Value Oriented Programming. Part 1
You Say You Want to Write a Function
Tony Van Eerd
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
A Possible Future of Software Development

Sean Parent
Principal Scientist & Manager
Software Technology Lab

July 25, 2007
May 16, 2007
BoostCon #1
You say you want to write a function....

Value Oriented Programming, Part 1: Functions

Tony Van Eerd
C++Now 2023
A long time ago in a galaxy far, far away....
You say you want to write a function....
You say you want to write a function....

Now that is a lie!
void veryImportantFunction()
{
    ...
    Projector proj ...
    ...
    proj.showPattern(delta) ...
    ...
};
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() < 5 * M_PI / 180)
};
void veryImportantFunction()
{
    ...
    Projector proj ...
    ...
    proj.showPattern(delta) ...
    ...

    // if projector facing North...
    if (proj->getPose().orientation().yaw() < 5 * M_PI / 180)
        ...
};
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)
        ...
};
void veryImportantFunction() {
    ...  // Code
    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)
        ...
};
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)
        ...
};
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)
        ...
};
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)
    ...
};
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)
        ...
};
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)
        …
};
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)
        ...
};
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)
        ...
};
Rich Hickey (Clojure)
*Simple Made Easy*

**Simple/Complex**

*plecto (latin) – fold / braid*

Simple – one fold
Complex – many folds
Rich Hickey (Clojure)
Simple Made Easy

Easy
adjacens (latin)
close at hand

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

Tony Van Eerd
C++Now, May 2021
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)
        ...
};
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)
        ...
};

“How did we get here?
One step at a time.
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 / 180 
        || proj->getPose().orientation().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; 
    if (proj->getPose().orientation().yaw() < 5 * M_PI / 180 || 
        proj->getPose().orientation().yaw() > 355 * M_PI / 180) 
        ... 
};
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)
        ...
};
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)
        ...
};
void veryImportantFunction()
{
    ... // code...
    Projector proj ...
    ... // code...
    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 / 180
        ...
}
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 / 180)
  {
    ...
  }
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 / 180)
        ...
}
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 / 180)
        {
            ...
        }
    }
}
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))
            ...
}
void veryImportantFunction()
{
    …
    Projector proj …
    …
    proj.showPattern(delta) …
    …
    // check for north OTHERWISE …
    if (projector_facing_North(proj))
        if (isFacingNorth(proj))
            …
};
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;
};
There's a function for that
#CppKoan

There's a function for that

What's it called?
There's a function for that
What's it called?
Guess
There's a function for that
What's it called?
Guess
readConfigFile()?
There's a function for that
What's it called?
Guess
readConfigFile()?
Pass the filename
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
What's it called?
Guess
readConfigFile()?
Pass the filename
What's the path?
There's a function for that too
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?
```
#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
```
#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())?
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.
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.
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;
};
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;
};

Functions are answers to questions. Write the functions you want to see in the world. Code top-down on the way down.
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;
}
bool isFacingNorth(Projector proj) {
    auto dir = proj.orientation();
    auto tolerance = proj...

    proj.showVideo(...)

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

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

Complecting Made Easy

Tony Van Eerd
C++Now, May 2021
```cpp
bool isFacingNorth(Projector proj) {
    auto dir = proj.orientation();
    auto tolerance = proj...;
    proj.showVideo(...)
    ... math on dir ...
    if (... < tolerance ...) return true;
    return false;
}

void veryImportantFunction() {
    ... Projector proj ...
    ... if (isFacingNorth(proj)) ...
};
```
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.
AND YOU GET A MODEL POINTER!

AND YOU GET A MODEL POINTER!
AND YOU GET A MODEL POINTER!

AND YOU GET A MODEL POINTER!
```cpp
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.
void veryImportantFunction()
{
    ... Projector proj ...
    ...
    if (isFacingNorth(proj.orientation(),5)
    ...
};

bool isFacingNorth(Orientation dir, double tolerance)
{
    ... math on dir ...
    if (... < tolerance ...)
        return true;
    return false;
};
void veryImportantFunction()
{
    ...
    Projector proj ...
    ...
    if (isFacingNorth(proj.orientation(), 5))
        ...
};

bool isFacingNorth(Orientation dir, double tolerance)
{
    ...
    ... math on dir ...
    if (... < tolerance ...)
        return true;
    return false;
};
void veryImportantFunction()
{
    ... Projector proj ...
    ...
    if (isFacingNorth(proj.orientation(), 5))
        ...
    ...}

bool isFacingNorth(Orientation dir, double tolerance)
{
    ... 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 tol = getTolerance(proj);
    return isFacingNorth(dir, tol);
};

bool isFacingNorth(Orientation dir, double tolerance)
{
    ... 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 tol = getTolerance(proj);
    return isFacingNorth(dir, tol);
};

bool isFacingNorth(Orientation dir, double tolerance)
{
    ... 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;
};
bool isFacingNorth(Projector proj)
{
    auto dir = proj.orientation();
    auto tolerance = proj...

    proj.showVideo(…)

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

void veryImportantFunction()
{
    ...
    Projector proj ...
    ...
    if (isFacingNorth(proj))
        ...
}
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;
};
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)
{
    ...
    if (... < tolerance ...)  
        return true;
    return false;
};

**KYSS:**
Keep Your Stuff Separate

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

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

All I can tell you is, developer, you have to wait.
Value Oriented Programming, Part 1: Functions
Value Oriented Programming, Part 1: Functions

Takes input, MODIFIES it, produces output

Takes input, produces output

17 → 23
Separate *calculating* from *doing*
- *Grokking Simplicity* by Eric Normand

Takes input, MODIFIES it, produces output

Takes input, produces output

17 → 23
Separate *calculating* from *doing*  
- *Grokking Simplicity* by Eric Normand

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

Takes input, MODIFIES it, produces output
But if you are still writing functions that return void now

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

You ain't gonna make it with anyone anyhow

Takes input, MODIFIES it, produces output

```c
void StarOn(Sneetch & sneetch) {
    // measure sneetch...
    // find belly...
    // calculate star position(s)...
    // apply star(s) to sneetch
};
```
Separate *calculating* from *doing*
- Grokking Simplicity by Eric Normand

You ain't gonna make it with anyone anyhow

Takes input, MODIFIES it, produces output

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

*Returning void is a code smell*
Separate *calculating* from *doing*  
- Grokking Simplicity by Eric Normand

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

Takes input, MODIFIES it, produces output
Takes input, MODIFIES it, produces output

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

```cpp
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

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

“pure” functions
- Same input gives same output
- No side effects

Separate calculating from doing
- Grokking Simplicity by Eric Normand
void StarOn(Sneetch & sneetch, int num) {
    vector locs = calcStarPos(sneetch, num);
    applyStars(sneetch, locs);
};

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);
};

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

"pure" functions
- Same input gives same output
- No side effects

Local Reasoning

Separate calculating from doing
- Grokking Simplicity by Eric Normand

Takes input, MODIFIES it, produces output

Takes input, produces output
David John Wheeler
(previously worked with Bjarne Stroustrup)
Recent breakthrough research
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 cost 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 classification 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 with Bjarne Stroustrup)

Recent breakthrough research

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 re-
turned to the appropriate point to which it
is required. One of the simpler methods was
that used for the closed sub-routines of the
SIMAC 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.

Because, however, we have seen that the
orders that are obeyed are identical with those
of the machine. However, the interactive 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 movements of the pro-
gram 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 programmer's knowledge of the pro-
gramme 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
otherwise.

[Page 235]
David John Wheeler (previously worked with Bjarne Stroustrup)

Recent breakthrough research

David John Wheeler (previously worked with Bjarne Stroustrup)

Recent breakthrough research

Sub-routines seem to have two distinct uses in programs. 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 borne 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.
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 generalised 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 organisation of the individual sub-routines there remains the method of the general organisation 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
“allows the coder to concentrate on one section of a programme at a time without the overall detailed programme continually intruding”
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 in 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 division 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 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 ready to use a sub-routine which will meet the specifications with a small amount of manipulation than to make one speculatively 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 generalised 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 organisation of the individual sub-routines there remains the method of the general organisation 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
THE USE OF SUB-RUTINES 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, which is 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 generalised 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.
Functions are the fundamental building block of programming

Recent breakthrough research - 1951
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())
    ...
};
bool isFacingNorth(Orientation dir, double tolerance) {
/* math on dir */
if (dir < tolerance) return true;
return false;
}

class Projector {
public:

    bool isFacingNorth() const;

private:

/* implementation */
};

void veryImportantFunction() {
    ... Projector proj ...
    ...
    if (proj.isFacingNorth())
    ...
    if (cam.isFacingNorth())
};
void veryImportantFunction()
{
   ...
   Projector proj ...
   ...
   if (proj.isFacingNorth())
      ...
   if (cam.isFacingNorth())
};
bool isFacingNorth(Orientation dir, double tolerance) {
    ...
    if (... < tolerance ...) {
        return true;
    } else {
        return false;
    }
}

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

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

class Camera : Device {
    bool isFacingNorth() const override;
    ...
};
void veryImportantFunction()
{
    ...
    Projector proj ...
    ...
    if (proj.isFacingNorth()) ...
    ...
    if (cam.isFacingNorth());
};
void veryImportantFunction()
{
    ... Projector proj ...
    ...
    if (proj.isFacingNorth())
    ...
    if (cam.isFacingNorth())
};
class DirectionalDevice : Device
{
    virtual bool isFacingNorth() const;
};

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

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

void veryImportantFunction()
{
    ... Project...
    ... if (proj)
    ... if (cam)
};
void veryImportantFunction()
{
    ... // Code
    Projector proj ...
    ...
    if (proj.isFacingNorth())
    {
        ...
    }
    if (cam.isFacingNorth())
    {
        ...
    }
}
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;
}
bool isFacingNorth(Orientation dir, double tolerance)
{
    ... math on dir ...
    if (... < tolerance ...)
        return true;
    return false;
};
```cpp
void veryImportantFunction()
{
    ...
    Projector proj ...
    ...
    if (proj.orientation.isFacingNorth(5))
    ...
};

class Orientation
{
    ...
    bool isFacingNorth(double tol) const;
    ...
};
```
```cpp
bool isFacingNorth(Orientation, double);

class Orientation
{
    ... bool isFacingNorth(double tol) const;
    ...
};
```
```cpp
bool isFacingNorth(Orientation, double);
```

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

“Isomorphic”
bool isFacingNorth(Orientation dir, double tolerance);

void veryImportantFunction()
{
    ... math on dir ...
    if (... < tolerance ...) return true;
    return false;
}

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

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

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

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

Classes are made of Velcro
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(Orientation dir, double tolerance) {
    // math on dir
    if (... < tolerance ...) return true;
    return false;
}

void veryImportantFunction() {
    ... Projector proj ...
    ... if (proj.orientation.isFacingNorth(5)) ...
}
bool isFacingNorth(Orientation, double tol) const;

void veryImportantFunction() {
    ... Projector proj ...
    ...
    if (proj.orientation.isFacingNorth(5)) ...
    ...
}
bool isFacingNorth(Orientation, double);

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

Uses external Orientation API

Classes are made of Velcro

Classes are a nexus of Complecting

Accesses internals

Classes are made of Velcro

Classes are a nexus of Complecting

Uses external Orientation API

Accesses internals
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);

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

Uses external Orientation API

Accesses internals

“Invariants”
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;
    …
}

Using external Orientation API

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
    {
        return true;
    }

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?

```java
string func(string s);
```
How many string functions are there?

```cpp
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

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

class Orientation
{
    ...    
    bool isFacingNorth(double tol) const;
    ...    
};
```
void veryImportantFunction()
{
    ... Projector proj ...
    ...
    if (isFacingNorth(proj.orientation, 5))
        ...
};

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

Classes are made of Velcro

Classes are a nexus of Complecting
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;
};
You say you'll change the param direction

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

You say you'll change the param direction
You say you'll change the
dir param direction

It’s so beautiful.

```cpp
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();
}

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();
    }
}

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...

```c
void StarOn(Sneetch & sneetch, int num) {
    vector locs = calcStarPos(sneetch, num);
    applyStars(sneetch, locs);
};
```
void StarOn(Sneetch & sneetch, int num)
{
    vector locs = calcStarPositions(sneetch, num);
    applyStars(sneetch, locs);
};
Speaking of names...

Work:
- calc, find, determine

No Work:
- get

```c
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

```cpp
void StarOn(Sneetch & sneetch, int num) {
    vector locs = getStarPositionsForSneetch(sneetch, num);
    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
...
std::vector<Point3D> cam3d = cameraTo3D(...);
...
...
// step 2, solve via mapper
...
// step 3, ...
...
// validate fit
...
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
validate_fit(...);
Which One Is Better: Map of Vectors, or Multimap?

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
Which C++ Container to Use

While advising on how to make code more interesting, we came across an interesting case of choosing the correct data structure for the project.

We had to associate a key with a value. 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
Which One Is Better: Map of Vectors, or Multimap?
```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);
        }
    }
}
```
void EventMediator::emit(Event const& event) const
{
    auto eventID = event.getEventID();
    auto receiversEntries = receiversRegistry_.equal_range(eventID);
    for (auto receiverEntry = receiversEntries.first; receiverEntry != receiversEntries.second; ++receiverEntry)
    {
        auto const& receiver = receiverEntry->second;
        receiver->reactTo(event);
    }
}
void Mediator::notifyListeners(Event const & ev)
{
```cpp
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);
}
void Mediator::notifyListeners(Event const & ev) {
    auto const & listeners = getAllListeners(ev);
    for (auto listener : listeners)
        listener->nonBlockingNotify(ev);
}

Details change,
Abstractions remain.
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;
}
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;
}
bool push(int val)
{
    int prev = 0;
gen int ent;  
gen int tmp;
gen int 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;
        }
        gen int 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) {
    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;
}

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); 
}
void push(int val) {
    int prev = 0;
    geni ent;
    geni tmp;
    geni old = tmp = tail; // laxatomic 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;
}

bool push(int val) {
    geni pos = tailish; // relaxed load
do {
    }
    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 Am I A Good Programmer?

    Kate Gregory
    kate@gregons.com
    @gregons on Twitter
A committee member: “why don’t we have a lambda_inserter…”

```cpp
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…”

```c++
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:”

```c++
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{}}()));
```

“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{}}());
```

Write the functions you want to see in the world.
A committee member: “why don’t we have a lambda_inserter…”

“I need to write this instead:

```
Write the functions you want to see in the world.
```

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

```c++
void AutoMapEnvironmentSensing::gatherImageData(ICameraControl* camera)
{
    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.
Code **top-down** on the way down, and **bottom-up** on the way back up.

Returning **void** is a code smell
```
bool AutoCalSensing::SingleShotEnvironmentSensing::addCorrespondences(SSTP::BlobGridParams gridParams)
{
    CameraProjectorCorrespondencePointSet correspondencesSet;

    // calculate correspondences

    // 150 lines of code...

    m_cameraProjectorCorrespondencePointSets.push_back(correspondencesSet);

    return true;
}
```
bool AutoCalSensing::SingleShotEnvironmentSensing::addCorrespondences(SSTP::BlobGridParams grid)
{
    CameraProjectorCorrespondencePointSet correspondencesSet;

    // calculate correspondences

    //
    //
    //
    // 150 lines of code...
    //
    //
    m_cameraProjectorCorrespondencePointSets.push_back(correspondencesSet);

    return true;
}
bool AutoCalSensing::SingleShotEnvironmentSensing::addCorrespondences(SSTP::BlobGridParams grid)
{
    // 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
Returning **void** is a code smell

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

Can’t predict the future?
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.

Functions are answers to questions.

Pure functions are thread safe!

**Local Reasoning!**

Functions can be barriers to Complecting.

Returning **void** is a code smell

KYSS: Keep your stuff separate

Details change, Abstractions remain.

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

- Eric Normand

No raw loops.

- Sean Parent

Functions are for sharing

Functions are the fundamental building block of programming

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

Good code begets good code

Classes are made of Velcro

Classes are a nexus of Complecting
You say you want to write a function....

Value Oriented Programming, Part 1: Functions

Tony Van Eerd
C++Now 2023