Programming Language Design

12/30 Generic Programming and Metaprogramming

Let's back to software reuse...

- A variety of approaches
  - From source to binary
- Library, examples include
  - UNIX system interface
    - *Providing OS services through a set of system calls*
  - LAPACK (Linear Algebra PACKage)
    - *Numeric computation library*
  - Xlib: interface to X Window System protocol
    - *Window management library*

However, they use specified interfaces

- Support a pre-determined set of types
- For example, sqrt is a function in C library
  - To calculate square root
  - `float sqrtf(float f);`
    → for floating point number
    `double sqrt(double d);`
    → for double-precision floating point number
- Suppose now we need a sqrt for quad-precision floating number?

Even we have overloading

- For example, in C++ we can have

```
float sqrt(float f);
double sqrt(double d);
```

- But we still have to define them individually!

- Furthermore, it doesn't support the type defined by user!
  - E.g. sqrt for quad-precision floating number

Generic programming

- A programming paradigm
  - A way to organize your program
  - Generalize software components
    - Find common things among similar implementation that share them same algorithm
  - Recap: Parametric polymorphism
    - When we talked about functions and types

Reference

- *David R Mussery and Alexander A Stepanov. Generic Programming, 1988.*

Recap: Parametric polymorphism

- An example: drop the first element in a List
  - When calling drop on a List, the precise input type never comes into play
    - *Acts uniformly on a range of types*

```
def drop[A](l: List[A]): List[A] = l.tail

val result1 = drop(List(1, 2, 3))
val result2 = drop(List("one", "two", "three"))
```

- Also known as generics

Generic!

– How to make programming generic

– Write the definition once,
and allow to use it for all types

→ Even for user-defined types

At an abstract or generic level

- Parameterized procedural schemata
  - completely independent of the underlying data representation
- We usually think of algorithms in a generic way
  - But here we emphasize structuring a software library founded on this idea

Generic programming (cont.)

- The idea of abstracting from concrete, efficient algorithms to obtain generic algorithms
  - Can be combined with different data abstraction
- E.g. a class of generic sorting algorithms which work with sequences
  - Can be instantiated in different ways
  - Works on arrays or linked lists

Generics in Java

- Java supports generic programming since 5.0
  - Based on Generic Java (GJ)

- Before generics
- Need explicit casting when using collections

```
ArrayList l = new ArrayList();
l.add("hello");
          :
String s = (String) l.get(0);
```

Reference

- *Gilad Bracha, Martin Odersky, David Stoutamire, and Philip Wadler. Making the future safe for the past: Adding genericity to the Java programming language, in OOPSLA'98.*

Without generics? Not just annoying

- Might result in runtime error!
  - *Compilers cannot help you*
  - *Exceptions will be raised at runtime*

```
ArrayList l = new ArrayList();
l.add("hello");
        :
Integer i = (Integer) l.get(0);  // !!
```

With Java Generics

- Now such errors can be found at compile-time
  - *since types are specified*

-
```
ArrayList<String> l = new ArrayList();
l.add("hello");
          :
String s = l.get(0);
Integer i = l.get(0);  // compile error!
```

Collections in Java

- Similar to Standard Template Library in C++
  - A lot of reusable data structures
- java.util.Collection
  - List, Map, Set, Stack, Queue
- Allow you to use them with concrete types
- Before 5.0, they are provided with the type Objects and need casting

Type wildcard

- Wildcard (?) can be used in type parameters
  - E.g. List<?> is the supertype for all parameterizations of the generic type
  - Unbounded wildcards

-
```
List<Integer> il = new List<Integer>();
List<?> al = il;
```

Upper bound and lower bound

- Upper bound
  - Use ? along with extends T
  - The given type must be a subclass of T
  - ```
    List<Integer> il = new List<Integer>();
    List<? extends Number> nl = il;
    ```
- Lower bound
  - Use ? along with super T
  - The given type must be a superclass of T
  - ```
    List<Number> nl = new List<Number>();
    List<? super Integer> il = nl;
    ```

Diamond operator

- After Java 7, we can simply omit the type parameters in <>
- For such an annoying case
  - `Map<String, List<String>> m = new HashMap<String, List<String>>();`
- Now it can be
  - `Map<String, List<String>> m = new HashMap<>();`

→ Leave it to compilers

Recap: In Java, generics is invariant

- Generics in Java
  - Let class/interface be parameterized over types

```
ArrayList<String> strs = new ArrayList<String>();
ArrayList<Object> objs = strs;    // error!
```

  - error: incompatible types: ArrayList<String> cannot be converted to ArrayList<Object>

```
// if we allow it...
objs.add(new Integer(1));
String str = strs.get(0);    // what is it??
```

Generics in Scala

- Type bounds are declared with
  - Upper bound: T <: A
    - *T refers to a subtype of A*
  - Lower bound: T >: A
    - *T refers to a supertype of A*
- Variances can be defined as well
  - Use “+” or “–” to denote

Variances in generic classes in Scala

- `class Stack[T]`
  - Invariant subtyping regarding the type parameter T
- `class Stack[+T]`
  - T is used only in covariant positions
    - *Stack[T] is a subtype of Stack[S] if T is a subtype of S*
- `class Stack[-T]`
  - T is used only in contravariant positions
    - *Stack[S] is a subtype of Stack[T] if T is a subtype of S*

C++ Standard Template Library

- An example of generic programming

- A variety of data containers and algorithms
  - Can be applied to either built-in or user types
  - Allow their composition
- STL is a generic library
  - Its components are heavily parameterized
  - Almost every component is a template

Template in C++

- A feature that allows functions and classes to operate with generic types
  - Compile-time feature

- Function templates
- Class templates
- Variable templates
  - after C++14

Function template

- `template <typename id> function_decl`
- For example, define a function template for getting maximal value

Or define with `template <class T>`

```
template <typename T>
T max(T a, T b) {
  return a > b ? a : b;
}
```

Use our max function

- Specify the concrete types
-
```
int main() {
    std::cout << max<int>(6, 4) << std::endl;
    std::cout << max<double>(4.4, 6.6) << std::endl;
}
```

- Or without the type enclosed in angle brackets
  - Compilers can deduce the type if it is not ambiguous
-
```
    std::cout << max(6, 4) << std::endl;
    std::cout << max(4.4, 6.6) << std::endl;
```

What the compiler actually does?

- For the two function calls
- ```
  std::cout << max(6, 4) << std::endl;
  std::cout << max(4.4, 6.6) << std::endl;
  ```
- The compiler generates the follows for us
  - ```
    int max(int a, int b) {
      return a > b ? a : b;
    }
    ```
  - ```
    float max(float a, float b) {
      return a > b ? a : b;
    }
    ```
    ```

Multiple template parameters

- E.g. check the equality of two values
-
```
template <class T, class U>
bool equal(T a, U b) {
    return a == b;
}
```

- Use our equal function
-
```
int main() {
    std::cout << equal(6, 6.0) << std::endl;
    std::cout << equal(0, true) << std::endl;
}
```

Without such templates…

- We have to define all functions for every combination!
  - `bool equal(int a, int b) { … }`
  - `bool equal(int a, bool b) { … }`
  - `bool equal(int a, float b) { … }`
  - `bool equal(int a, double b) { … }`
  - `bool equal(bool a, int b) { … }`
  - `bool equal(float a, bool b) { … }`
  - `bool equal(double a, float b) { … }`
  - `bool equal(bool a, bool b) { … }`
  - :

Non-type arguments are available

- Allow to use constant expression in template arguments
  - Note that we cannot pass a variable since it is determined at compile-time

```
template <class T, int N>
bool equal(T a) {
    return a == N;
}

int main() {
    std::cout << equal<double, 6>(6.1) << std::endl;
    std::cout << equal<bool, 0>(false) << std::endl;
}
```

- *It looks like a function, but actually generates two functions*

Class template

- Define a family of classes
  - *template <parameter-list> class_decl*
- E.g. define a generic stack class

```
template <class T, int N>
class stack {
private:
    T values[N];
    int top;
public:
    stack() {
        top = 0;
    }
    bool push(T v) {
        if (top >= N)    return false;
        values[top++] = v;
        return true;
    }
    T pop() {
        if (top >= 0)    return values[--top];
        return 0;
    }
};
```

Use our generic stack

- It's easy to define stacks with different types
- int main() {

```
    stack<int, 10> is;
    std::cout << is.push(6) << std::endl;
    std::cout << is.push(4) << std::endl;
    std::cout << is.pop() << std::endl;
    std::cout << is.pop() << std::endl;

    stack<float, 5> fs;
    std::cout << fs.push(2.2) << std::endl;
    std::cout << fs.push(6.6) << std::endl;
    std::cout << fs.pop() << std::endl;
    std::cout << fs.pop() << std::endl;
}
```

Variable template

- Define a family of variables or static data members
  - template <parameter-list> variable_decl
- E.g. define a pi variable along with function template

```
template <class T>
constexpr T pi = T(3.14159265358979323385);

template<class T>
T circular_area(T r) {
        return pi<T> * r * r;
}
```

Yield to different values

- According to the concrete type
-
```
int main() {
    std::cout << circular_area(5.0) << std::endl;
    // 78.5398
    std::cout << circular_area(5) << std::endl;
    // 75
}
```

- You might need to specify the version of c++ standard by giving the following flag:

```
-std=c++14
```

Macros in C

- Macros are often used in C language for some sort of generic programming
  - Object-like macros
  - Function-like macros
- Expanded by preprocessors
  - Inside lexer
  - Does not know anything about keywords!

Object-like macros

- A simple identifier which will be replaced by a code fragment
  - Looks like a data object

- E.g. define pi
- `#define PI 3.1415926`
- Use it like a variable or constant
-
```
float circular_area(float r) {
    return PI * r * r;
}
```

Function-like macros

- Define a macro, the use of which looks like a function call

- E.g. define some sort function to get the larger value
  - `#define MAX(a, b) (((a) > (b)) ? (a) : (b))`
- Use this macro
  - `printf("max: %f\n", MAX(5, 5.1));`

Macros are simply expanded…

- It can do anything replacement arbitrarily…
  - Of course performance is pretty good
    - *E.g. PI is simply replaced with the constant*
- It is also possible to define an EDSL with macros!
  - Embedded Domain-Specific Language
- However, the use of macros is discouraged!
  - No type check

→ Use it only if you know what you are doing!

Powerful but very dangerous

- No type check will be performed!
  - Cannot benefit from type check
- Difficult to debug…
  - basically error message is for the code after replacement
- Nowadays there are other solutions
  - Templates, generics, constants, …

Another kind of macros

- Actually the macros we saw is just one kind of macros
  - When people talk about macros
    - *Not necessarily the macros in C*
- The term “macros” means
  - An instruction to perform a task
- Is there any other “smarter” macro system?

→ Syntactic macros

Syntactic macros in Lisp

- Instead of simply replacement, it performs at the level of AST
  - Lisp expression will be passed to the macro Macro can do arbitrary computation
  - The result of this call is also Lisp code
- The macro language is Lisp itself
  - While C macros doesn't know how C works

→ Can be used to extend the language itself

Reference

- *ftp://ftp.gnu.org/old-gnu/Manuals/elisp-manual-20-2.5/html_chapter/elisp_13.html*

Examples of macros in Lisp

- Define a trivial macro
  - `(defmacro five () (+ 2 3))`
  - Use it in an expression
    - `(five)`
  - It is equivalent to use 5 since the computation is done at compile-time
- We may also quote it to prevent from evaluation
  - `(defmacro five () '(+ 2 3))`
  - `(five)` will return `(+ 2 3)`

Hygiene problem

- Hygiene
  - *The practice of keeping yourself and your living and working areas clean in order to prevent illness and disease*

--- Oxford Advanced Learner's Dictionary

Hygiene problem (cont.)

- Occur when expanding a macro (either in C or Lisp)
  - Might not be hygienic when variables are used inside macros
  - Variable bindings performed in macro expansion will hide existing variable bindings

- For example, a variable named “i” that is declared in a macro will hide the “i” declared outside

Hide existing variables bindings

- Suppose we define a loop macro to repeatedly evaluate the given expression

```
#define LOOP(n, expr) do { for (int i=0; i<n; i++) { expr; } } while(0)
```

  - *do-while is used to make syntax acceptable…*

- Then we use such a macro

```
int main() {
    int i = 100;
    LOOP(5, printf("hello! %d\n", i));
}
```

  - It prints out “hello! 0”, “hello! 1”, …
    rather than “hello! 100”, “hello! 100”, …

Hygienic macros

- Use a symbol that is guaranteed not to appear anywhere else in the code
  - Prevent accidental capture
- Supported in Scheme, Racket, Rust, etc.

What is meta?

higher; beyond

--- Oxford Advanced Learner's Dictionary

beyond or at a higher level

--- Longman Dictionary of Contemporary English

Examples of “meta-”

- Metaphysics
  - The branch of philosophy that deals with the nature of existence, truth and knowledge
- Metalanguage
  - The words and phrases that people use to talk about or describe language or a particular language

--- Oxford Advanced Learner’s Dictionary

Examples of “meta-” (cont.)

- Metafiction
  - A type of play, novel, etc. in which the author deliberately reminds the audience, reader, etc. that it is fiction and not real life

--- Oxford Advanced Learner’s Dictionary

Metafiction

– For example, in a scene that the hero and his friend are fighting against a lot of enemies…
One of the hero's friends sighs:

"We shouldn't waste time here…
The number of enemies never decrease
unless we defeat the boss."

"Why??"

"It is just such a system."

"System? What system?"

"Never mind…"

Metafiction (cont.)

- Given a one-hour drama
  - Every week there is a mystery and the hero has to solve it…
  - In a scene after 45 minutes past,
    the hero suddenly says:

"Hey, we have to hurry!
We have to solve it within 15 minutes!"

"Why? Why 15 minutes?"

"Well…we can't put off until next week"

Examples of “meta-” (cont.)

- Metadata
  - Information that describes other information in order to help you understand or use it

--- Oxford Advanced Learner’s Dictionary

Metadata

- Data for photo files
  - E.g. EXIF information such as shooting parameters and geolocation
- Data for music files
  - E.g. ID3 tag containing title, artist, album, etc.

Metaprogramming

- To write programs that treat programs as data
  - Programs manipulate data

- A way to write and organize your program
  - Programming paradigm
  - A technique for implementation

Why metaprogramming?

- High programming productivity
  - More generative
    - *Minimize the number of lines of source code*
  - Self-adaptive at runtime
    - *Without recompilation*
  - Especially for library developer

Why metaprogramming? (cont.)

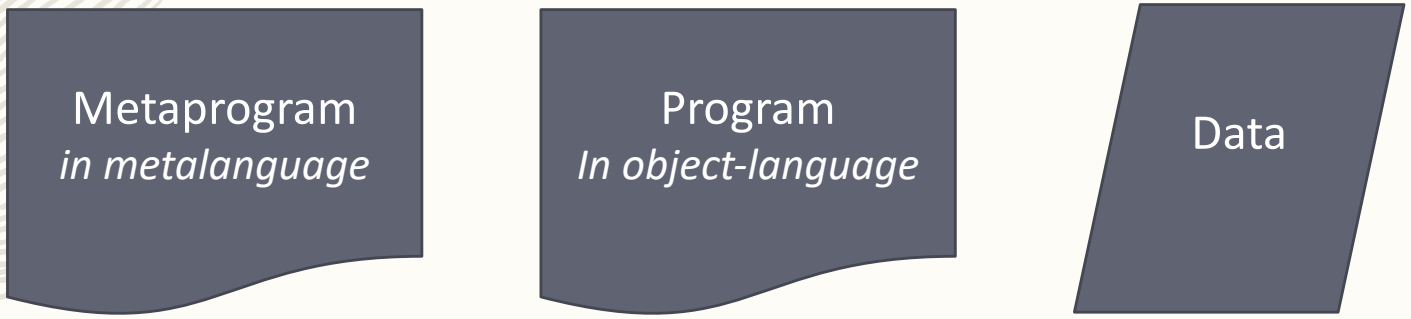
- E.g. make library support transparent
  - Automatically insert a call to a specific method before calling certain methods
  - Might remind you of AOP?
- E.g. optimize the performance
  - Implicitly transform the code in certain pattern into another pattern to get better performance
  - Might remind you of DSL?

Why metaprogramming? (cont.)

- You might also want to do more than implementing generic algorithms
  - To make your code extremely short
  - Transform the code according to code pattern
- Maybe not a good practice…
  - Since it might not base on algorithms
  - It might break the rules in the language

Metaprograms and object-programs

- Metaprograms manipulate object-programs
  - Object-programs manipulate data
- Metaprograms are written in metalanguages
  - Object-programs are written in object-languages



Don’t confuse with…

- Some concepts related to “meta”

- Type introspection
  - Knowing types and properties at runtime
  - Subset of reflection
- Reflection
  - Manipulate them at runtime
  - A strategy to do metaprogramming

Type introspection

- Just knowing objects' information at runtime, that's all
  - E.g. get, check, and safely cast the type of an object
  - Cannot modify them
- Runtime type information (RTTI) in C++
- instanceof in Java

RTTI in C++

- Was added to C++ language due to the needs in class libraries
  - Every vendor was implementing it

→ Cause incompatibilities between libraries

→ Support at language-level

Reference

- https://msdn.microsoft.com/en-us/library/b2ay8610.aspx

RTTI in C++ (cont.)

- To use it, usually we need to add a flag to enable it in compliation
  - `dynamic_cast` operator
    - *For conversion of polymorphic types*
  - `typeid` operator
    - *For identifying the exact type of an object*
  - `type_info` class
    - *Hold the type information returned by* `typeid`

Reference

- https://msdn.microsoft.com/en-us/library/b2ay8610.aspx

Reflection

- Examine or modify runtime behavior
  - Invoke a method or instantiate an object without knowing its name at compile time
  - Monitor or decide the execution at runtime
- Enable applications to perform operations which would otherwise be impossible
- Reflection API in Java

Reference

- https://docs.oracle.com/javase/tutorial/reflect/index.html

Reflection API in Java

- JVM instantiate an immutable instance of `java.lang.Class` for every object
  - Access class declaration information such as class modifiers and types
  - List constructors, fields, methods, and nested classes

- No free lunch!
  - Performance overhead

Reference

- https://docs.oracle.com/javase/tutorial/reflect/index.html

Reflection API in Java (cont.)

- Should avoid using it if there is another solution
  - Since it might break some restrictions in Java
    - *E.g. access private fields and methods*
- Why we need it?
  - To implement debugger and development environments
  - To implement a flexible test suite

Now back to metaprogramming…

- How could we do “meta” things?
  - To define things that define things
- In some programming languages
  - Expressions can be evaluated or quoted for being evaluated later
  - E.g. Lisp

In OO languages

- Objects are instances of classes
  - Classes define the structure and behavior of their object instances
- How to allow programmers to define how classes look like?
  - Give the things, the instances of which are classes → classes of classes

`java.lang.Class`

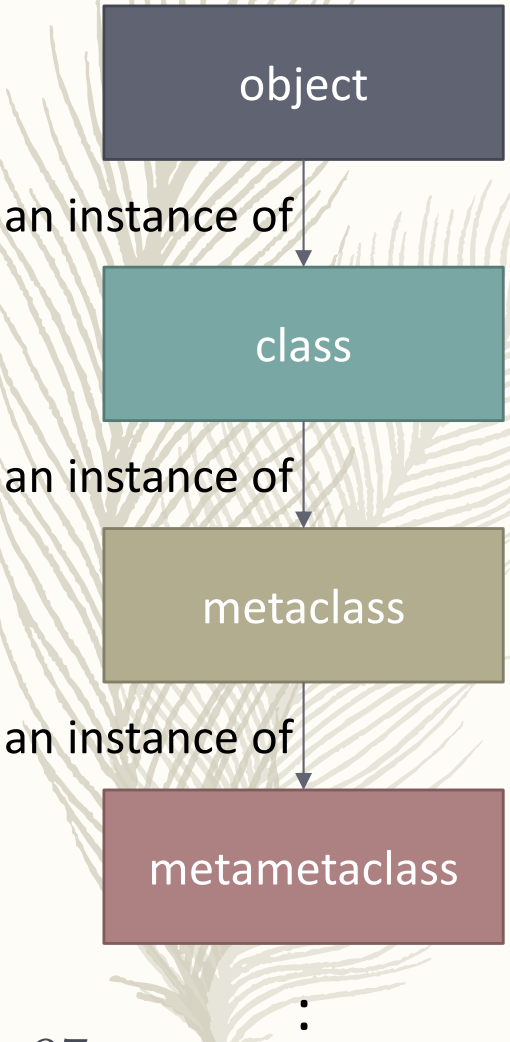
- Remember `java.lang.Class` in Java
  - Which describes the representations of classes
- It's the only "class of classes" in Java
  - For all classes, and no way to customize it
- Is it possible to allow programmers to define how these classes of classes look like?
  - In some languages, yes
  - E.g. the default one in Python is `type`, programmers can replace it with a subclass of `type`
  - Common Lisp Object System (CLOS), OpenC++, etc.

Think about classes of classes

- Classes whose instances are classes are called metaclasses
  - Objects are instances of classes
  - Classes are instances of metaclasses
  - Metaclasses are instances of what?

→ metametaclasses??

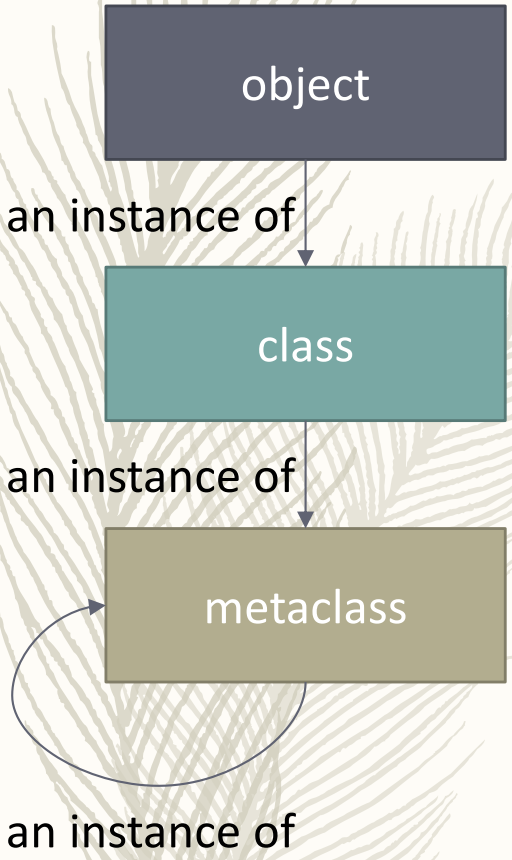
“Turtles all the way down”



- i.e. infinite regression
- A mythology says that Earth is actually flat and supported on the back of a World Turtle
  - *Supported by a larger World Turtle, …*

  → *What the final one is standing on?*

Unmoved mover



- To explain all the motion in the universe
  - A concept in philosophy
  - Deal with “turtles all the way down”?
- Metaclasses for all the meta-level classes
  - Metaclasses are instances of themselves

Metaobject Protocol

- Metaobjects: the objects manipulating objects

- Metaobject protocol
  - How metaobjects behave
  - How to customize metaobjects

- Use of first-class metaclasses is the first step

Reference

- *Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. The Art of Metaobject Protocol, 1991.*
- Richard P. Gabriel. *A Review of The Art of the Metaobject Protocol, 2010.*

Example1: in OpenC++

– Suppose we have such a class

```
Node* get_next_of_next(Node* p)
{
  Node* q = p->next;
  return q->next;
}
```

– To inherit a property “persistent”, users have to extend the class PersistentObject in a library

```
class Node : public PersistentObject {
public:
  Node* next;
  double value;
};
```

Reference

- *Shigeru Chiba. A Metaobject Protocol for C++, in OOPSLA’95.*

Example1: in OpenC++ (cont.)

– If such a property is supported at language-level, we can simply add a keyword

```
persistent class Node : public PersistentObject {
public:
 Node* next;
 double value;
};
```

– However, it is provided as a library, so we have to call Load() every time

```
Node* get_next_of_next(Node* p)
{
   Node* q = (p->Load(), p->next);
   return (q->Load(), q->next);
}
```

Example1: in OpenC++ (cont.)

- OpenC++ allow library developers to define

```
Expression
PersistentClass::CompileReadDataMember(Environment env,
                                       String member_name,
                                       String variable_name)
{
  return MakeParseTree("(Load(%s), %e)",
                       member_name,
                       Class::CompileReadDataMember(…));
}
```

- Then library users can simply write

```
metaclass Node : public PersistentClass {
class Node {
public:
  Node* next;
  double value;
};

_
Node* get_next_of_next(Node* p)
{
  Node* q = p->next;
  return q->next;
}
```

Runtime? Compile-time?

- Metaobjects may work at runtime
  - Like how reflection does
  - Which are responsible for runtime behavior and allow users to change and tune
  - Benefit from runtime information but have performance penalty
- Or exist only at compile-time
  - Control the compilation of programs

Other approaches

- Syntactic macro
  - Expanded and evaluated at compile-time
  - E.g. Lisp, Scheme, etc.
- Template programming
  - Generate code based on template at compile-time
  - E.g. template metaprogramming in C++

Example2: in C++

- Without template, to calculate factorial

```
int factorial(int n) {
  if (n == 0)
    return 1;
  return n * factorial(n - 1);
}

int main () {
 std::cout << factorial(6) << std::endl;  // 720
}
```

Example2: in C++ (cont.)

- Calculate factorial at compile-time using template

```
template <int n>
struct Factorial
{
  enum { value = n * Factorial<n-1>::value };
};

template <>
struct Factorial<0>
{
  enum { value = 1 };
};

auto main() -> int
{
 std::cout << Factorial<6>::value << std::endl;
}
```

Example3: inline expansion

- Expand loop inline to get better performance

```
template<int dim>
Vector<dim>&
Vector<dim>::operator+=(const Vector<dim>& rhs)
{
  for (int i = 0; i < dim; ++i)
    value[i] += rhs.value[i];
  return *this;
}
```

- Then at compile-time the use of Vector<2> becomes

```
template<>
Vector<2>&
Vector<2>::operator+=(const Vector<2>& rhs) {
  value[0] += rhs.value[0];
  value[1] += rhs.value[1];
  return *this;
}
```

Other approaches (cont.)

- Multi-stage programming
  - Runtime code generation and program execution
  - Instead of having compile-time and runtime
  - Type systems should be designed to statically ensure that
    - *Dynamically generated programs are type-safe*
  - E.g. MetaML, MetaOCaml, etc.

Some concepts smell “meta”

- Partial evaluation
  - Evaluate parts of the program or expressions to get better performance
    - *Source-to-source transformation*
  - Via program specialization
    - *For specific cases, some expressions can be always evaluated to something*
  - E.g. we know that a variable holds a particular value in this case and thus transform it to a constant value

Some concepts smell “meta” (cont.)

- Dependent type
  - A type that depends on a value
  - E.g. given a function returns a 2D matrix type
    - *The matrix type clearly specify its size: m*n*
    - *Prevent bugs by moving the numbers to type checking*
  - Supported in Agda, ATS, Coq, etc.
  - Note that the type in C++ template does not accept variable

Challenge…

- Usually the code is
  - Difficult to understand and debug
    - *Since thinking programming at meta-level*
  - Very useful especially for library/tool support
  - Basically not for “usual” use

- So in some languages
  - Metaprogramming is partially supported, e.g. reflection
  - Metaprogramming is used to implement a paradigm

Next lecture

1/6 **Practice 8**