EIFFEL and C++

1. C++ VS Eiffel

• Fundamentally different

• Eiffel purely Object Oriented

• C++ supports many methodologies
2. Include Files

- C++ supports the “C” style include of include files

- Eiffel has no include files. Libraries are included as they are used.
3. Preconditions/Postconditions/Invariants

- Eiffel supports preconditions and post conditions for class features:

(example)

```eiffel
thisFunction(a,b:INTEGER) is
  require  --the pre-condition
    a > b
  do
    ...
  end
  ensure --the post-condition
    Result > 0
  invariant
    a_non_negative: 0 <= a
end
```
• C++ has no formal support for such concepts, however the pre-condition and post-conditions could be emulated:

(example)

```cpp
int myClass::thisFunction(int a, int b)
{
    // pre-condition
    if( a > b ) throw an_exception_class;

    ...

    // post-condition
    if( Result > 0) throw an_exception_class;
    return result;
}
```

Note: the invariant cannot easily be duplicated in C++
4. Dynamic vs Static Binding

• C++ defaults to Static Binding
  Dynamic binding in C++ only occurs when a super class has a member method declared as *virtual*.

• Eiffel defaults to Dynamic Binding
5. In-line commands

• C++ allows user created in-line commands

  (example in *.h file)

  ```c++
  class MyClass
  {
  public:

  const memberType getMember() const {return Member;}

  };
  ```

• Eiffel lets the compiler decide what functions should be in-line
6. Operator overloading

• Both C++ and Eiffel Support operator overloading
7. Type Casting

• C++ allows type casting

(example in *.cpp file)

double D;
int I;

I = int(D);

// Result: The double D is truncated to (the largest integer) <= D.

• The developers of Eiffel consider type casting dangerous, and thus it is not supported.
8. Friend functions

• C++ supports class “friends”

(example in *.h file)

class MyClass
{
    public:
        friend ostream&  operator << (ostream&, Complex&);
};

• Eiffel considers the friend concept as contrary to the OO thought process.
9. Garbage collection

• In C++, the *delete* command must be used

(Example in *.cpp file)

```cpp
void MyClass::functionName()
{
    otherClass *A = new otherClass( args...);

    delete A;
    // without this line, repeated calls of MyClass.FunctionName will result in a
    // memory leak.
}
```

• Eiffel has automatic garbage collection.
10. Inheritance

- **In C++:**
  
  \( (example \text{ in } *.h \text{ file}) \)

  ```
  class ChildClass : public ParentClass
  {
      ...
  }
  ```

- **In Eiffel**
  
  \( (example) \)

  ```
  class CHILD inherit PARENT
  
  --features
  
  end
  ```
11. Virtual Functions, Deferred/Abstract Classes

Terminology:

- *Deferred/Abstract Class*- A class that can be instantiated only through a descendant that implements the outlined features of the abstract class.

- *Deferred/Void feature/method*- A function, which is a member of some class, whose declaration is present, but is not defined. The definition must be included in descendants of the given class.
• In C++:

• A method is made virtual using the **virtual** command:

  *(example in *.h file)*

```cpp
class ParentClass
{
    public:
        virtual void displayName( );
};
//----------------------------------------------------------
class ChildClass:ParentClass
{
    public:
        void displayName( ) {cout << Name};
        // NOTE: the function is in-line for convenience
};
```

• The redefined method must have the same signature
• In Eiffel

(example)
class PARENT
feature
    displayName is
defered
end
end

-------------------------------------------------

class CHILD inherit PARENT
    redefine displayName end

feature
    displayName is
do
    ...
end
end
Abstract class In C++: Must include one pure virtual function

- A method is made pure virtual using the `virtual` command, along with "=0":
  
  *(example in *.h file)*

```cpp
class ParentClass
{
    public:
        virtual void displayName() = 0;
}
```

- All derived classes that do not implement a pure virtual function are also abstract
• In Eiffel: a deferred class is created using the deferred command

(example)

defined class PARENT

--features

end
12. Constructors

- C++ has default and copy constructors, as well as constructors with other signatures:

```cpp
// example in *.h file

class MyClass
{
public:
    MyClass(); // default constructor
    MyClass(int a); // constructor with signature
    MyClass(const MyClass&); // copy constructor
};
```

NOTE: “MyClass(int a = 10);” could serve as both a default constructor and a constructor with a signature.
• Eiffel has no constructors per say, but by including features in the “creation” section gives a similar effect

*(example)*

class MYCLASS

creation
  make

feature

  make is
    do
      ...
    end
  end

end
13. Reference Semantics

• In C++, referencing can be difficult:
  (example in *.cpp file)

    MyClass A, *B, **C;

    A=*B; // object assigned  NOTE: “=” operator may need to be overloaded
    B=*C; // reference assigned
    A=**C; // object assigned
    A=&A; // reference assigned
    C=&&A; // reference to reference assigned
In Eiffel, pointers don’t exist. An assignment operator always passes a reference, unless “clone()” is used.

(example)

feature
  thisFunction is

    local a,b:MYCLASS

    do

      create a.make(1)
      create b.make(2)

      a := b  --reference passed
      a := clone(b) -- object passed

    end
14. Templates

- In Eiffel, all “templates” are automatically available in given libraries.

- In C++, the needed template must be included.
  (example in *.cpp file)

```cpp
#include<vector>

void MyClass::MyMethod(args)
{
    vector<int> a;
    vector<double> b;
    vector<MyOtherClass> c;
    MyOtherClass MyObject;

    c.push_back(MyObject); // insert an instance of MyOtherClass into the
    // vector c
}
```
15. The Standard Template Library (STL)

The STL is a useful collection of “tried and true” templates that implement the most commonly used data structures and algorithms. This includes lists, stacks, queues, heaps, trees, and sorting of various kinds.

A good reference is listed below:


Other References for this seminar:

Meyer, Bertrand. *Object-Oriented Software Construction Second Edition*