Lecture 9

Good Practices

Attributes should never be public

encapsulation

Writing better classes

- Capitalize class names
- Method names start with lower case (except constructors / destructors)
- Give special names to members

NEVER start a name with two underscores (reserved for compiler vendors).

```
// We can use "this", implicit pointer to the current object.
void Sample::setval(int val) {
  this->val = val;
}

// Better, we can use the "scope operator"
void Sample::setval(int val) {
  Sample::val = val;
}

// Nothing says that implementation and prototype
// should be strictly identical.
void Sample::setval(int v) {
  val = v;
}

// Having a special name for members removes any ambiguity
// in what is probably the easiest and simplest way.
void Sample::setval(int val) {
  m_val = val;
}
```

```
/*
  Note that C++ supports a special syntax in which
  you can supply after the name of a constructor
  the name of an attribute called like a function;
  it means that it it initialized with this value.
  If initialization is the only thing that the
  constructor does, the body may be empty and no
  other implementation is needed.
*/
class Sample {
  ...
  Sample(int val):m_val(val) {};
  ...
}
```

Object creation / destruction

It's really important to understand in C++ when and how object are created/destroyed, because not understanding the rules can lead to unexpected crashes

Complex classes need to respect some rules

Coplien's Canonical Class

```
class T {
public:
   T();            // Default Constructor
   T(const T&);    // Copy Constructor
   ~T();           // Destructor (may be virtual)
   T &operator=(const T&);
                   // Assignment operator
};
```

Default Constructor

Because of arrays.

As soon as you define a constructor, no default constructor will be created.

You cannot specify any parameter when you create an array of objects. You cannot initialize it like a C array of structures, because attributes are private.

```
class YearToMonth {
  ...
  YearToMonth(short years=0); // Enough to fix the problem
  ...
};
```

Default values are only specified in the method prototype, not in the implementation.

Destructor

Because of heap memory

if the object allocates heap memory in the constructor or at a later stage in its life, this will not be freed by the default destructor and will lead to **memory leaks**.

Copy Constructor

To have a destructor-safe deep copy

Necessary when pointers to heap areas inside the object

- CREATES a new object from a previous one
- Created by default (byte by byte copy)
- Used when objects are passed by value to a function, or returned by a function

```
ObjectType(const ObjectType& original) {
  _name = original._name + "_copy";
  cout << "Creating object " << _name << endl;
}
```

C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do it blows your whole leg off.

Java: garbage collector

Deep copy versus Shallow copy

If we want to be safe, we need a copy constructor that performs a "deep copy", which means that it also allocates and copies anything that the original object points to.

```
Dummy::Dummy(const Dummy& dum) {
  m_sz = dum.m_sz;
  m_tab = new int[m_sz];
  for (int i = 0; i < dum.m_sz; i++) {
    m_tab[i] = dum.m_tab[i];
  }
}
```

Assignment operator

You can redefine operators in C++

Operators in C and C++

`operatorsymbol()`

C++ knows a lot of functions/methods named `operator<symbol>()` which can be used as such or simply as the symbol.

- `operator+()`

Method or Function?

Rule: **if the current object plays the lead part method, otherwise function**

Method

```
class T {
public:
  ostream& operator<<(ostream& os){
    // code here ...
    return os;
  }
}
```

`t.operator<<(cout)` is equal to `t << cout`

- Not the way it is used

- Also, not possible to add a method to `ostream`

Function

```
class YearToMonth {
public:
  // Don't use a namespace in a header file
  friend std::ostream& operator<<(std::ostream& os,
                          const YearToMonth& ytm);
};
```

- `++` **Method** (affecting the current object)
- `+` **Friend function.** Takes two objects, return a third one.

The assignment operator recommended by Coplien **is a method**, because we are only dealing with objects of the class.

Possibly a need for deleting, then recreating memory areas.

System Calls

MOSTLY UNIX/LINUX

`$ man <name>"`

Unix/Linux manual pages

1. General commands
2. **System calls**
3. C library functions
4. File formats

Network Programming

The best part in this lecture! In case you miss it, here is the slide for this lecture.

Slides

…