Generic Programming

HackTheU 2019 Workshop

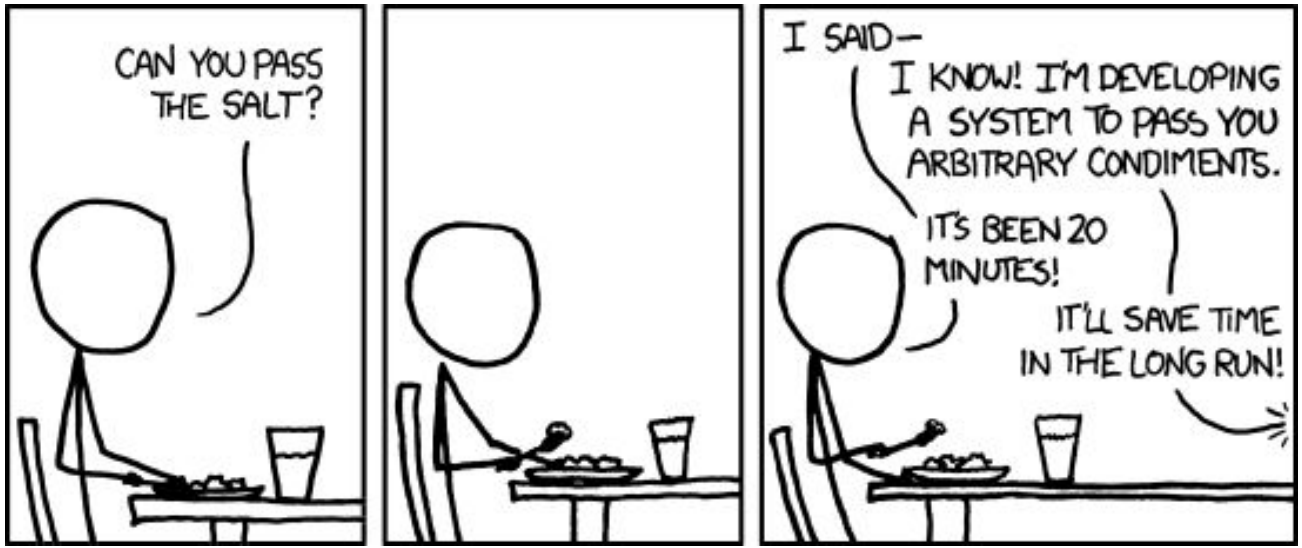
Joshua Smith

Why C++

- Loads of features
- Program how you want (OOP, Functional, Spaghetti)
- Fast and compiler optimized
    - Leverage the power of the compiler
- Access to full stack (hardware to OS)
- The language with which I happen to be most familiar

What does it mean to program generically?

- Generic - Relating to or descriptive of an entire group or class
- Program to solve a family of similar problems not just the one at hand
- Leave implementation details for later



Demo: Time a function - Specific Implementation

```
std::optional<int> timeFindMax(std::vector<int> const& v)
{
    std::chrono::time_point start = std::chrono::steady_clock::now();
    std::vector<int>::const_iterator i = std::max_element(v.begin(), v.end());
    std::chrono::time_point end = std::chrono::steady_clock::now();
    std::cout << "Duration: "
        << std::chrono::duration_cast<std::chrono::microseconds>(end - start).count()
        << " microseconds\n";
    if(i == v.end()) return {};
    return *i;
}
```

What implementation details are baked into this algorithm?

Demo: Time a function - Internal Types

```
std::optional<int> timeFindMax(std::vector<int> const& v)
{
    auto start = std::chrono::steady_clock::now();
    auto i = std::max_element(v.begin(), v.end());
    auto end = std::chrono::steady_clock::now();
    std::cout << "Duration: "
        << std::chrono::duration_cast<std::chrono::microseconds>(end - start).count()
        << " microseconds\n";
    if(i == v.end()) return {};
    return *i;
}
```

Have we changed the external interface at all?

Demo: Time a function - External Contained Type

```
template <class ContainedType>
std::optional<ContainedType> timeFindMax(std::vector<ContainedType> const& v)
{
    auto start = std::chrono::steady_clock::now();
    auto i = std::max_element(v.begin(), v.end());
    auto end = std::chrono::steady_clock::now();
    std::cout << "Duration: "
        << std::chrono::duration_cast<std::chrono::microseconds>(end - start).count()
        << " microseconds\n";
    if(i == v.end()) return {};
    return *i;
}
```

What rules does ContainedType have to obey?
What if I want to time finding the max in another STL container? A custom container?

Demo: Time a function - Container Agnostic

```
template <class ContainerType>
std::optional<typename ContainerType::value_type> timeFindMax(ContainerType const& v)
{
    auto start = std::chrono::steady_clock::now();
    auto i = std::max_element(v.begin(), v.end());
    auto end = std::chrono::steady_clock::now();
    std::cout << "Duration: "
        << std::chrono::duration_cast<std::chrono::microseconds>(end - start).count()
        << " microseconds\n";
    if(i == v.end()) return {};
    return *i;
}
```

What rules does ContainerType have to obey?
Do we still have baked in implementation details? What are they?

Demo: Time a function - Functional Programming

```
template <class FunctionType>
auto timeFunction(FunctionType&& func)
{
    auto start = std::chrono::steady_clock::now();
    auto i = func();
    auto end = std::chrono::steady_clock::now();
    std::cout << "Duration: "
        << std::chrono::duration_cast<std::chrono::microseconds>(end - start).count()
        << " microseconds\n";
    return i;
}
```

What rules does FunctionType have to obey? What can FunctionType be?
Notice how types are defined by their interface.

Demo: Time a function - Log Callback

```
template <class LogCallbackType, class FunctionType>
auto timeFunction(LogCallbackType&& log_callback, FunctionType&& func)
{
    auto start = std::chrono::steady_clock::now();
    auto i = func();
    auto end = std::chrono::steady_clock::now();
    log_callback(end - start);
    return i;
}
```

Any implementation details left?


THIS ABSTRACTION NEEDS
MORE ABSTRACTION
quickmeme.com

Demo: Time a function - Variadic Template

```
template <class LogCallbackType, class FunctionType, class... ArgsTypePack>
auto timeFunction(LogCallbackType&& log_callback, FunctionType&& func, ArgsTypePack&&... args)
{
    auto start = std::chrono::steady_clock::now();
    auto i = func(std::forward<ArgsTypePack&&>(args)...);
    auto end = std::chrono::steady_clock::now();
    log_callback(end - start);
    return i;
}
```

Demo: Time a function - Comparison

```
template <class LogCallbackType, class FunctionType, class... ArgsTypePack>
auto timeFunction(LogCallbackType&& log_callback, FunctionType&& func, ArgsTypePack&&... args)
{
    auto start = std::chrono::steady_clock::now();
    auto i = func(std::forward<ArgsTypePack&&>(args)...);
    auto end = std::chrono::steady_clock::now();
    log_callback(end - start);
    return i;
}
```

```
std::optional<int> timeFindMax(std::vector<int> const& v)
{
    std::chrono::time_point start = std::chrono::steady_clock::now();
    std::vector<int>::const_iterator i = std::max_element(v.begin(), v.end());
    std::chrono::time_point end = std::chrono::steady_clock::now();
    std::cout << "Duration: "
        << std::chrono::duration_cast<std::chrono::microseconds>(end - start).count()
        << " microseconds\n";
    if(i == v.end()) return {};
    return *i;
}
```

Generic vs Specific Code Design

Generic

- Code has an extremely long lifetime
- Heavy abstraction leads to easily testable components
- Freeing for developers
- Fewer bugs?
- More readable

Specific

- Faster to develop?
- Simpler
  - Humans learn from examples
  - Example -> Abstraction
  - For example: Parenting
- More specific compiler errors

Demo: Time a function - OOP Implementation - Base

```cpp
template <class CallableType>
class Timeable {};

template <class ReturnType, class... ArgsTypePack>
class Timeable<ReturnType(ArgsTypePack...)>
{
public:
    auto timeIt(ArgsTypePack&&... args)
    {
        auto start = std::chrono::steady_clock::now();
        auto retVal = timed_function(std::forward<ArgsTypePack&&>(args)...);
        auto end = std::chrono::steady_clock::now();
        log_callback(std::chrono::duration_cast<std::chrono::nanoseconds>(end - start).count());
        return retVal;
    }
private:
    virtual ReturnType timed_function(ArgsTypePack...) = 0;
    virtual void log_callback(long long) = 0;
};
```

Demo: Time a function - OOP Implementation - Derived

```
template <class ContainerType>
class GetMaxElement : public Timeable<typename ContainerType::value_type(ContainerType const&)>
{
public:
    typename ContainerType::value_type operator()(ContainerType const& v)
    {
        return *std::max_element(v.begin(), v.end());
    }
private:
    virtual typename ContainerType::value_type timed_function(ContainerType const& v) override final
    {
        return this->operator()(v);
    }
    virtual void log_callback(long long d) override final
    {
        std::cout << "Nanoseconds: " << d << "\n";
    }
};
```

Demo: Time a function - OOP Implementation - Usage

```
auto v = std::vector<int>{1,2,3,4,5,6,7,8,9};
auto timeableMaxElementObject = GetMaxElement<decltype(v)>();
timeableMaxElementObject.timeIt(v);
```

Functional vs Object Oriented Programming

Functional

- General to all types of “Callables”
- Less code bloat
- Easier to read?
- Relies heavily on C++ type deduction
- The interface is extremely ‘light’
- Less or equally complex?

OOP

- Thoughts?

C++20
Program in Concepts

Programming Scenario - Brainstorm

- You work for a company that specializes in software for image processing.
- Your task (should you choose to accept it/it's your job):
  - Develop a new pipeline for RGB image data
  - Your pipeline should perform the following functions in order:
    - Convert the image to grayscale.
    - Calculate the histogram of the image
    - Return the max element of the histogram

How would you generalize to solve the entire family of problems related to this one and be a great asset to your company?


The Specification
Just Changed!

Programming Scenario - New Specification

- Your task:
  - Develop a new pipeline for grayscale image data
  - Your pipeline should perform the following functions in order:
    - Determine if it is a picture of a cat

Does your design still satisfy the specification?

How extensive are the changes needed to meet the new specification?

Demo: Pipeline - Interface

```
template <class DataType, class... FunctionTypePack>
auto pipeline(DataType const& d, FunctionTypePack&&... funcs)
{
    return pipeline_impl(d, std::forward<FunctionTypePack&&>(funcs)...);
}
```

Demo: Pipeline - Implementation

```
template <class DataType>
auto pipeline_impl(DataType const& d)
{
    return d;
}

template <class DataType, class FirstFunctionType, class... FunctionTypePack>
auto pipeline_impl(DataType const& d, FirstFunctionType&& func, FunctionTypePack&&... funcs)
{
    return pipeline_impl(func(d), std::forward<FunctionTypePack&&>(funcs)...);
}
```

Demo: Pipeline - Usage

```
auto itIsACat = pipeline(
        cat_data,
        std::bind(dotproduct, std::cref(weights), std::placeholders::_1),
        std::bind(add, std::placeholders::_1, bias),
        sigmoid,
        isCat);
```

The End!
Questions?

Joshua Smith

joshua.smith4@aggiemail.usu.edu

https://github.com/joshua-smith4/HackTheU_Workshop_Generic_Programming