C++ Naming Conventions

Names are the key to program readability. If the name is appropriate everything fits together naturally, relationships are clear, meaning is derivable, and reasoning from common human expectations works as expected. Good names save time when debugging and save time when extending.

If you find all your names could be Thing and DoIt then you should probably revisit your design. Avoid the temptation to have short names everywhere, and avoid non-standard abbreviations.

Source code is meant to be read by humans. This is the most important thing to remember. As well as communicating your intent to the machine, you must make it clear what that intent is to those who will read the source code. This includes you! Code you've written more than about 3 weeks ago may as well have been written by somebody else.

A cryptic example:

```
Dog d
Lion l

l.dvr(d)
```

A more descriptive version:

```
Dog myPetDog
Lion escapedLion

escapedLion.devour(myPetDog)
```

1 Class Names

Name the class after what it is. If you can't think of what it is that is a clue you have not thought through the design well enough.

- Use upper case letters as word separators, lower case for the rest of a word
- First character in a name must be upper case
- No underscores ('_') are permitted

```
class OdeSolver
class ParameterFile
```

2 Method and Function Names

Usually every method and function performs an action, so the name should make clear what it does: CheckForErrors() instead of ErrorCheck(), DumpDataToFile() instead of DataFile(). This will also make functions and data objects more distinguishable. Each method/function should begin with a verb.

- Classes are often nouns. By making function names verbs and following other naming conventions programs can be read more naturally.
- Suffixes are sometimes useful:
  - Max - to mean the maximum value something can have.
  - Count - the current count of a running count variable.
  - Key - key value.

  For example: RetryMax to mean the maximum number of retries, RetryCount to mean the current retry count.
- Prefixes are sometimes useful:
  - is/has - to ask a question about something. Whenever someone sees Is or Has they will know it's a question.
  - get - get a value.
  - set - set a value.

  For example: `if(HasHitRetryLimit())` ...
- Use the same naming rules as for class names

```
class OdeSolver
{
public:
   int  SolveEquation();
   void HandleError();
}
```

3 No All Upper Case Abbreviations

When confronted with a situation where you could use an all upper case abbreviation instead use an initial upper case letter followed by all lower case letters. No matter what.

Take for example `NetworkABCKey`. Notice how the C from ABC and K from key are confused.

```
class FluidOz             // NOT FluidOZ
class NetworkAbcKey       // NOT NetworkABCKey
```

4 Pointer Variables

Pointers should be prepended by a 'p' in most cases. Place the * close to the variable name rather than the pointer type.

```
String *pName= new String;
String *pName, name, address; // note, only pName is a pointer.
```

5 Class Attribute Names

- Private attribute names should be prepended with the underscore character 'm'.
- After the 'm' use the same rules as for class names.
- 'm' always precedes other name modifiers like 'p' for pointer.

```
class CleaningDepartment{

public:
   int      ComputeErrorNumber();
private:
   int      mCleanHouse;
   int      mErrorNumber;
   String   *mpName;
}
```

6 Reference Variables and Functions Returning References

References should be prepended with 'r'.

This establishes the difference between a method returning a modifiable object and the same method name returning a non-modifiable object.

```
class Test{
public:
   void              TestConveyorStart(StatusInfo& rStatus);

   //returns a modifiable object so requires the 'r' prefix
   StatusInfo&       rGetStatus();

   //returns a constant (non-modifiable) object so doesn't have the 'r' prefix
   const StatusInfo&  GetStatus() const;

private:
   StatusInfo&       mrStatus;
}
```

7 Method Argument Names

The first character should be lower case. All word beginnings after the first letter should be upper case as with class names.

```
class WackyRace{
public:
   int StartYourEngines(  Engine& rSomeEngine,
                          Engine anotherEngine);
}
```

8 Variable Names on the Stack

When variables are created in a C++ program (when the variables are in scope) the memory required to hold the variable is allocated from the program stack, and when the variable goes out of scope, the memory which was taken on the stack is freed. When memory is allocated dynamically (by the programmer - using the `new` keyword), or if variables are declared as class attributes memory is taken from the heap.

- Use all lower case letters
- Use '_' as the word separator.

With this approach the scope of the variable is clear in the code. And now all variables look different and are identifiable in the code.

```
int  ProcessMonitor::HandleError(int errorNumber){
   int            error= OsErr();
   Time           time_of_error;
   ErrorProcessor error_processor;
}
```

9 Global Constants

Global constants should be all caps with '_' separators.

```
 const double TWO_PI = 6.28318531;
```

10 Static Variables

Static variables should be prepended with 's'.

```
private:
   static StatusInfo msStatus;
```