ECE326

PROGRAMMING LANGUAGES

Lecture 8 : Reflective Programming

Kuei (Jack) Sun

ECE

University of Toronto

Fall 2020

First Class Citizen

- Can do everything that other entities can
- Without restrictions:
  - Can be used
    - E.g. assigned to variable, passed to function
  - Can be constructed
    - Especially in a local scope
  - Have a type

First Class Citizen

- Can do everything that other entities can
  - Example
    - **Can be used**
    - Can be constructed
    - Have a type
- E.g. Integer (in C++)
  - Can be assigned to a variable ✓

    ```
    x = 5;
    ```

  - Can be passed to or return from a function ✓

    ```
    int foo(int a);
    ```

First Class Citizen

- Can do everything that other entities can
  - Example
    - Can be used
    - **Can be constructed**
    - Have a type
- E.g. Integer (in C++)
  - Can be constructed in local scope ✓

```
int x = 88;
```

First Class Citizen

- Can do everything that other entities can
  - Example
    - Can be used
    - Can be constructed
    - **Have a type**
- E.g. Integer (in C++)
  - The type of Integer is `int` ✓

```
int x = 88;
```

Not First Class

- E.g. Function (in C++)

  ```
  int foo(char a) { ... }
  ```

  - Can be used ✓

    ```
    x = foo; // x is a function pointer
    ```

  - Can be constructed ✗

    ```
    // cannot create new function at runtime
    ```

  - Have a type ✓

    ```
    int (* x)(char) = foo; // foo is of type int(*)(char)
    ```

First Class Citizen

- Objects are first class in Python
- Everything in Python is an *object*
  - first class citizen!
    - Exception: reserved words and operators
- E.g. type is first class in Python

```
>> a = int
>> a()
0
>> type(int)
<type ‘type’>
```

Introspection

- The ability to examine the type or attribute of a value
  - At runtime
- Python examples
  - `isinstance(object, cls)`
    - Checks if object is an instance of class

```
class A : pass
class B(A) : pass

>> obj = B()
>> isinstance(obj, B)
True
```

```
>> isinstance(obj, A)
True
>> isinstance(obj, int)
False
>> isinstance(A, B)
False
```

Introspection

- issubclass(cls$_1$, cls$_2$)
  - Checks if class is a subclass of another

```
class A : pass
class B(A) : pass

>> issubclass(B, A)
True
```

```
>> issubclass(A, B)
False
>> obj = B()
>> issubclass(obj, B)
TypeError: obj must be a class
```

- dir(object=None)
  - Returns a list of object’s attributes

```
>> dir(A)
[‘__init__’, ‘__class__’, '__delattr__', '__dict__', …]
```

Introspection

- `hasattr(object, name)`
  - Checks if string *name* is the name of one of the object’s attribute

```
class A:
  x = 5
  def foo(): pass

>> hasattr(A, ‘x’)
True
```

```
>> hasattr(A, ‘y’)
False
>> my_name = ‘foo’
>> hasattr(A, my_name)
True
```

- `type(object)`
  - Returns type of object

```
>> a = A()
>> type(a)
<class ‘__main__.A’>
```

```
>> type(A.foo)
<class ‘function’>
>> type(a.foo)
<class ‘method’>
```

Introspection

- C++ Example
  - Runtime Type Information (RTTI)
  - `typeid`
    - Returns the type id of an object

```
if (typeid(Student) == typeid(*object)) {
    return hash_student(object);
}
```

  - `dynamic_cast`
    - Downcasts a base class pointer to a subclass pointer, if valid

```
Animal * ap = animals.pop();
Lion * lp = dynamic_cast<Lion *>(ap);
```

Static Introspection

- Introspection at compile time
- Treating compiler as a *white box*
  - The compiler reveals what it knows about an entity
    - type, variable, expression, …etc
  - Makes use of how compiler internally represents an entity
- C++ example
  - `decltype`
    - Returns the type of an expression at compile time

```
decltype(7/2) a = (7/2);  // a is of type int
```

Reflection

- The ability for a program to introspect and modify itself
  - Changes its own code, such as structure and behaviour
  - Can even change the programming language itself
    - E.g. syntax, semantic, implementation
- Static reflection
  - Generates compile-time meta-objects
    - E.g. `dir` from Python for C++, only accessible at compile-time

Reification

- Turns abstract representation into concrete data types and/or addressable objects
- Simpler definition
  - Converting compile time types into run-time entities
- Java Example
  - Type information kept to perform runtime type checking

```
String strings[] = {"a", "b"};
Object objects[] = strings;     // allowed at compile time
objects[0] = 5;                 // allowed at compile time

java.lang.ArrayStoreException: java.lang.Integer
```

Type Erasure

- Removal of type information/checks at runtime
  - Type checking at compile-time, none at runtime
- C++ Example

```
struct data {
  int norm;
  int sample[16];
} ;
```

```
int normalized(struct data * d, int i) {
  return d->sample[i] / d->norm;
}
```

```
movslq      %edx, %rdx
movl        4(%rcx,%rdx,4), %eax
cltd
Idivl       (%rcx)
ret
```

Generated code assumes correct structure (struct data) is passed in. No type checking is made at runtime, which improves performance

Python Reflection

- `setattr(object, name, value)`
  - Set an object’s attribute with *name* to arbitrary *value*

```
class A:
  x = 5

>> setattr(A, ‘x’, {1})
>> A.x
{1}
```

```
>> setattr(A, ‘y’, 7)
>> A.y
7
# equivalent to A.z = None
>> setattr(A, ‘z’, None)
```

- `getattr(object, name, default=None)`
  - Retrieves an attribute by name, return *default* if not found.
    If default is not specified, raise AttributeError

```
>> getattr(A, ‘q’, 0)
0
```

```
# equivalent to A.q
>> getattr(A, ‘q’)
```

Python Reflection

- `delattr(object, name)`
  - Delete an object’s attribute by *name*

```
class A:
  x = 5
>> hasattr(A, ‘x’)
True
```

```
# equivalent to del A.x
>> delattr(A, ‘x’)
>> hasattr(A, ‘x’)
False
```

- `globals(), locals(), vars(object)`
  - Returns a dictionary of all global/local/instance variables

```
>> globals()
{'__name__': '__main__', '__doc__': None, '__package__':
None, '__annotations__': {}, '__builtins__': <module
'builtins' (built-in)>, …}
```