
        return true;
    return false;
};
```

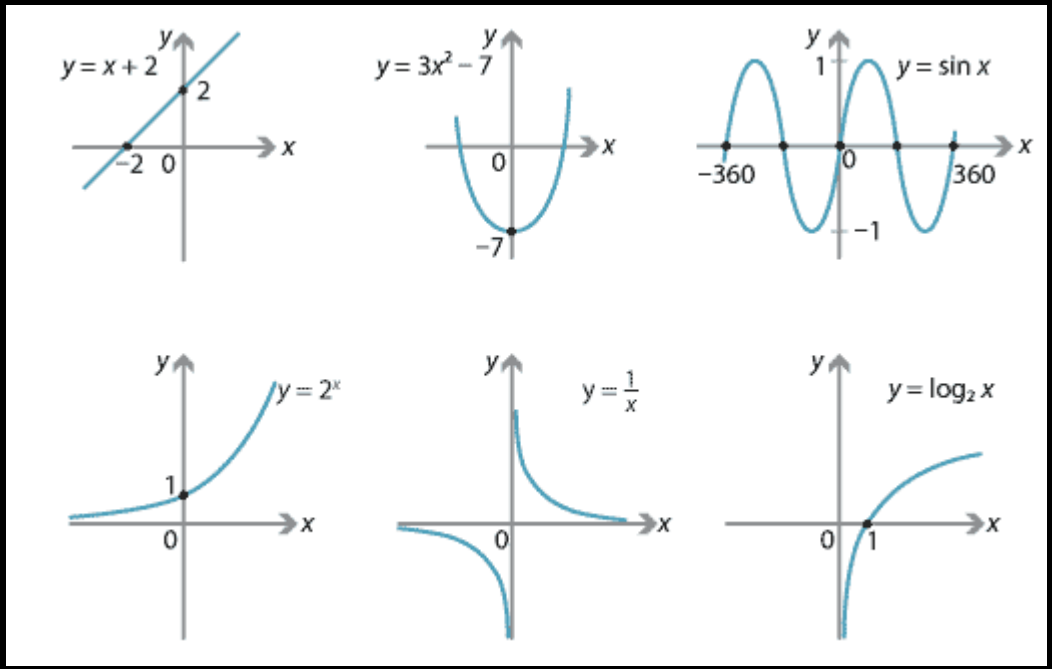
Value Oriented Programming, Part 1: Functions


```
bool isFacingNorth(Projector proj)
{
    auto dir = proj.orientation();
    auto tol = getTolerance(proj);
    return isFacingNorth(dir, tol);
};
```

```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
    ... math on dir …
    if (… < tolerance …)
        return true;
    return false;
};
```

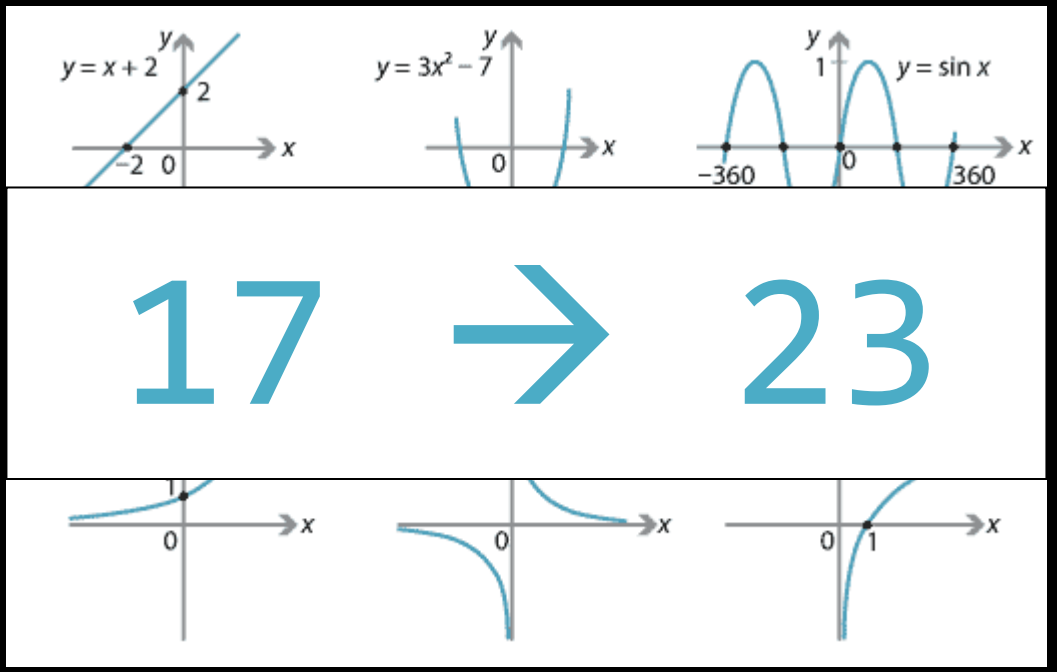
Value Oriented Programming, Part 1: **Functions**





Value Oriented Programming, Part 1: **Functions**

Takes input, MODIFIES it, produces output

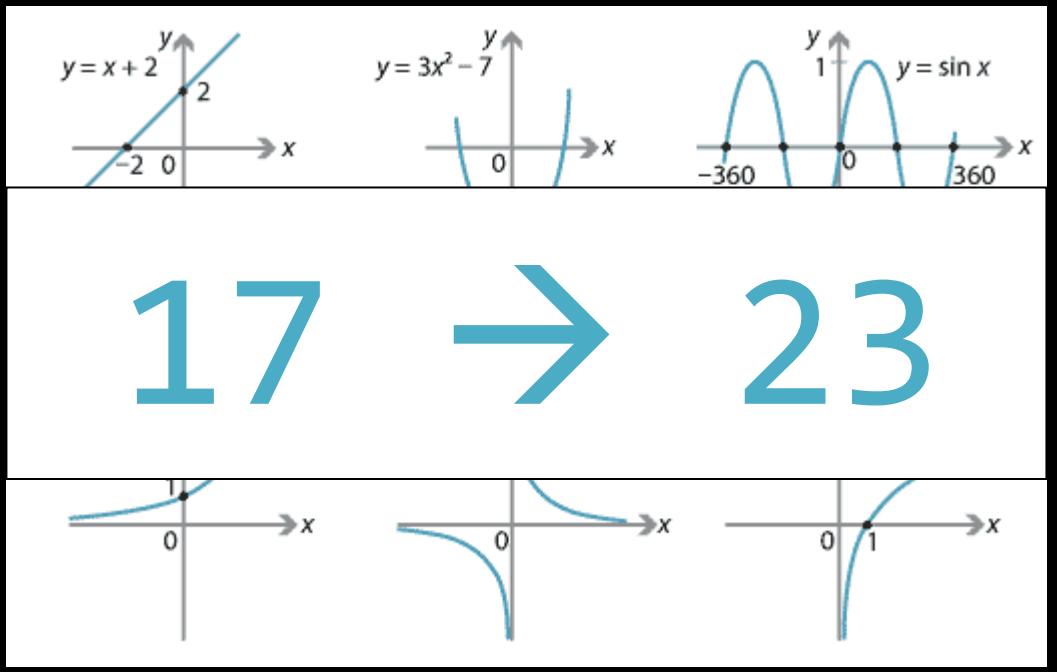


Takes input, produces output

*Separate **calculating** from **doing***

- Grokking Simplicity by Eric Normand

Takes input, MODIFIES it, produces output



Takes input, produces output

*Separate **calculating** from **doing***

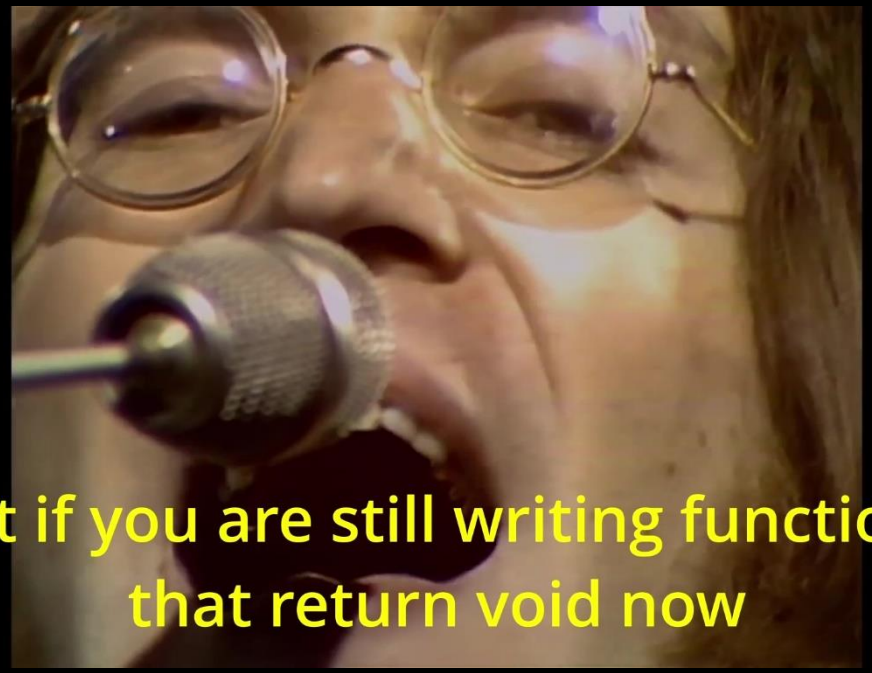
- Grokking Simplicity by Eric Normand


```
void StarOn(Sneetch & sneetch)
{
  // measure sneetch…
  // find belly…
  // calculate star position(s)…
  // apply star(s) to sneetch
};
```

Takes input, MODIFIES it, produces output

*Separate **calculating** from **doing***

- Grokking Simplicity by Eric Normand




```
void StarOn(Sneetch & sneetch)
{
  // measure sneetch…
  // find belly…
  // calculate star position(s)…
  // apply star(s) to sneetch
};
```

Takes input, MODIFIES it, produces output

*Separate **calculating** from **doing***

- Grokking Simplicity by Eric Normand


```
void StarOn(Sneetch & sneetch)
{
  // measure sneetch…
  // find belly…
  // calculate star position(s)…
  // apply star(s) to sneetch
};
```

Takes input, MODIFIES it, produces output

*Separate **calculating** from **doing***

- Grokking Simplicity by Eric Normand

*Returning **void** is a code smell*

```
void StarOn(Sneetch & sneetch)
{
  // [illegible] sneetch…
  // [illegible]…
  // calculate star position(s)…
  // apply star(s) to sneetch
};
```

Takes input, MODIFIES it, produces output

*Separate **calculating** from **doing***

- Grokking Simplicity by Eric Normand


```
void StarOn(Sneetch & sneetch)
{
  // measure sneetch…
  // find belly…
  // calculate star position(s)…
  // apply star(s) to sneetch
};
```

Takes input, MODIFIES it, produces output

*Separate **calculating** from **doing***

- Grokking Simplicity by Eric Normand


```
void StarOn(Sneetch & sneetch, int num)
{
  vector locs = calcStarPos(sneetch, num);
  applyStars(sneetch, locs);
};
```

Takes input, MODIFIES it, produces output

*Separate **calculating** from **doing***

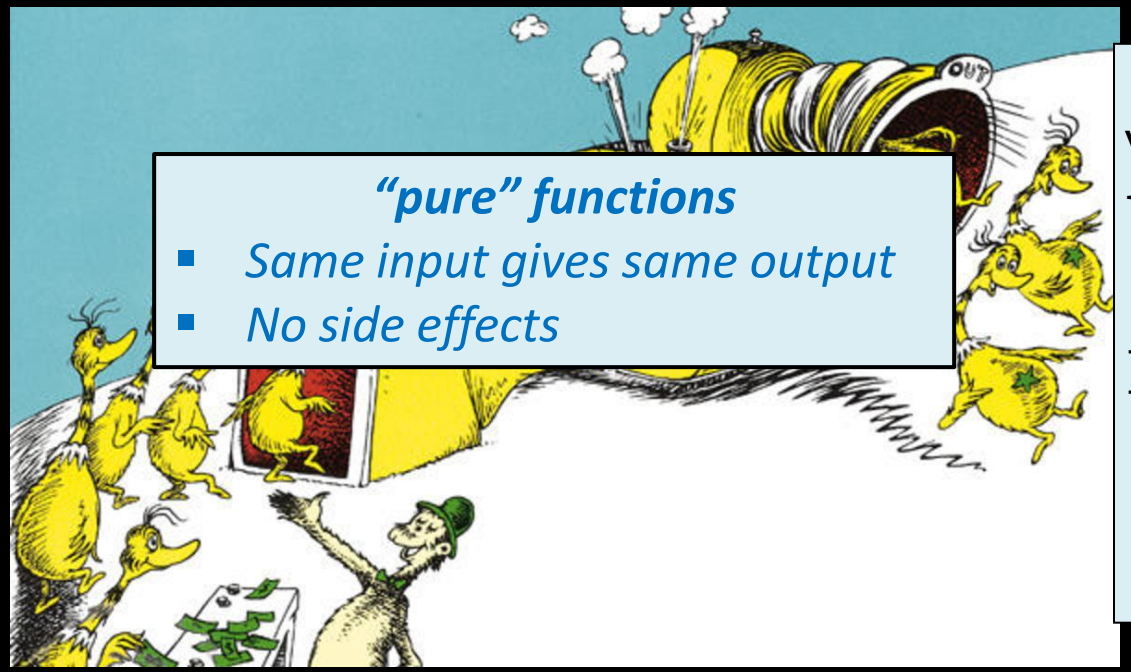
- Grokking Simplicity by Eric Normand


```
void StarOn(Sneetch & sneetch, int num)
{
  vector locs = calcStarPos(sneetch.geom(),
  applyStars(sneetch, locs);
};
```

Takes input, MODIFIES it, produces output

*Separate **calculating** from **doing***

- Grokking Simplicity by Eric Normand



"pure" functions

- *Same input gives same output*
- *No side effects*

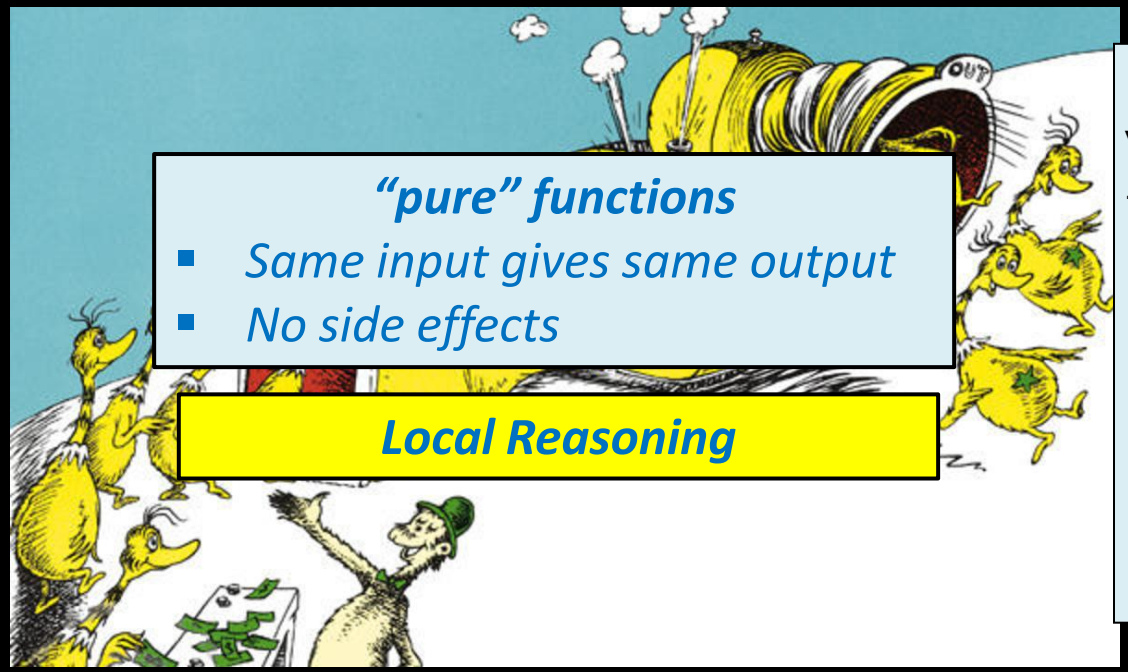
Takes input, MODIFIES it, produces output

```
void StarOn(Sneetch & sneetch, int num)
{
  vector locs = calcStarPos(sneetch, num);
  applyStars(sneetch, locs);
};
```

Takes input, produces output

*Separate **calculating** from **doing***

- Grokking Simplicity by Eric Normand



"pure" functions

- *Same input gives same output*
- *No side effects*

Local Reasoning

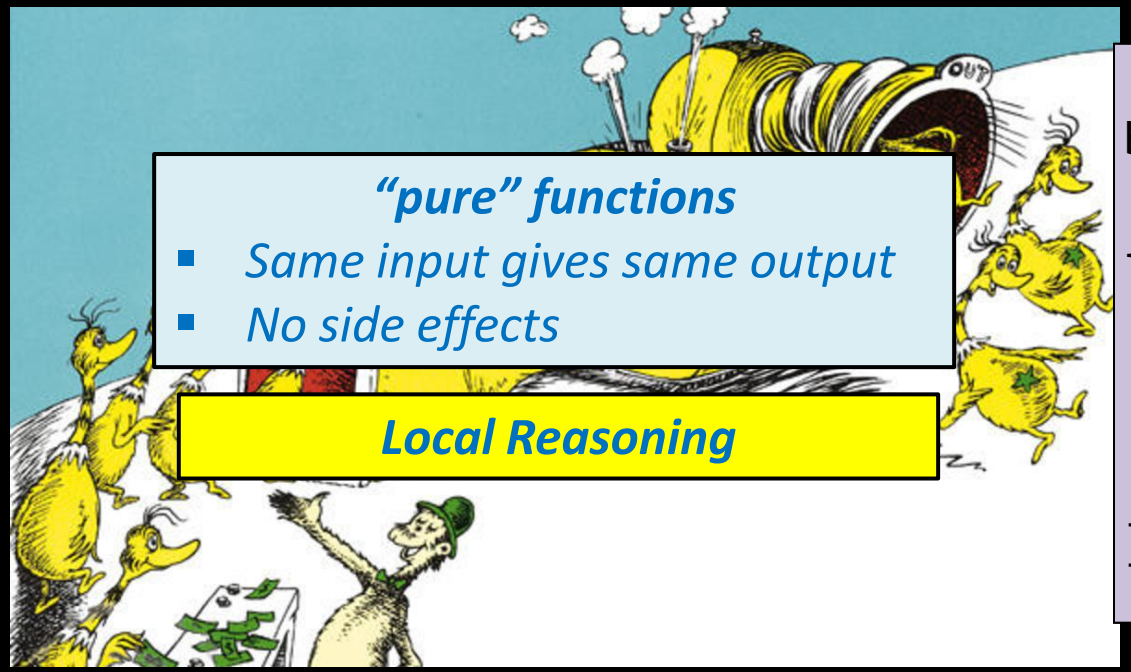
```
void StarOn(Sneetch & sneetch, int num)
{
  vector locs = calcStarPos(sneetch, num);
  applyStars(sneetch, locs);
};
```

Takes input, MODIFIES it, produces output

Takes input, produces output

*Separate **calculating** from **doing***

- Grokking Simplicity by Eric Normand




```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
    ... math on dir …
    if (… < tolerance …)
        return true;
    return false;
};
```

Takes input, MODIFIES it, produces output

Takes input, produces output

David John Wheeler

(previously worked with Bjarne Stroustrup)

Recent breakthrough research

David John Wheeler

(previously worked w

Recent breakthrough

THE USE OF SUB-ROUTINES IN PROGRAMMES

D. J. Wheeler

Cambridge & Illinois Universities

A sub-routine may perhaps best be described as a self-contained part of a programme, which is capable of being used in different programmes. It is an entity of its own within a programme. There is no necessity to compose a programme of a set of distinct sub-routines; for the programme can be written as a complete unit, with no divisions into smaller parts. However it is usually advantageous to arrange that a programme is comprised of a set of sub-routines, some of which have been made specially for the particular programme while others are available from a 'library' of standard sub-routines. The reasons for this will be discussed below.

When a programme has been made from a set of sub-routines the breakdown of the code is more complete than it would otherwise be. This allows the coder to concentrate on one section of a programme at a time without the overall detailed programme continually intruding. Thus the sub-routines can be more easily coded and the tested in isolation from the rest of the programme. When the entire programme has to be tested it is with the foreknowledge that the incidence of mistakes in the sub-routines is zero (or at least one order of magnitude below that of the untested portions of the programme!)

If library sub-routines exist for the major part of a code then the task of constructing the easier to use a sub-routine which will meet the specifications with a small amount of manipulation than to make one specially for the purpose.

It should be pointed out that the preparation of a library sub-routine requires a considerable amount of work. This is much greater than the effort merely required to code the sub-routine in its simplest possible form. It will usually be necessary to code it in the library standard form and this may detract from its efficiency in time and space. It may be desirable to code it in such a manner that the operation is generalized to some extent. However, even after it has been coded and tested there still remains the considerable task of writing a description so that people not acquainted with the interior coding can nevertheless use it easily. This last task may be the most difficult.

Besides the organization of the individual sub-routines there remains the method of the general organization of the library. How are the sub-routines going to be stored? Are they going to be stored on punched paper tape or are they going to be available in the auxiliary store of the machine? Usually it will be found that it is not possible to write the sub-routines such that they may be put into arbitrary positions in the store-although in certain machines this is now possible. Usually some translation process will have to be

David John Wheeler

(previously worked w

Recent breakthrough

routines can be more easily coded and the tested in isolation from the rest of the programme. When the entire programme has to be tested it is with the foreknowledge that the incidence of mistakes in the sub-routines is zero (or at least one order of magnitude below that of the untested portions of the programme!)

If library sub-routines exist for the major part of a code then the task of constructing the remaining part of the programme is naturally very much less than if the code had to be written from the very beginning. However, one will rarely have available sub-routines to do exactly what is required and thus a certain amount of manipulation may be necessary before a given sub-routine can be used. Even so,it is usually far

organization of the library. How are the sub-routines going to be stored? Are they going to be stored on punched paper tape or are they going to be available in the auxiliary store of the machine? Usually it will be found that it is not possible to write the sub-routines such that they may be put into arbitrary positions in the store-although in certain machines this is now possible. Usually some translation process will have to be arranged so that an invariant form of sub-routine stored on some medium such as paper tape can be translated to the form required in a particular application. This translation is possible because fixed rules can be set up for adjusting a sub-routine so that it becomes correct in the set of locations in which it is put and used.

One next considers the methods by which sub-routines can be used. There are a number of different ways of transferring control to sub-routines and arranging that control is returned to the appropriate point to which it is required. One of the simpler methods was that used for the closed sub-routines of the EDSAC in which it was arranged that when the sub-routine had performed its part of the computation then control was returned to a point in the main programme immediately after the orders which had called it into use. This has been described in detail by Goldstine. This perhaps facilitates thinking of a sub-

'orders' that are obeyed are identical with those of the machine. However, the interpretive routine retains control and so it is possible to print out extra information about the course of the programme. This extra information makes it possible to follow the meanderings of the program in detail thus helping to locate the errors of a programme. This is not a good method of finding errors in programmes as it takes a long time and the programmers knowledge of the programme is not utilized - as it should be - in tracing the fault. However, it is a useful last resort and can quite often give out information about a code which would be difficult to find

David John Wheeler

(previously worked w

Recent breakthrough

EDSAC in which it was arranged that when the sub-routine had performed its part of the computation then control was returned to a point in the main programme immediately after the orders which had called it into use. This has been described in detail by Goldstine. This perhaps facilitates thinking of a sub-routine as an 'order' of the machine although it is usually of a more complicated kind than that wired in the circuits of the machine.

A second more interesting type of sub-routine is an interpretive routine. In this type of routine it is arranged that a sequence of operations is performed each time the sub-routine is called into action, each operation being determined by one parameter or 'order' in a list of such 'orders'. This type of sub-routine is particularly useful for coding certain special types of arithmetic for the machine, for example, floating point arithmetic in which numbers are expressed as $\mathbf{X} \times 10^{p}$. Thus the sub-routine executes the 'orders' in the list in a similar fashion to the way that the machine obeys ordinary orders. However, the orders that it does are determined by the parts of the sub-routine, and so can be made to do any kind of operation or arithmetic.

One extension of an interpretive routine is a checking routine which is so arranged that the

of a programme. This is not a good method of finding errors in programmes as it takes a long time and the programmers knowledge of the programme is not utilized - as it should be - in tracing the fault. However, it is a useful last resort and can quite often give out information about a code which would be difficult to find in any other way.

Sub-routines seem to have two distinct uses in programmes. The first and most obvious use is for the evaluation of functions, a simple example being the evaluation of sine x given x. The second use is for the organization of processes such as the integration of a function given f(x). This second type requires more consideration to make it useful and general. For instance how should f(x) be specified for the sub-routine? One obvious and useful way is to allow the integrating sub-routine access to an auxiliary sub-routine which is capable of evaluating f(x).

The above remarks may be summarized by saying sub-routines are very useful-although not absolutely necessary-and that the prime objectives to be born in mind when constructing them are simplicity of use, correctness of codes and accuracy of description. All complexities should-if possible -be buried out of sight.

David John Wheeler

(previously worked w

Recent breakthrough

"self-contained part of a programme"

"an entity of its own"

THE USE OF SUB-ROUTINES IN PROGRAMMES

D. J. Wheeler

Cambridge & Illinois Universities

A sub-routine may perhaps best be described as a self-contained part of a programme, which is capable of being used in different programmes. It is an entity of its own within a programme. There is no necessity to compose a programme of a set of distinct sub-routines; for the programme can be written as a complete unit, with no divisions into smaller parts. However it is usually advantageous to arrange that a programme is comprised of a set of sub-routines, some of which have been made specially for the particular programme while others are available from a 'library' of standard sub-routines. The reasons for this will be discussed below.

When a programme has been made from a set of sub-routines the breakdown of the code is more complete than it would otherwise be. This allows the coder to concentrate on one section of a programme at a time without the overall detailed programme continually intruding. Thus the sub-routines can be more easily coded and the tested in isolation from the rest of the programme. When the entire programme has to be tested it is with the foreknowledge that the incidence of mistakes in the sub-routines is zero (or at least one order of magnitude below that of the untested portions of the programme!)

If library sub-routines exist for the major part of a code then the task of constructing the easier to use a sub-routine which will meet the specifications with a small amount of manipulation than to make one specially for the purpose.

It should be pointed out that the preparation of a library sub-routine requires a considerable amount of work. This is much greater than the effort merely required to code the sub-routine in its simplest possible form. It will usually be necessary to code it in the library standard form and this may detract from its efficiency in time and space. It may be desirable to code it in such a manner that the operation is generalized to some extent. However, even after it has been coded and tested there still remains the considerable task of writing a description so that people not acquainted with the interior coding can nevertheless use it easily. This last task may be the most difficult.

Besides the organization of the individual sub-routines there remains the method of the general organization of the library. How are the sub-routines going to be stored? Are they going to be stored on punched paper tape or are they going to be available in the auxiliary store of the machine? Usually it will be found that it is not possible to write the sub-routines such that they may be put into arbitrary positions in the store-although in certain machines this is now possible. Usually some translation process will have to be

David John Wheeler

(previously worked w

Recent breakthrough

"allows the coder to concentrate on one section of a programme at a time without the overall detailed programme continually intruding"

THE USE OF SUB-ROUTINES IN PROGRAMMES

D. J. Wheeler

Cambridge & Illinois Universities

A sub-routine may perhaps best be described as a self-contained part of a programme, which is capable of being used in different programmes. It is an entity of its own within a programme. There is no necessity to compose a programme of a set of distinct sub-routines; for the programme can be written as a complete unit, with no divisions into smaller parts. However it is usually advantageous to arrange that a programme is comprised of a set of sub-routines, some of which have been made specially for the particular programme while others are available from a 'library' of standard sub-routines. The reasons for this will be discussed below.

When a programme has been made from a set of sub-routines the breakdown of the code is more complete than it would otherwise be. This allows the coder to concentrate on one section of a programme at a time without the overall detailed programme continually intruding. Thus the sub-routines can be more easily coded and the tested in isolation from the rest of the programme. When the entire programme has to be tested it is with the foreknowledge that the incidence of mistakes in the sub-routines is zero (or at least one order of magnitude below that of the untested portions of the programme!)

If library sub-routines exist for the major part of a code then the task of constructing the easier to use a sub-routine which will meet the specifications with a small amount of manipulation than to make one specially for the purpose.

It should be pointed out that the preparation of a library sub-routine requires a considerable amount of work. This is much greater than the effort merely required to code the sub-routine in its simplest possible form. It will usually be necessary to code it in the library standard form and this may detract from its efficiency in time and space. It may be desirable to code it in such a manner that the operation is generalized to some extent. However, even after it has been coded and tested there still remains the considerable task of writing a description so that people not acquainted with the interior coding can nevertheless use it easily. This last task may be the most difficult.

Besides the organization of the individual sub-routines there remains the method of the general organization of the library. How are the sub-routines going to be stored? Are they going to be stored on punched paper tape or are they going to be available in the auxiliary store of the machine? Usually it will be found that it is not possible to write the sub-routines such that they may be put into arbitrary positions in the store-although in certain machines this is now possible. Usually some translation process will have to be

David John Wheeler

(previously worked w

Recent breakthrough

"Thus the subroutines can be more **easily coded and tested in isolation** from the rest of the programme."

THE USE OF SUB-ROUTINES IN PROGRAMMES

D. J. Wheeler

Cambridge & Illinois Universities

A sub-routine may perhaps best be described as a self-contained part of a programme, which is capable of being used in different programmes. It is an entity of its own within a programme. There is no necessity to compose a programme of a set of distinct sub-routines; for the programme can be written as a complete unit, with no divisions into smaller parts. However it is usually advantageous to arrange that a programme is comprised of a set of sub-routines, some of which have been made specially for the particular programme while others are available from a 'library' of standard sub-routines. The reasons for this will be discussed below.

When a programme has been made from a set of sub-routines the breakdown of the code is more complete than it would otherwise be. This allows the coder to concentrate on one section of a programme at a time without the overall detailed programme continually intruding. Thus the sub-routines can be more easily coded and the tested in isolation from the rest of the programme. When the entire programme has to be tested it is with the foreknowledge that the incidence of mistakes in the sub-routines is zero (or at least one order of magnitude below that of the untested portions of the programme!)

If library sub-routines exist for the major part of a code then the task of constructing the easier to use a sub-routine which will meet the specifications with a small amount of manipulation than to make one specially for the purpose.

It should be pointed out that the preparation of a library sub-routine requires a considerable amount of work. This is much greater than the effort merely required to code the sub-routine in its simplest possible form. It will usually be necessary to code it in the library standard form and this may detract from its efficiency in time and space. It may be desirable to code it in such a manner that the operation is generalized to some extent. However, even after it has been coded and tested there still remains the considerable task of writing a description so that people not acquainted with the interior coding can nevertheless use it easily. This last task may be the most difficult.

Besides the organization of the individual sub-routines there remains the method of the general organization of the library. How are the sub-routines going to be stored? Are they going to be stored on punched paper tape or are they going to be available in the auxiliary store of the machine? Usually it will be found that it is not possible to write the sub-routines such that they may be put into arbitrary positions in the store-although in certain machines this is now possible. Usually some translation process will have to be

David John Wheeler - Subroutines

(Bjarne Stroustrup's thesis supervisor)

Recent breakthrough research - **1951**

THE USE OF SUB-ROUTINES IN PROGRAMMES

D. J. Wheeler

Cambridge & Illinois Universities

A sub-routine may perhaps best be described as a self-contained part of a programme, which is capable of being used in different programmes. It is an entity of its own within a programme. There is no necessity to compose a programme of a set of distinct sub-routines; for the programme can be written as a complete unit, with no divisions into smaller parts. However it is usually advantageous to arrange that a programme is comprised of a set of sub-routines, some of which have been made specially for the particular programme while others are available from a 'library' of standard sub-routines. The reasons for this will be discussed below.

When a programme has been made from a set of sub-routines the breakdown of the code is more complete than it would otherwise be. This allows the coder to concentrate on one section of a pro-

easier to use a sub-routine which will meet the specifications with a small amount of manipulation than to make one specially for the purpose.

It should be pointed out that the preparation of a library sub-routine requires a considerable amount of work. This is much greater than the effort merely required to code the sub-routine in its simplest possible form. It will usually be necessary to code it in the library standard form and this may detract from its efficiency in time and space. It may be desirable to code it in such a manner that the operation is generalized to some extent. However, even after it has been coded and tested there still remains the considerable task of writing a description so that people not acquainted with the interior coding can nevertheless use it easily. This last task may be the most difficult.

David John Wheeler - Subroutines

(Bjarne Stroustrup's thesis supervisor)

Recent breakthrough research - **1951**

Functions are the fundamental building block of programming

THE USE OF SUB-ROUTINES IN PROGRAMMES

D. J. Wheeler

Cambridge & Illinois Universities

A sub-routine may perhaps best be described as a self-contained part of a programme, which is capable of being used in different programmes. It is an entity of its own within a programme. There is no necessity to compose a programme of a set of distinct sub-routines; for the programme can be written as a complete unit, with no divisions into smaller parts. However it is usually advantageous to arrange that a programme is comprised of a set of sub-routines, some of which have been made specially for the particular programme while others are available from a 'library' of standard sub-routines. The reasons for this will be discussed below.

When a programme has been made from a set of sub-routines the breakdown of the code is more complete than it would otherwise be. This allows the coder to concentrate on one section of a pro-

easier to use a sub-routine which will meet the specifications with a small amount of manipulation than to make one specially for the purpose.

It should be pointed out that the preparation of a library sub-routine requires a considerable amount of work. This is much greater than the effort merely required to code the sub-routine in its simplest possible form. It will usually be necessary to code it in the library standard form and this may detract from its efficiency in time and space. It may be desirable to code it in such a manner that the operation is generalized to some extent. However, even after it has been coded and tested there still remains the considerable task of writing a description so that people not acquainted with the interior coding can nevertheless use it easily. This last task may be the most difficult.


The Meaning of Life…


But when you talk about
member functions


```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
    ... math on dir …
    if (… < tolerance …)
        return true;
    return false;
};
```

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (proj.isFacingNorth())
    …
};
```

```
b
{
```

```
class Projector
{
    …
    bool isFacingNorth() const;
    …
};
```

```
}
```

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (proj.isFacingNorth())
    …
  if (cam.isFacingNorth())
};
```

```
b
{
```

```
class Projector
{
    …
    bool isFacingNorth() const;
    …
};
```

```
}
```

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (proj.isFacingNorth())
    …
  if (cam.isFacingNorth())
};
```

```
class Projector
{
    …
    bool isFacingNorth() const;
    …
};
```

```
class Camera
{
    …
    bool isFacingNorth() const;
    …
};
```

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (proj.isFacingNorth())
    …
  if (cam.isFacingNorth())
};
```

```
class Device
{
   virtual bool isFacingNorth() const;
};
```

```
class Projector : Device
{
    …
    bool isFacingNorth() const override;
    …
};
```

```
class Camera : Device
{
    …
    bool isFacingNorth() const override;
    …
};
```

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (proj.isFacingN
    …
  if (cam.isFacingNo
};
```

```
class MediaServer : Device
{
    …
    ???
};
```

```
class Device
{
   virtual bool isFacingNorth() const;
};
```

```
class Projector : Device
{
    …
    bool isFacingNorth() const override;
```

```
    …
};
```

```
era : Device
```

```
    bool isFacingNorth() const override;
    …
};
```

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (proj.isFacingNorth())
    …
  if (cam.isFacingNorth())
};
```

```
class DirectionalDevice : Device
{
   virtual bool isFacingNorth() const;
};
```

```
class Projector : DirectionalDevice
{
    …
    bool isFacingNorth() const override;
    …
};
```

```
class Camera : DirectionalDevice
{
    …
    bool isFacingNorth() const override;
    …
};
```

```
class DirectionalDevice : Device
{
   virtual bool isFacingNorth() const;
};
```

```
class Projector : DirectionalDevice
…
North() const override;
```

```
void very
{
  …
  Project
  …
  if (pro
    …
  if (cam
};
```

```
rectionalDevice
…
bool isFacingNorth() const override;
…
};
```

Share Implementation?

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (proj.isFacingNorth())
    …
  if (cam.isFacingNorth())
};
```

```
class DirectionalDevice : Device
{
   virtual bool isFacingNorth() const;
};
```

```
class Projector : DirectionalDevice
{
    …
    bool isFacingNorth() const override;
    …
};
```

```
class Camera : DirectionalDevice
{
    …
    bool isFacingNorth() const override;
    …
};
```

Share Implementation?

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj.orientation,tol))
    …
  if (isFacingNorth(cam.orientation,tol))
};
```

```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
   ... math on dir …
   if (… < tolerance …)
       return true;
   return false;
};
```

Share Implementation?

Functions are for sharing

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj.orientation,tol))
    …
  if (isFacingNorth(cam.orientation,tol))
};
```

```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
   ... math on dir …
   if (… < tolerance …)
       return true;
   return false;
};
```

```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (proj.orientation.isFacingNorth(5))
    …
};
```

```
class Orientation
{
    …
    bool isFacingNorth(double tol) const;
    …
};
```

```
bool isFacingNorth(Orientation, double);
```

```
class Orientation
{
    …
    bool isFacingNorth(double tol) const;
    …
};
```

“Isomorphic”

```
bool isFacingNorth(Orientation, double);
```

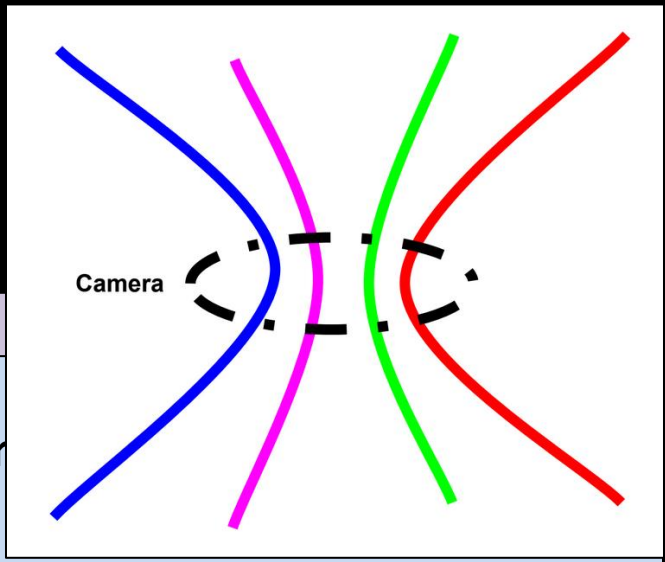
```
class Orientation
{
    …
    bool isFacingNorth(double tol) const;
    …
};
```

```
bool isFacingN                                                    le tol) const;
```

Classes are made of Velcro

```
bool isFacingN                                                    le tol) const;
```

```
class Camera
{
  CameraId id;
  DevicePath path;

  int bitdepth;
  Resolution resolution;
  int gain;
  int exposure; // units?

  Pose pose;
  Calibration * calibration;

  int binarizationThreshold;
  int sharpness; // UI only
  int multicamEdgeThreshold;

  Image baseImage;
  Image negativeImage;
  Image maskToUse;
  Image reverseMask;

  Device * device;
  INativeCamera * camera;
```



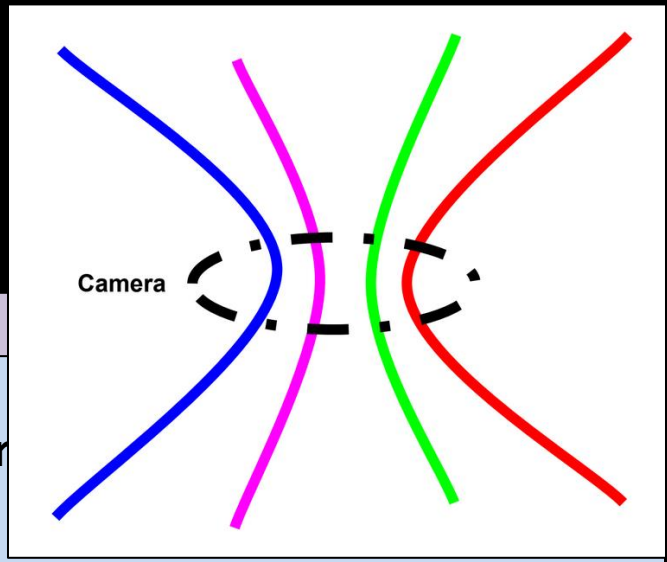
bool isFacingNorth(Orienta

ation

acingNorth(double tol) const;

Classes are made of Velcro

Classes are a nexus of Complecting




```
bool isFacingNorth(Orientation, double);
```

```
class Orientation
{
    …
    bool isFacingNorth(double tol) const;
    …
};
```

Classes are made of Velcro

Classes are a nexus of Complecting

```
bool isFacingNorth(Orientation, double);
```

```
class Orientation
{
    …
    bool isFacingNorth(double tol) const;
    …
};
```

Uses external Orientation API

Accesses internals

Good Code

- Begets good code
- Works

```
bool isFacingNorth(Orientation, double);
```

Uses external Orientation API

```
    …
    bool isFacingNorth(double tol) const;
    …
};
```

Accesses internals

Classes are made of Velcro

Classes are a nexus of Complecting

```
bool isFacingNorth(Orientation, double);
```

```
class Orientation
{
    …
    bool isFacingNorth(double tol) const;
    …
};
```

Uses external Orientation API

Accesses internals

“Invariants”

```
bool isFacingNorth(Orien                                    const;
```

Uses external Orie

https://www.aristeia.com/Papers/CUJ_Feb_2000.pdf

Scott Meyers

How Non-Member Functions Improve Encapsulation

When it comes to encapsulation, sometimes less is more.

I'll start with the punchline: If you're writing a function that can be implemented as either a member or as a non-friend non-member, you should prefer to implement it as a non-member function. That decision *increases* class encapsulation. When you think encapsulation, you should think non-member functions.

Surprised? Read on.

Background

When I wrote the first edition of *Effective C++* in 1991 [1], I examined the problem of determining where to declare a

1997 [2], I made no changes to this part of the book.

In 1998, however, I gave a presentation at Actel, where Arun Kundu observed that my algorithm dictated that functions should be member functions even when they could be implemented as non-members that used only C's public interface. Is that really what I meant, he asked me? In other words, if f could be implemented as a member function or a non-friend non-member function, did I really advocate making it a member function? I thought about it for a moment, and I decided that that was not what I meant. I therefore modified the algorithm to look like this [3]:

```
if (f needs to be virtual)
   make f a member function of C;
else if (f is operator>> or
         operator<<)
```

Encapsulation

Encapsulation is a *means*, not an end. There's nothing inherently desirable about encapsulation. Encapsulation is useful only because it yields other things in our software that we care about. In particular, it yields flexibility and robustness. Consider this struct, whose implementation I think we'll all agree is unencapsulated:

```
struct Point {
   int x, y;
};
```

The weakness of this struct is that it's not flexible in the face of change. Once clients started using this struct, it would, practically speaking, be very hard to change it; too much client code would be broken. If we later decided we want

How many string functions are there?

How many string functions are there?

```
string func(string s);
```

How many string functions are there?

```
string func(string s);
```

∞ ?

How many string functions are there?

```
string func(string s);
```

∞ ?

which ∞ ?

How many string functions are there?

```
string func(string s);
```

∞ ?

which ∞ ?

A lot.

How many string functions are there?

```
string func(string s);
```

∞ ?

which ∞ ?

Should they all be member functions?

Classes are made of Velcro

Classes are a nexus of Complecting

```
bool isFacingNorth(Orientation, double);
```

```
class Orientation
{
    …
    bool isFacingNorth(double tol) const;
    …
};
```

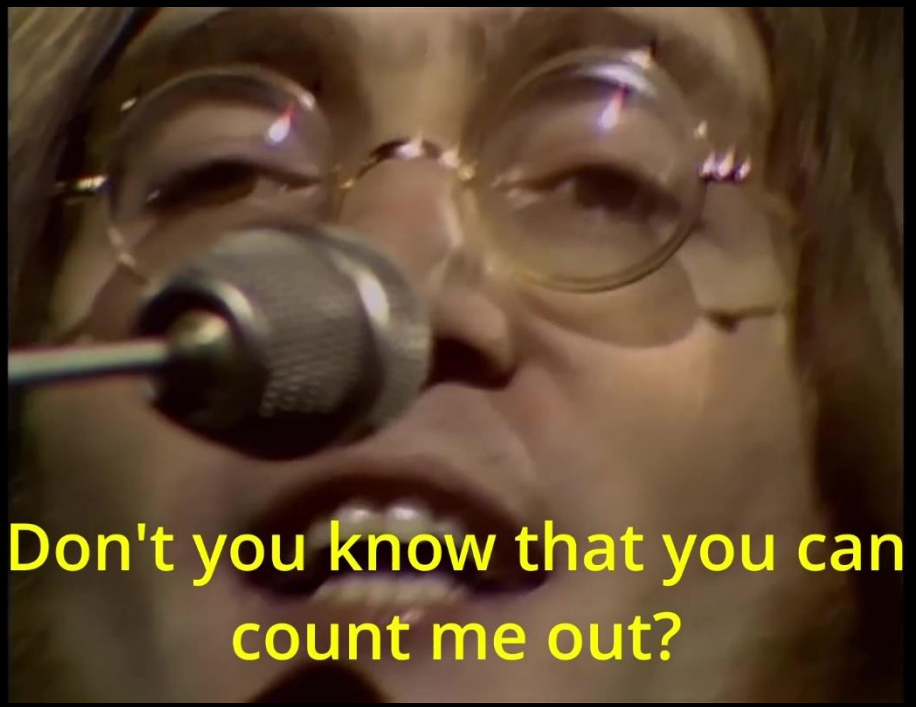
Classes are made of Velcro

Classes are a nexus of Complecting

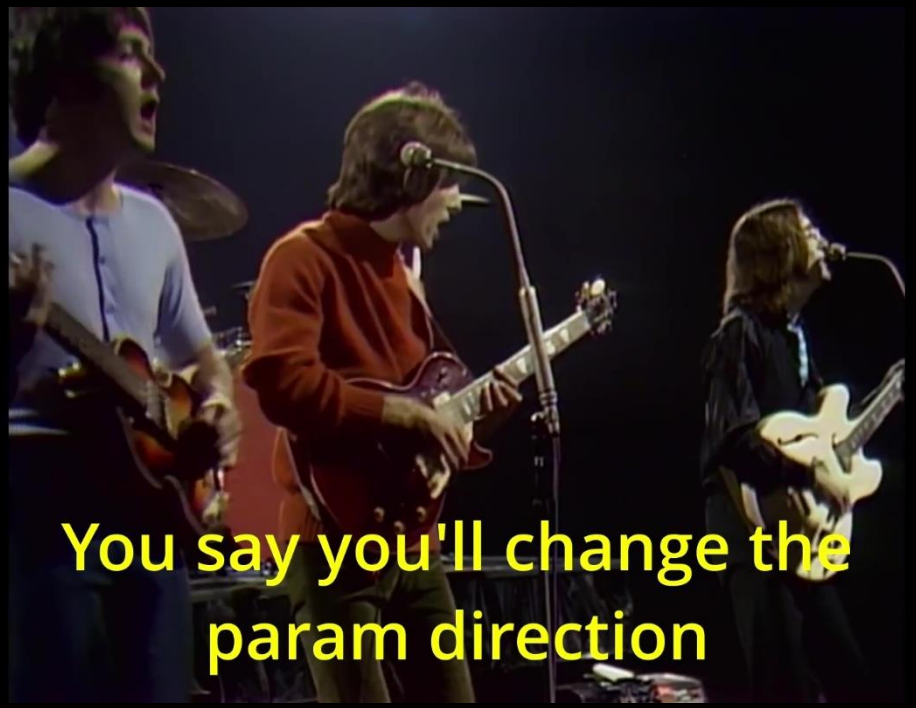
```
void veryImportantFunction()
{
  …
  Projector proj …
  …
  if (isFacingNorth(proj.orientation, 5))
    …
};
```

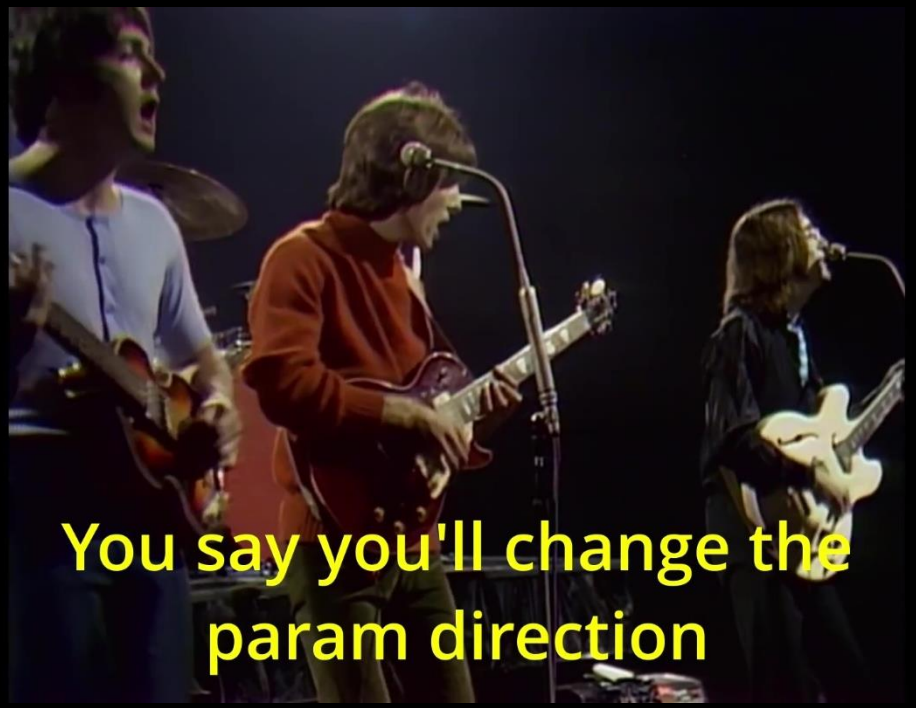
```
bool isFacingNorth(Orientation dir,
                   double tolerance)
{
   ... math on dir …
   if (… < tolerance …)
       return true;
   return false;
};
```


But when you talk about
member functions

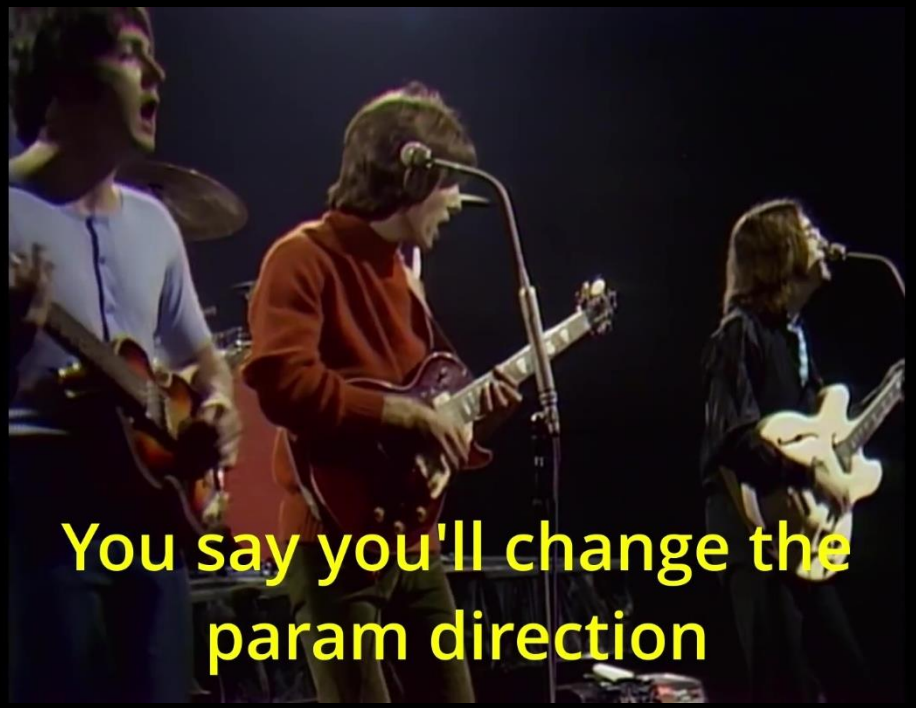

Don't you know that you can
count me out?

Classes are made of Velcro

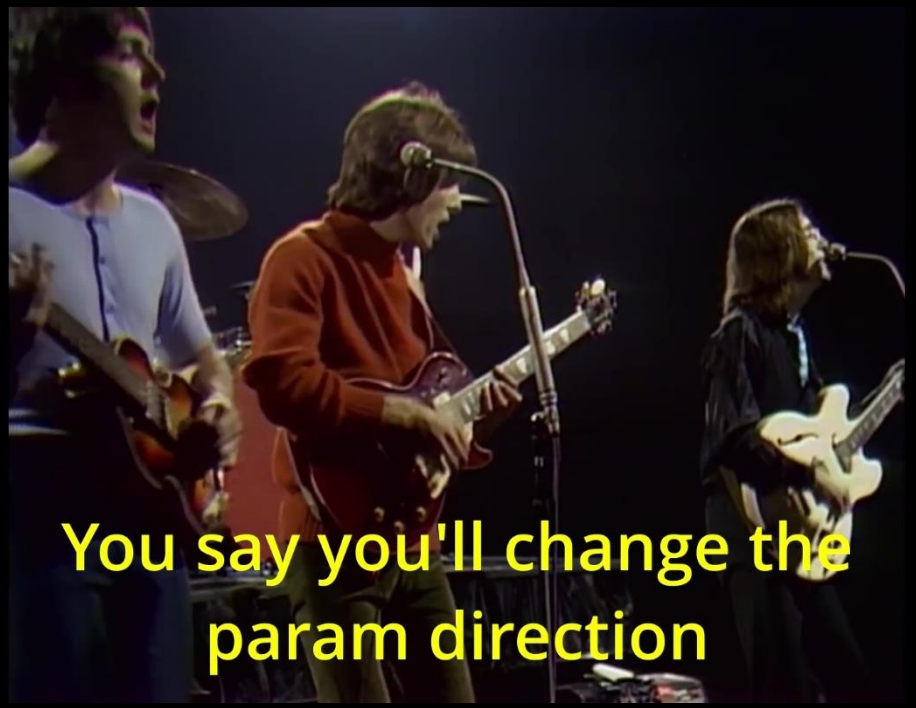

You say you'll change the
param direction



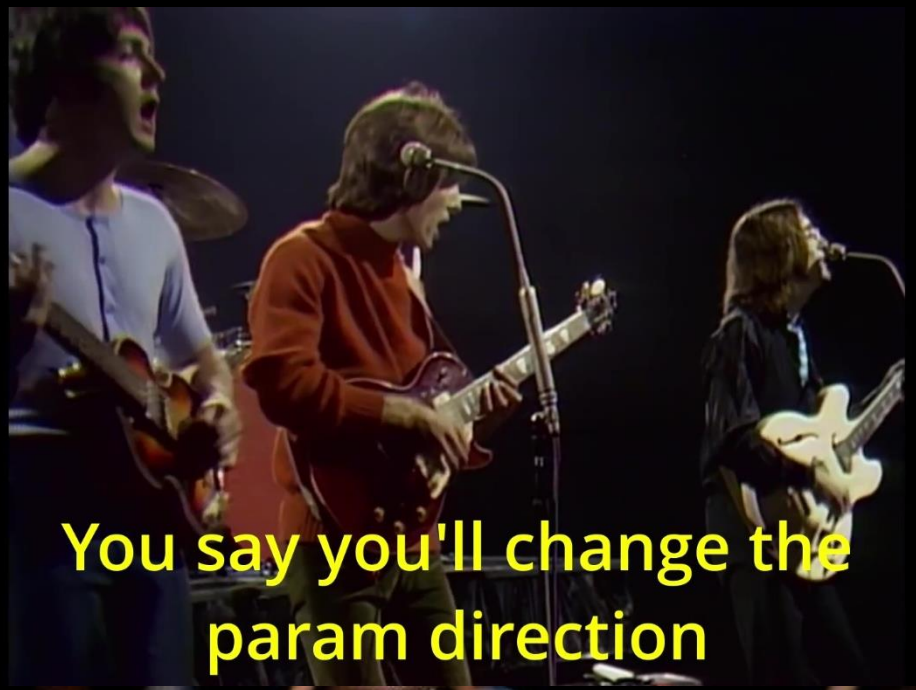

```
bool isFacingNorth(Orientation & dir,
                   double tolerance)
{
   ... math on dir …
   if (… < tolerance …)
       return true;
   return false;
};
```




```
void f(std::vector<Item> & v, Etc etc)
{
   …
};
```




```
std::vector<Item> f(Etc etc)
{
  std::vector<Item> v;
  …
  return v;
};
```




```
std::vector<Item> f(Etc etc)
{
  std::vector<Item> v;
  …
  return v;
};
```

```
void veryImportantFunction()
{
  for (…) {
     auto v = f(etc);
     …
  }
}
```

```
std::vector<Item> f(Etc etc)
{
  std::vector<Item> v;
  …
  return v;
};
```

```
void veryImportantFunction()
{
  std::vector<Item> v;
  for (…) {
     f(v, etc);
     …
     v.clear();
  }
}
```

```
void f(std::vector<Item> & v, Etc etc)
{
   …
};
```

```
void veryImportantFunction()
{
  std::vector<Item> v;
  for (…) {
     f_ugly(v, etc);
     …
     v.clear();
  }
}
```

```
void f_ugly(std::vector<Item> & v, Etc etc
{
   …
};
```

```
std::vector<Item> f(Etc etc)
{
  std::vector<Item> v;
  f_ugly(v, etc);
  return v;
};
```

```
void veryImportantFunction()
{
  std::vector<Item> v;
  for (…) {
     f(v, etc);
     …
     v.clear();
  }
}
```

What's going on?

```
void veryImportantFunction()
{
  std::vector<Item> v;
  for (…) {
     v = f(etc);
     …
  }
}
```

f() takes an etc, makes a v

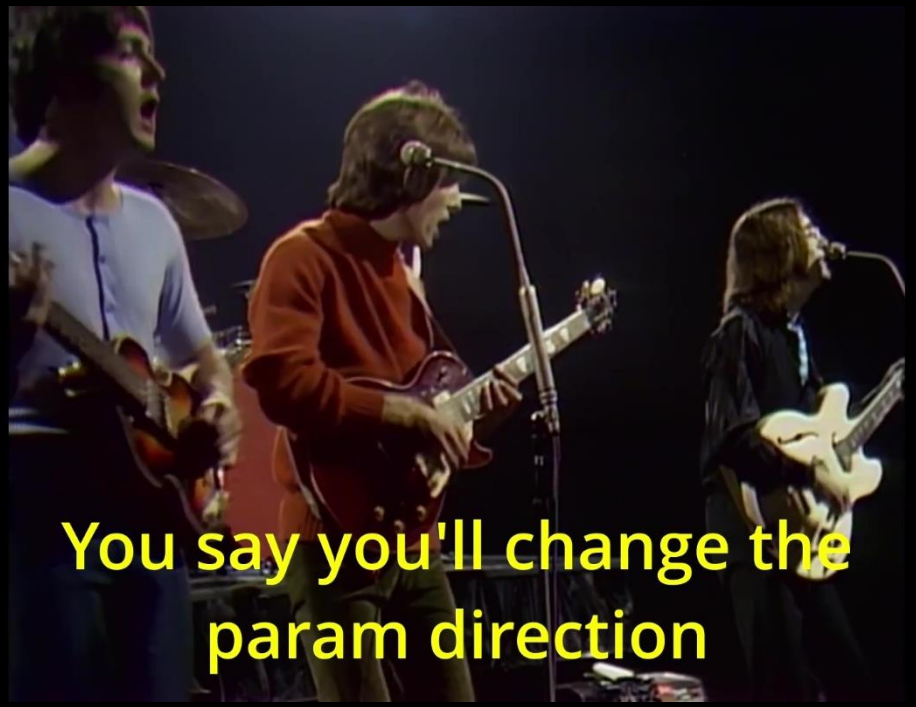
```
void veryImportantFunction()
{
  std::vector<Item> v;
  for (…) {
     f(v, etc);
     …
     v.clear();
  }
}
```

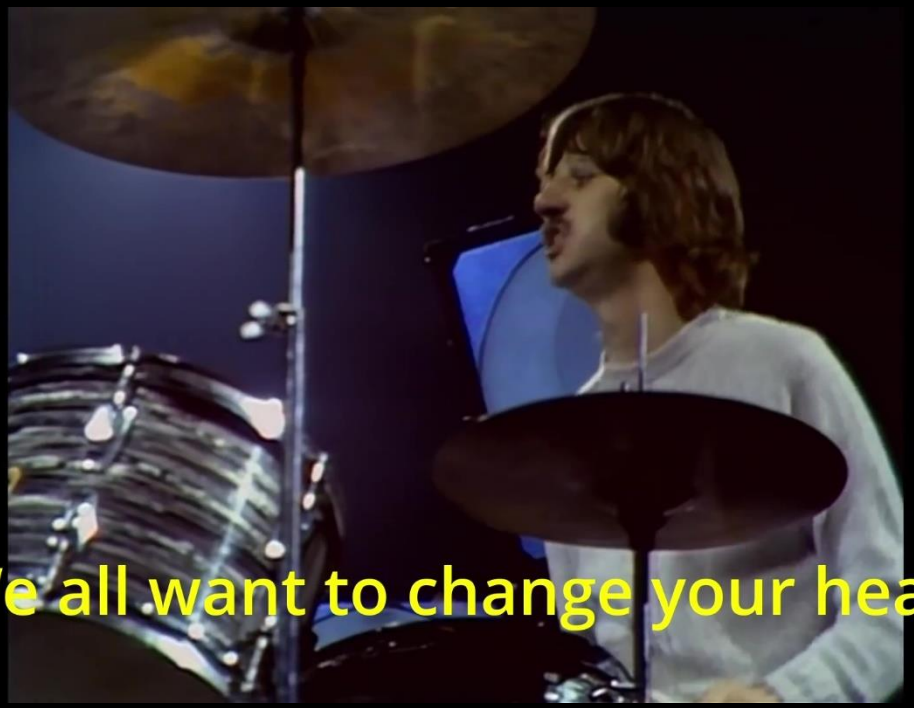
```
void veryImportantFunction()
{
  std::vector<Item> v;
  for (…) {
     v = f(etc);
     …
  }
}
```

What's going on?

f() takes an etc, makes a v

- etc (probably) not modified
- v modified (obviously)
- **Essential for debugging**


You say you'll change the
param direction


We all want to change your head

Speaking of names…

Speaking of names…

```
void StarOn(Sneetch & sneetch, int num)
{
  vector locs = calcStarPos(sneetch, num);
  applyStars(sneetch, locs);
};
```

Speaking of names…

```
void StarOn(Sneetch & sneetch, int num)
{
  vector locs = calcStarPositions(sneetch, num);
  applyStars(sneetch, locs);
};
```

Speaking of names…

Work:
- calc, find, determine

No Work:
- get

```
void StarOn(Sneetch & sneetch, int num)
{
  vector locs = getStarPositions(sneetch, num);
  applyStars(sneetch, locs);
};
```

Speaking of names…

Work:

- calc, **find**, determine

No Work:

- get

Speaking of names…

Work:
- calc, find, determine

No Work:
- get

Don’t mention param types

```
void StarOn(Sneetch & sneetch, int num)
{
  vector locs = getStarPositionsForSneetch(sneetch, n
  applyStars(sneetch, locs);
};
```

But Tony…

But Tony…

those are just made up examples!

We work in the real world with real code…

```
...
...
// step 1, turn 2D camera pts into 3D points
std::vector<Point3D> cam3d[2];
std::vector<Point2D> proj2d[2];
auto& camData = getCamData();
auto& projData = getProjData();
for (int ptIdx = 0; ptIdx < camData.size(); ptIdx++)
{
    auto& camPoint = camData[ptIdx];
    Point3D p1;
    Point3D p2;
    m_camera->mapPointTo3D(camPoint, p1, p2, calibrationType);
    auto iset = sg->intersect(p1, p2);

    for (auto& intersection : iset)
    {
       ...
       ...
    }
}
...
...
// step 2, solve via mapper
...
// step 3, ...
...
// validate fit
...
```

```
...
...
// step 1, turn 2D camera pts into 3D points
std::vector<Point3D> cam3d[2];
std::vector<Point2D> proj2d[2];
auto& camData = getCamData();
auto& projData = getProjData();
for (int ptIdx = 0; ptIdx < camData.size(); ptIdx++)
{
    auto& camPoint = camData[ptIdx];
    Point3D p1;
    Point3D p2;
    m_camera->mapPointTo3D(camPoint, p1, p2, calibrationType);
    auto iset = sg->intersect(p1, p2);

    for (auto& intersection : iset)
    {
       ...
       ...
    }
}
...
...
// step 2, solve via mapper
...
// step 3, ...
...
// validate fit
...
```

No raw loops.
- Sean Parent

```
...
...

std::vector<Point3D> cam3d = cameraTo3D(…);

...
...
// step 2, solve via mapper
...
// step 3, ...
...
// validate fit
...
```

No raw loops.
- Sean Parent

```
...
...

std::vector<Point3D> cam3d = camTo3D(…);

...
...
// step 2, solve via mapper
...
// step 3, ...
...
// validate fit
...
```

```
...
...

std::vector<Point3D> cam3d = camTo3D(…);

...
...
// step 2, solve via mapper
...
// step 3, ...
...
// validate fit
validate_fit(…);
...
```

Fluent {C++}

Jonathan Boccara's blog

POPULAR

- How to split a string in C++
- Making code expressive with lambdas
- The Complete Guide to Building Strings In

Tweet

Which One Is Better: Map of Vectors, or Multimap?

Published April 10, 2018 - 10 Comments

While advising on how to make code more expressive on the SFME project, I came across an interesting case of choosing the right data structure, which I'll share with you with the permission of the authors of the projects.

We had to associate a key with several values, and perform various operations. Should we use a map of vectors, or is a multimap more appropriate? Let's see the case in more details, and compare the two solutions.

The case: an event mediator

Fluent {C++}

Jonathan Boccara's blog

POPULAR

- How to split a string in C++
- Making code expressive with lambdas
- The Complete Guide to Building Strings In

Tweet

Which C
Vecto

While advising on how to make
interesting case of choosing the
of the authors of the projects.

We had to associate a key with
map of vectors, or is a multimap more appropriate? Let's see the case in more details, and compare the two solutions.

The case: an event mediator

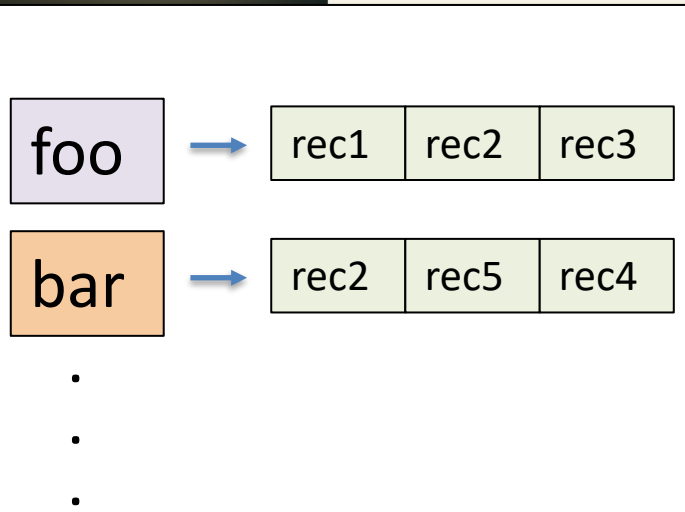
Fluent {C++}

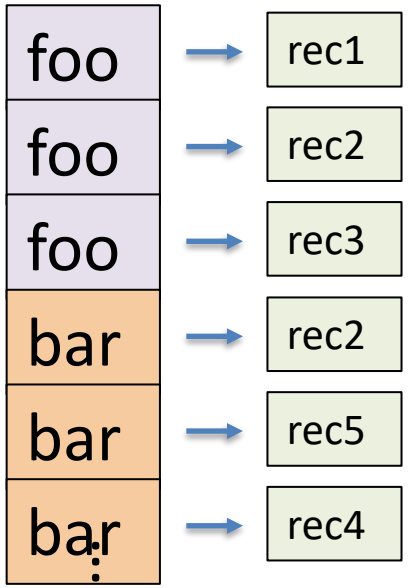
Jonathan Boccara's blog

Tweet

Which One Is Better: Map of Vectors, or Multimap?

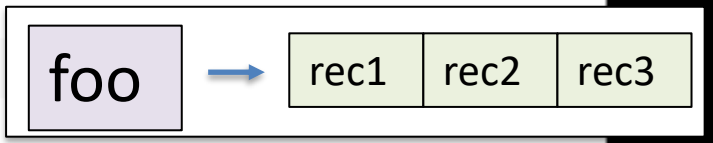
Published April 10, 2018 - 10 Comments





CHRISTIE

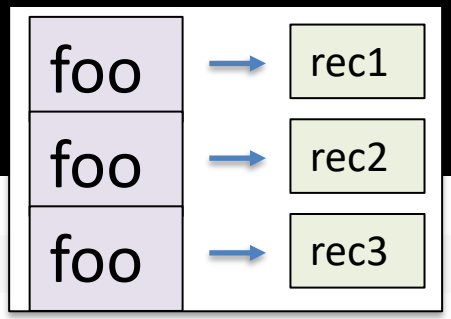
```cpp
void EventMediator::emit(Event const& event) const
{
    auto eventID = event.getEventID();
    auto receiversEntry = receiversRegistry_.find(eventID);
    if (receiversEntry != end(receiversRegistry_))
    {
        auto const& receivers = receiversEntry->second;
        for (auto const& receiver : receivers)
        {
            receiver->reactTo(event);
        }
    }
}
```



//get all receivers for event

//call each receiver

CHRISTIE




```
void EventMediator::emit(Event const& event) const
{
    auto eventID = event.getEventID();
    auto receiversEntries = receiversRegistry_.equal_range(eventID); //get all receivers for event
    for (auto receiverEntry = receiversEntries.first; receiverEntry != receiversEntries.second; ++receiverEntry)
    {
        auto const& receiver = receiverEntry->second;
        receiver->reactTo(event);
    }
}
```

//call each receiver

```
void Mediator::notifyListeners(Event const & ev)
{
```

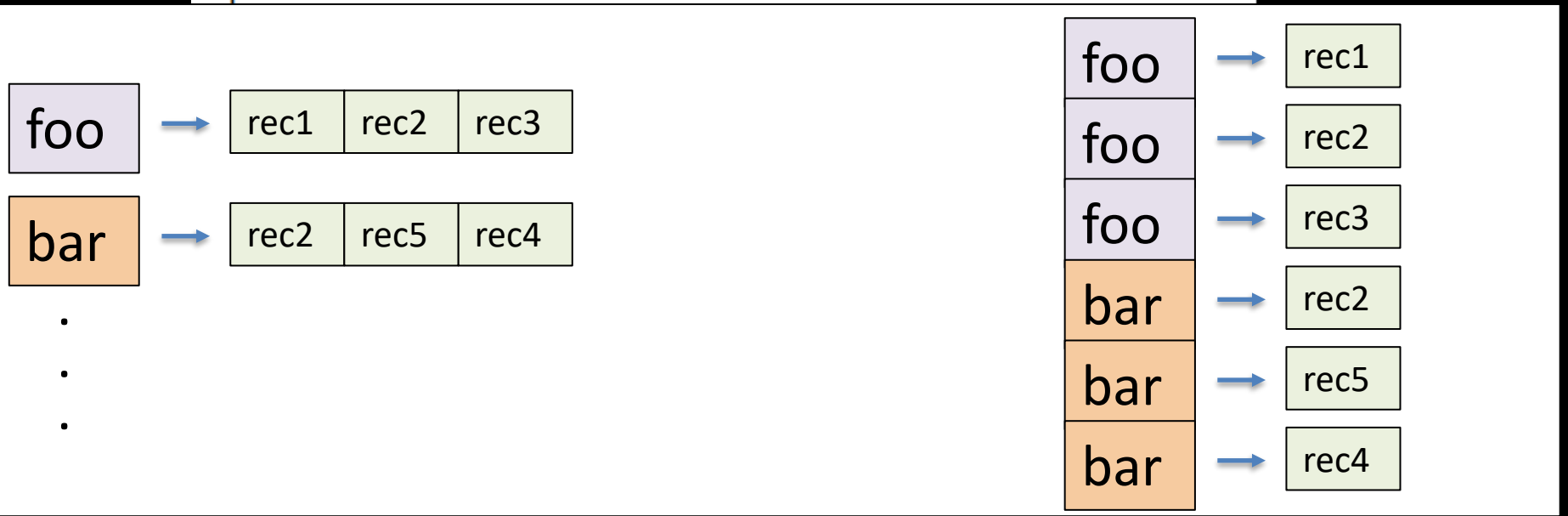
CHRISTIE

```
void Mediator::notifyListeners(Event const & ev)
{
    auto const & listeners = getListeners(ev); //get all receivers for even
    for (auto listener : listeners)
        listener->nonBlockingNotify(ev);
}
```

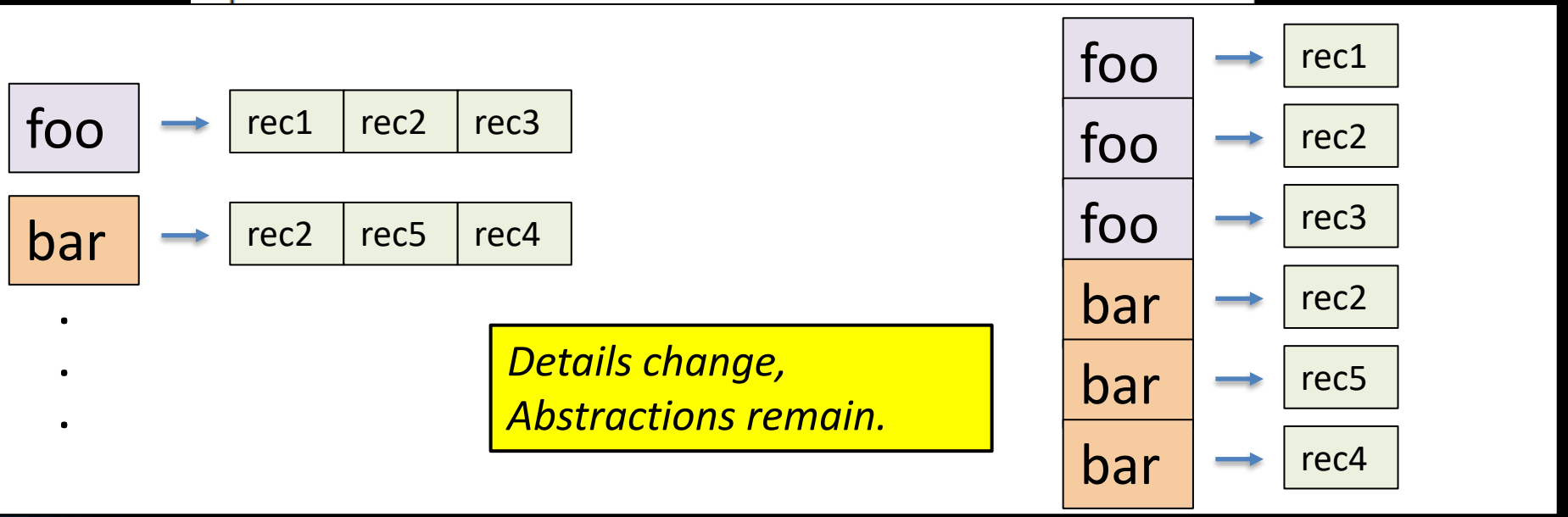
//call each receiver

```
void Mediator::notifyListeners(Event const & ev)
{
    auto const & listeners = getListeners(ev);
    for (auto listener : listeners)
        listener->nonBlockingNotify(ev);
}
```

```
void Mediator::notifyListeners(Event const & ev)
{
    auto const & listeners = getListeners(ev);
    for (auto listener : listeners)
        listener->nonBlockingNotify(ev);
}
```



```
void Mediator::notifyListeners(Event const & ev)
{
    auto const & listeners = getListeners(ev);
    for (auto listener : listeners)
        listener->nonBlockingNotify(ev);
}
```



```
bool push(int val)
{
 int prev = 0;
 geni ent;
 geni tmp;
 geni old = tmp = tail;  // laxtomic load
 do {
   ent = buffer[tmp].load(relaxed);
   while( ! is_zero(ent, tmp.gen) ) {
      if (ent.gen < prev) {
        while(!tail.CAS(old,tmp) && old < tmp) { }
        return false; // full
      }  else tmp.incr();
      if (ent.data) prev = ent.gen;
   }
   geni newg{val, tmp.gen};
 } while ( ! buffer[tmp].CAS(ent, newg, release));
 tmp.incr(); // go to next
 // update if no one else has gone as far:
 while (!tail.CAS(old, tmp) && old < tmp)  { }
 return true;
}
```

CHRISTIE

```
bool push(int val)
{
 int prev = 0;
 geni ent;
 geni tmp;
 geni old = tmp = tail;  // laxtomic load
 do {
   ent = buffer[tmp].load(relaxed);
   while( ! is_zero(ent, tmp.gen) ) {
      if (ent.gen < prev) {
        while(!tail.CAS(old,tmp) && old < tmp) { }
        return false; // full
      }  else tmp.incr();
      if (ent.data) prev = ent.gen;
   }
   geni newg{val, tmp.gen};
 } while ( ! buffer[tmp].CAS(ent, newg, release));
 tmp.incr(); // go to next
 // update if no one else has gone as far:
 while (!tail.CAS(old, tmp) && old < tmp)  { }
 return true;
}
```

No raw loops.
- Sean Parent

CHRISTIE

```
bool push(int val)
{
 int prev = 0;
 geni ent;
 geni tmp;
 geni old = tmp = tail;  // laxtomic load
 do {
   ent = buffer[tmp].load(relaxed);
   while( ! is_zero(ent, tmp.gen) ) {
      if (ent.gen < prev) {
        while(!tail.CAS(old,tmp) && old < tmp) { }
        return false; // full
      }  else tmp.incr();
      if (ent.data) prev = ent.gen;
   }
   geni newg{val, tmp.gen};
 } while ( ! buffer[tmp].CAS(ent, newg, release));
 tmp.incr(); // go to next
 // update if no one else has gone as far:
 while (!tail.CAS(old, tmp) && old < tmp)  { }
 return true;
}
```

No raw loops.
- Sean Parent

CHRISTIE

```
bool push(int val)
{
 int prev = 0;
 geni ent;
 geni tmp;
 geni old = tmp = tail;  // laxtomic load
 do {
   ent = buffer[tmp].load(relaxed);
   while( ! is_zero(ent, tmp.gen) ) {
     if (ent.gen < prev) {
       while(!tail.CAS(old,tmp) && old < tr
       return false; // full
     }  else tmp.incr();
     if (ent.data) prev = ent.gen;
   }
   geni newg{val, tmp.gen};
 } while ( ! buffer[tmp].CAS(ent, newg, release));
 tmp.incr(); // go to next
 // update if no one else has gone as far:
 while (!tail.CAS(old, tmp) && old < tmp)  { }
 return true;
}
```

```
void push(int val) {
  geni pos = tailish; // relaxed load
  do {
    pos = find_tail(pos);
  } while (!try_write_value(pos, val));
  tailish = pos+1;  // thanks Sebastian
}

geni find_tail(geni pos) { // precond: pos <= tail
 while(!maybe_tail(buff[pos.val].load(relaxed), pos.gen))
    pos++;
 return pos;
}

bool maybe_tail(entry e, int gen) {
  return e.data == 0 && e.gen == gen
       || e.data  != 0 && e.gen < gen;
}

bool try_write_value(geni pos, int val) {
  entry old{0, pos.gen};
  entry nu{val, pos.gen};
  return buffer[pos].c_e_weak(old, nu, release, relaxed);
}
```

```
bool push(int val)
{
 int prev = 0;
 geni ent;
 geni tmp;
 geni old = tmp = tail;  // laxtomic load
 do {
   ent = buffer[tmp].load(relaxed);
   while( ! is_zero(ent, tmp.gen) ) {
     if (ent.gen < prev) {
       while(!tail.CAS(old,tmp) && old < tr
       return false; // full
     }  else tmp.incr();
     if (ent.data) prev = ent.gen;
   }
   geni newg{val, tmp.gen};
 } while ( ! buffer[tmp].CAS(ent, newg, release));
 tmp.incr(); // go to next
 // update if no one else has gone as far:
 while (!tail.CAS(old, tmp) && old < tmp)  { }
 return true;
}
```

```
void push(int val) {
   geni pos = tailish; // relaxed load
   c
   }
   t
}

ge
  w                                                    n))

  r
}

bo
   return e.data == 0 && e.gen == gen
         || e.data  != 0 && e.gen < gen;
}

bool try_write_value(geni pos, int val) {
   entry old{0, pos.gen};
   entry nu{val, pos.gen};
   return buffer[pos].c_e_weak(old, nu, release, relaxed);
}
```

A committee member: “why don’t we have a lambda_inserter…”

```
std::sample(in.begin(), in.end(), std::lambda_inserter([](auto && item) {
   std::cout << item << "\n";
}), 5, std::mt19937{std::random_device{}()});
```

A committee member: “why don’t we have a lambda_inserter…”

```
std::sample(in.begin(), in.end(), std::lambda_inserter([](auto && item) {
   std::cout << item << "\n";
}), 5, std::mt19937{std::random_device{}()});
```

“I need to write this instead:”

```
struct HitEstimatorReference {
    HitEstimator * t;
    using value_type = ValueType;
    void push_back(const ValueType & value) {
        t->Add(value);
    }
};

HitEstimatorReference ref{this};

std::sample(data.begin(), data.end(), std::back_inserter(ref), count,
std::mt19937{std::random_device{}()});
```

A committee member: “why don’t we have a lambda_inserter…”

```
std::sample(in.begin(), in.end(), std::lambda_inserter([](auto && item) {
   std::cout << item << "\n";
}), 5, std::mt19937{std::random_device{}()});
```

Write the functions
you want to see in the world.

“I need to write this instead:”

```
struct HitEstimatorReference {
    HitEstimator * t;
    using value_type = ValueType;
    void push_back(const ValueType & value) {
        t->Add(value);
    }
};

HitEstimatorReference ref{this};

std::sample(data.begin(), data.end(), std::back_inserter(ref), count,
std::mt19937{std::random_device{}()});
```

A committee member: “why don’t we have a lambda_inserter…”

```
std::sample(in.begin(), in.end(), std::lambda_inserter([](auto && item) {
   std::cout << item << "\n";
}), 5, std::mt19937{std::random_device{}()});
```

Write the functions you want to see in the world.

“I need to write this instead:”

```
struct HitEstimatorReference {
    HitEstimator * t;
    using value_type = ValueType;
    void push_back(const ValueType & value)
        t->Add(value);
    }
};

HitEstimatorReference ref{this};

std::sample(data.begin(), data.end(), std::back_inserter(ref), count,
std::mt19937{std::random_device{}()});
```

```
// lambda_inserter
// converts any function taking one arg
// into a back_inserter style iterator
template<typename T>
struct lambda_inserter
{
…
}

// sample that takes a output function instead of an iterator
sample(Iterator beg, Iterator end, Function perSample, etc…)
```

```
void AutoMapEnvironmentSensing::gatherImageData(ICameraControl* camer
{
    StructuredLightLib::DataSet* ds = m_state.ds.get();
    ds->m_cameras.clear();
```

```
void AutoMapEnvironmentSensing::gatherImageData(ICameraControl* camer
{
    StructuredLightLib::DataSet* ds = m_state.ds.get();
    ds->m_cameras.clear();
```

THREADS!?!
Pure functions are thread safe!

Master, I am honoured by your visit.

I was out walking my dog. Your function, why does it take BigCommonStruct instead of just x and y?

BCS is where we keep x and y, master.

<Bark>

He's hungry.

Shall I prepare him food?

No, just open the fridge, let him take what he wants.

- Ancient C++ Koan

*Code **top-down** on the way down,*
*and **bottom-up** on the way back up.*

```
void AutoMapEnvironmentSensing::gatherImageData(ICameraControl* camer
{
    StructuredLightLib::DataSet* ds = m_state.ds.get();
    ds->m_cameras.clear();
```

*Code **top-down** on the way down,*
*and **bottom-up** on the way back up.*

```
void AutoMapEnvironmentSensing::gatherImageData(ICameraControl* camer
{
    StructuredLightLib::DataSet* ds = m_state.ds.get();
    ds->m_cameras.clear();
```

*Code **top-down** on the way down,*
*and **bottom-up** on the way back up.*

```
void AutoMapEnvironmentSensing::gatherImageData(ICameraControl* camer
{
    StructuredLightLib::DataSet* ds = m_state.ds.get();
    ds->m_cameras.clear();
```

*Returning **void** is a code smell*

```cpp
bool AutoCalSensing::SingleShotEnvironmentSensing::addCorrespondences(SSTP::BlobGridParams gri
{
    CameraProjectorCorrespondencePointSet correspondencesSet;

    // calculate correspondences

    //
    //
    //
    // 150 lines of code...
    //
    //
    //

    m_cameraProjectorCorrespondencePointSets.push_back(correspondencesSet);

    return true;
}
```

```
bool AutoCalSensing::SingleShotEnvironmentSensing::addCorrespondences(SSTP::BlobGridParams gri
{
    CameraProjectorCorrespondencePointSet correspondencesSet;

    // calculate correspondences

    //
    //
    //
    // 150 lines of code...
    //
    //
    //

    m_cameraProjectorCorrespondencePointSets.push_back(correspondencesSet);

    return true;
}
```

*Separate **calculating** from **doing***

- Grokking Simplicity by Eric Normand

```
bool AutoCalSensing::SingleShotEnvironmentSensing::addCorrespondences(SSTP::BlobGridParams gri
{
    intSet correspondencesSet;

    // calculate correspondences

    //
    //
    //
    // 150 lines of code...
    //
    //
    //

    m_cameraProjectorCorrespondencePointSets.push_back(correspondencesSet);

    return true;
}
```

*Returning **void** is a code smell*

*Separate **calculating** from **doing***
- Grokking Simplicity by Eric Normand

```
bool AutoCalSensing::SingleShotEnvironmentSensing::addCorrespondences(SSTP::BlobGridParams gri
{
    CameraProjectorCorrespondencePointSet correspondencesSet = determineCorrespondences(grid, 

    m_cameraProjectorCorrespondencePointSets.push_back(correspondencesSet);

    return true;
}
```

*Returning **void** is a code smell*

*Separate **calculating** from **doing***

- Grokking Simplicity by Eric Normand

```
bool AutoCalSensing::SingleShotEnvironmentSensing::addCorrespondences(SSTP::BlobGridParams gri
{
    CameraProjectorCorrespondencePointSet correspondencesSet = determineCorrespondences(grid, c

    m_cameraProjectorCorrespondencePointSets.push_back(correspondencesSet);

    return true;
}
```

*Returning **void** is a code smell*

*Separate **calculating** from **doing***
- Grokking Simplicity by Eric Normand

Can’t predict the future?

```
bool AutoCalSensing::SingleShotEnvironmentSensing::addCorrespondences(SSTP::BlobGridParams gri
{
    CameraProjectorCorrespondencePointSet correspondencesSet = determineCorrespondences(grid, c

    m_cameraProjectorCorrespondencePointSets.push_back(correspondencesSet);

    return true;
}
```

*Returning **void** is a code smell*

*Separate **calculating** from **doing***
- Grokking Simplicity by Eric Normand

Can’t predict the future?
Don’t need to.

Can’t predict the future?
Don’t need to.

*Code **top-down** on the way down, and **bottom-up** on the way back up.*

Write the functions you want to see in the world.

*Separate **calculating** from **doing***
- Eric Normand

Functions are answers to questions.

KYSS: Keep your stuff separate

No raw loops.
- Sean Parent

Pure functions are thread safe!

*Returning **void** is a code smell*

Functions are for sharing

Local Reasoning!

Details change, Abstractions remain.

Functions are the fundamental building block of programming

Functions can be barriers to Complecting

Can't predict the future?
Don't need to.

Classes are made of Velcro

Classes are a nexus of Complecting

Good code begets good code

You say you want to write a function....

Value Oriented Programming, Part 1: Functions

Tony Van Eerd

C++Now 2023

CHRISTIE