C++ Idioms by Example, lesson 1

Introducing some very basic C++ idioms, rules, guidelines and best practices

Olve Maudal
oma@pvv.org

November 2008

Disclaimer: *most of the items I will discuss here are indeed* ***idioms*** *and* ***conventions*** *of the C++ language, something that most experts will agree on. However some of the items are based on an* ***ecolect*** *– a programming style that is unique for a limited group of developers. There are also a few items based on my* ***idiolect****, which is a programming style that I believe in. And you may argue that one or two things I suggest here are just plain* ***idiotic****...*

My intention is not to tell you how to do C++ programming. I hope to use this presentation as a catalyst for discussion, therefor I have also included some controversial issues.

Ignite your flamethrower. Choose your battles!

Please criticise this program.

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
    int _value;
    int my_magic(int a, int b);
  public:
    Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return _value;
    }
    void print(char* prefix);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) {
    _value = seed;
  }

  int Foo::my_magic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = my_magic(_value, number);
    _value += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << _value << "\n";
  }
}
```

Please criticise this program

Spend 5-10 minutes to read through the code. Imagine that this is a piece of code that you found in a larger codebase. Criticise everything you see,

... apart from the fact that it is a contrived, incorrect and stupid program that does not do anything useful, and is probably better written in Python as `print "the answer=42"` (and do not spend time on the implmentation of my_magic)

```
$ cd ~/myprog
$ g++ foo.cpp main.cpp && ./a.out
the answer=43
```

What do you see in this code?

Happiness?

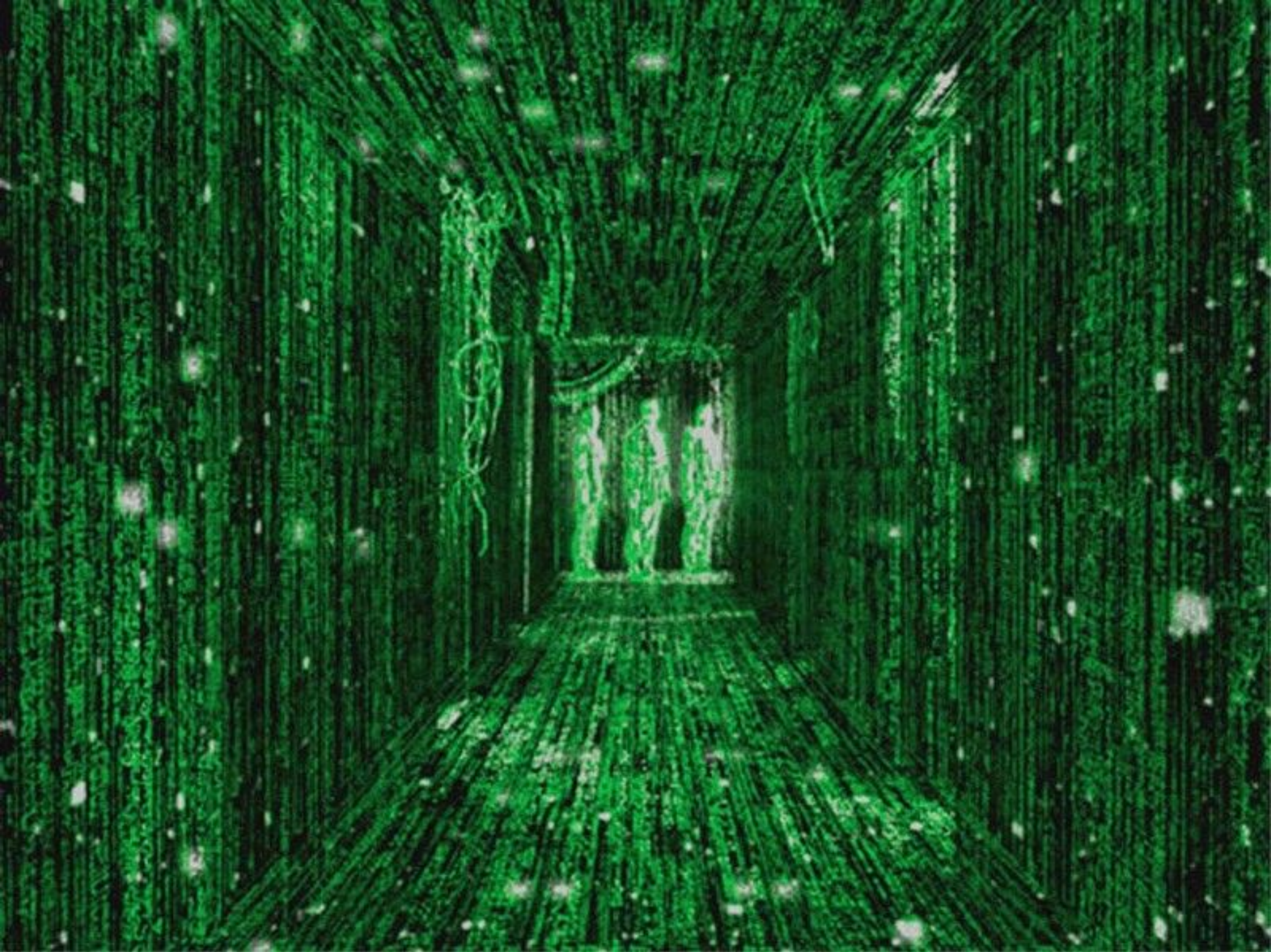
or trouble?

I see trouble.

It looks like the code has been written by someone with little experience in C++, or even worse, by someone who does not care...

Large amount of this kind of code will make your codebase rot.

But, once you start to believe - it is time to defeat the deteriorating agents



~/myprog/foo.hpp

```
namespace bar {
  class Foo {
    int _value;
    int my_magic(int a, int b);
  public:
    Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return _value;
    }
    void print(char* prefix);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) {
    _value = seed;
  }

  int Foo::my_magic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = my_magic(_value, number);
    _value += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << _value << "\n";
  }
}
```

```
$ cd ~/myprog
$ g++ foo.cpp main.cpp && ./a.out
the answer=43
```

1. always compile with high warning level, and treat warnings as errors

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
    int _value;
    int my_magic(int a, int b);
  public:
    Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return _value;
    }
    void print(char* prefix);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) {
    _value = seed;
  }

  int Foo::my_magic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = my_magic(_value, number);
    _value += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << _value << "\n";
  }
}
```

```
$ cd ~/myprog
$ g++ -Wall -Wextra -pedantic -Werror foo.cpp main.cpp && ./a.out
the answer=43
```

2. always use tools to support the building process

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
    int _value;
    int my_magic(int a, int b);
  public:
    Foo(int seed);
    int calc(int numb
    int getValue() {
      return _value;
    }
    void print(char*
  };
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) {
```

```
                                    b) {
```

```
                                    ue, number);
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "th
  for(int i=0; i<4; i
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

```
                                    ) {
                                    < "\n";
```

~/myprog/Makefile

```
CPPFLAGS=-Wall -Wextra -pedantic -Werror
LDFLAGS=

all: myprog

myprog: foo.o main.o
    $(CXX) -o $@ $(LDFLAGS) $+

clean:
    rm -f foo.o main.o myprog
```

```
$ cd ~/myprog
$ make
$ ./myprog
the answer=43
```

3. do not _prefix member variables, use postfix_ if you must

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
    int _value;
    int my_magic(int a, int b);
  public:
    Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return _value;
    }
    void print(char* prefix);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) {
    _value = seed;
  }

  int Foo::my_magic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = my_magic(_value, number);
    _value += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << _value << "\n";
  }
}
```

3. do not _prefix member variables, use postfix_ if you must

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
    int value_;
    int my_magic(int a, int b);
  public:
    Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return value_;
    }
    void print(char* prefix);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) {
    value_ = seed;
  }

  int Foo::my_magic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = my_magic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << value_ << "\n";
  }
}
```

4. public stuff should be declared first, then the private stuff

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
    int value_;
    int my_magic(int a, int b);
  public:
    Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return value_;
    }
    void print(char* prefix);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) {
    value_ = seed;
  }

  int Foo::my_magic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = my_magic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << value_ << "\n";
  }
}
```

4. public stuff should be declared first, then the private stuff

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return value_;
    }
    void print(char* prefix);
  private:
    int value_;
    int my_magic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) {
    value_ = seed;
  }

  int Foo::my_magic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = my_magic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << value_ << "\n";
  }
}
```

5. single argument constructors should usually be explicit

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return value_;
    }
    void print(char* prefix);
  private:
    int value_;
    int my_magic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) {
    value_ = seed;
  }

  int Foo::my_magic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = my_magic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << value_ << "\n";
  }
}
```

5. single argument constructors should usually be explicit

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return value_;
    }
    void print(char* prefix);
  private:
    int value_;
    int my_magic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) {
    value_ = seed;
  }

  int Foo::my_magic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = my_magic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << value_ << "\n";
  }
}
```

6. initialize the state of the object properly

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return value_;
    }
    void print(char* prefix);
  private:
    int value_;
    int my_magic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) {
    value_ = seed;
  }

  int Foo::my_magic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = my_magic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << value_ << "\n";
  }
}
```

6. initialize the state of the object properly

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return value_;
    }
    void print(char* prefix);
  private:
    int value_;
    int my_magic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::my_magic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = my_magic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << value_ << "\n";
  }
}
```

7. use a consistent naming convention, camelCase or under_score

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return value_;
    }
    void print(char* prefix);
  private:
    int value_;
    int my_magic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::my_magic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = my_magic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << value_ << "\n";
  }
}
```

7. use a consistent naming convention, camelCase or under_score

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return value_;
    }
    void print(char* prefix);
  private:
    int value_;
    int myMagic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << value_ << "\n";
  }
}
```

8. do not prefix queries and modifiers with get/set

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int getValue() {
      return value_;
    }
    void print(char* prefix);
  private:
    int value_;
    int myMagic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << value_ << "\n";
  }
}
```

8. do not prefix queries and modifiers with get/set

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() {
      return value_;
    }
    void print(char* prefix);
  private:
    int value_;
    int myMagic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << value_ << "\n";
  }
}
```

9. do not import namespaces

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() {
      return value_;
    }
    void print(char* prefix);
  private:
    int value_;
    int myMagic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

using namespace std;

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) {
    cout << prefix << value_ << "\n";
  }
}
```

9. do not import namespaces

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() {
      return value_;
    }
    void print(char* prefix);
  private:
    int value_;
    int myMagic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) {
    std::cout << prefix << value_ << "\n";
  }
}
```

10. query functions should be declared const

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() {
      return value_;
    }
    void print(char* prefix);
  private:
    int value_;
    int myMagic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) {
    std::cout << prefix << value_ << "\n";
  }
}
```

10. query functions should be declared const

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() const {
      return value_;
    }
    void print(char* prefix) const;
  private:
    int value_;
    int myMagic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

11. non-const functions are modifiers

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() const {
      return value_;
    }
    void print(char* prefix) const;
  private:
    int value_;
    int myMagic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

?

12. prefer free-standing functions

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() const {
      return value_;
    }
    void print(char* prefix) const;
  private:
    int value_;
    int myMagic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

12. prefer free-standing functions

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() const {
      return value_;
    }
    void print(char* prefix) const;
  private:
    int value_;
    int myMagic(int a, int b);
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

12. prefer free-standing functions

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() const {
      return value_;
    }
    void print(char* prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

13. use anonymous namespaces for private free-standing functions

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() const {
      return value_;
    }
    void print(char* prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

13. use anonymous namespaces for private free-standing functions

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() const {
      return value_;
    }
    void print(char* prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

14. do not inline stuff in the class definition

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() const {
      return value_;
    }
    void print(char* prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

14. do not inline stuff in the class definition

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() const;
    void print(char* prefix) const;
  private:
    int value_;
  };

  inline int Foo::value() const {
    return value_;
  }
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

15. by default keep your stuff in the implementation file

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() const;
    void print(char* prefix) const;
  private:
    int value_;
  };

  inline int Foo::value() const {
    return value_;
  }
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

15. by default keep your stuff in the implementation file

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() const;
    void print(char* prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

16. avoid member functions that both modifies and queries

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    int calc(int number = 7);
    int value() const;
    void print(char* prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  int Foo::calc(int number) {
    int result = myMagic(value_, number);
    value_ += result;
    return result;
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

16. avoid member functions that both modifies and queries

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number = 7);
    int value() const;
    void print(char* prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

17. default arguments are depreciated, use delegation if you must

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number = 7);
    int value() const;
    void print(char* prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

17. default arguments are depreciated, use delegation if you must

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(char* prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

18. the K&R vs BS war is over, use an extra space around & and *

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(char* prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(char *prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

18. the K&R vs BS war is over, use an extra space around & and *

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(char * prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

19. by not specifying const you say that something will change

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(char * prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

19. by not specifying const you say that something will change

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  const char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(const char * prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

20. reduce scope of variables

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  const char * prefix = "the answer=";
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(const char * prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

20. reduce scope of variables

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  const char * prefix = "the answer=";
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(const char * prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

21. for-loops in C++ are often not written like this

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  for(int i=0; i<4; i++) {
    foo.calc(i);
  }
  const char * prefix = "the answer=";
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(const char * prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

21. for-loops in C++ are often not written like this

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i) {
    foo.calc(i);
  }
  const char * prefix = "the answer=";
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(const char * prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

22. in C++ you do not need to explicitly return from main

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i) {
    foo.calc(i);
  }
  const char * prefix = "the answer=";
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(const char * prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

22. in C++ you do not need to explicitly return from main

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i) {
    foo.calc(i);
  }
  const char * prefix = "the answer=";
  foo.print(prefix);
  return 0;
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(const char * prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

22. in C++ you do not need to explicitly return from main

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i) {
    foo.calc(i);
  }
  const char * prefix = "the answer=";
  foo.print(prefix);
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(const char * prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

23. inject side-effects if you must have them

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i) {
    foo.calc(i);
  }
  const char * prefix = "the answer=";
  foo.print(prefix);
}
```

~/myprog/foo.cpp

```
#include <iostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(const char * prefix) const {
    std::cout << prefix << value_ << "\n";
  }
}
```

23. inject side-effects if you must have them

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(std::ostream & out,
               const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include <iostream>
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i) {
    foo.calc(i);
  }
  const char * prefix = "the answer=";
  foo.print(std::cout, prefix);
}
```

~/myprog/foo.cpp

```
#include <ostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(std::ostream & out,
                  const char * prefix) const {
    out << prefix << value_ << "\n";
  }
}
```

24. make sure headers compile by itself

~/myprog/foo.hpp

```
namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(std::ostream & out,
               const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include <iostream>
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i) {
    foo.calc(i);
  }
  const char * prefix = "the answer=";
  foo.print(std::cout, prefix);
}
```

~/myprog/foo.cpp

```
#include <ostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(std::ostream & out,
                  const char * prefix) const {
    out << prefix << value_ << "\n";
  }
}
```

24. make sure headers compile by itself

~/myprog/foo.hpp

```
#include <iostream>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(std::ostream & out,
               const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include <iostream>
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i) {
    foo.calc(i);
  }
  const char * prefix = "the answer=";
  foo.print(std::cout, prefix);
}
```

~/myprog/foo.cpp

```
#include <ostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(std::ostream & out,
                  const char * prefix) const {
    out << prefix << value_ << "\n";
  }
}
```

25. include your own header file first, standard libraries last

~/myprog/foo.hpp

```
#include <iostream>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(std::ostream & out,
               const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include <iostream>
#include "foo.hpp"

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i) {
    foo.calc(i);
  }
  const char * prefix = "the answer=";
  foo.print(std::cout, prefix);
}
```

~/myprog/foo.cpp

```
#include <ostream>
#include "foo.hpp"

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(std::ostream & out,
                  const char * prefix) const {
    out << prefix << value_ << "\n";
  }
}
```

25. include your own header file first, standard libraries last

~/myprog/foo.hpp

```
#include <iostream>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(std::ostream & out,
               const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i) {
    foo.calc(i);
  }
  const char * prefix = "the answer=";
  foo.print(std::cout, prefix);
}
```

~/myprog/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(std::ostream & out,
                  const char * prefix) const {
    out << prefix << value_ << "\n";
  }
}
```

26. prefer forward declarations in header files

~/myprog/foo.hpp

```
#include <iostream>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(std::ostream & out,
               const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i) {
    foo.calc(i);
  }
  const char * prefix = "the answer=";
  foo.print(std::cout, prefix);
}
```

~/myprog/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(std::ostream & out,
                  const char * prefix) const {
    out << prefix << value_ << "\n";
  }
}
```

26. prefer forward declarations in header files

~/myprog/foo.hpp

```
#include <iosfwd>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(std::ostream & out,
               const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i) {
    foo.calc(i);
  }
  const char * prefix = "the answer=";
  foo.print(std::cout, prefix);
}
```

~/myprog/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(std::ostream & out,
                  const char * prefix) const {
    out << prefix << value_ << "\n";
  }
}
```

27. don't need braces here

~/myprog/foo.hpp

```
#include <iosfwd>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(std::ostream & out,
               const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i) {
    foo.calc(i);
  }
  const char * prefix = "the answer=";
  foo.print(std::cout, prefix);
}
```

~/myprog/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(std::ostream & out,
                  const char * prefix) const {
    out << prefix << value_ << "\n";
  }
}
```

27. don't need braces here

~/myprog/foo.hpp

```
#include <iosfwd>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(std::ostream & out,
               const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i)
    foo.calc(i);
  const char * prefix = "the answer=";
  foo.print(std::cout, prefix);
}
```

~/myprog/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(std::ostream & out,
                  const char * prefix) const {
    out << prefix << value_ << "\n";
  }
}
```

28. avoid side-effects if you can, prefer free-standing functions

~/myprog/foo.hpp

```
#include <iosfwd>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
    void print(std::ostream & out,
               const char * prefix) const;
  private:
    int value_;
  };
}
```

~/myprog/main.cpp

```
#include "foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i)
    foo.calc(i);
  const char * prefix = "the answer=";
  foo.print(std::cout, prefix);
}
```

~/myprog/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void Foo::print(std::ostream & out,
                  const char * prefix) const {
    out << prefix << value_ << "\n";
  }
}
```

28. avoid side-effects if you can, prefer free-standing functions

~/myprog/foo.hpp

```
#include <iosfwd>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
  private:
    int value_;
  };
  void print(std::ostream & out,
             const char * prefix,
             const Foo & foo);
}
```

~/myprog/main.cpp

```
#include "foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i)
    foo.calc(i);
  const char * prefix = "the answer=";
  print(std::cout, prefix, foo);
}
```

~/myprog/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void print(std::ostream & out,
             const char * prefix,
             const Foo & foo) {
    out << prefix << foo.value() << "\n";
  }
}
```

29. do not open a namespace when implementing free-standing functions

~/myprog/foo.hpp

```
#include <iosfwd>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
  private:
    int value_;
  };
  void print(std::ostream & out,
             const char * prefix,
             const Foo & foo);
}
```

~/myprog/main.cpp

```
#include "foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i)
    foo.calc(i);
  const char * prefix = "the answer=";
  print(std::cout, prefix, foo);
}
```

~/myprog/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }

  void print(std::ostream & out,
             const char * prefix,
             const Foo & foo) {
    out << prefix << foo.value() << "\n";
  }
}
```

29. do not open a namespace when implementing free-standing functions

~/myprog/foo.hpp

```
#include <iosfwd>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
  private:
    int value_;
  };
  void print(std::ostream & out,
             const char * prefix,
             const Foo & foo);
}
```

~/myprog/main.cpp

```
#include "foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i)
    foo.calc(i);
  const char * prefix = "the answer=";
  print(std::cout, prefix, foo);
}
```

~/myprog/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }
}

void bar::print(std::ostream & out,
                const char * prefix,
                const bar::Foo & foo) {
  out << prefix << foo.value() << "\n";
}
```

30. operator overloading is sometimes a nice thing

~/myprog/foo.hpp

```
#include <iosfwd>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
  private:
    int value_;
  };
  void print(std::ostream & out,
             const char * prefix,
             const Foo & foo);
}
```

~/myprog/main.cpp

```
#include "foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i)
    foo.calc(i);
  const char * prefix = "the answer=";
  print(std::cout, prefix, foo);
}
```

~/myprog/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }
}

void bar::print(std::ostream & out,
                const char * prefix,
                const bar::Foo & foo) {
  out << prefix << foo.value() << "\n";
}
```

30. operator overloading is sometimes a nice thing

~/myprog/foo.hpp

```
#include <iosfwd>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
  private:
    int value_;
  };
  std::ostream & operator<<(
    std::ostream & out,
    const bar::Foo & foo);
}
```

~/myprog/main.cpp

```
#include "foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i)
    foo.calc(i);
  std::cout << "the answer="
            << foo << std::endl;
}
```

~/myprog/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }
}

std::ostream & bar::operator<<(
  std::ostream & out,
  const bar::Foo & foo) {
  return out << foo.value();
}
```

31. namespaces usually corresponds to directories, and vice versa

~/myprog/foo.hpp

```
#include <iosfwd>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
  private:
    int value_;
  };
  std::ostream & operator<<(
    std::ostream & out,
    const bar::Foo & foo);
}
```

~/myprog/main.cpp

```
#include "foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i)
    foo.calc(i);
  std::cout << "the answer="
            << foo << std::endl;
}
```

~/myprog/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }
}

std::ostream & bar::operator<<(
  std::ostream & out,
  const bar::Foo & foo) {
  return out << foo.value();
}
```

31. namespaces usually corresponds to directories, and vice versa

~/myprog/bar/foo.hpp

```
#include <iosfwd>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
  private:
    int value_;
  };
  std::ostream & operator<<(
    std::ostream & out,
    const bar::Foo & foo);
}
```

~/myprog/main.cpp

```
#include "bar/foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i)
    foo.calc(i);
  std::cout << "the answer="
            << foo << std::endl;
}
```

~/myprog/bar/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }
}

std::ostream & bar::operator<<(
  std::ostream & out,
  const bar::Foo & foo) {
  return out << foo.value();
}
```

32. use include guards in header files

~/myprog/bar/foo.hpp

```
#include <iosfwd>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
  private:
    int value_;
  };
  std::ostream & operator<<(
    std::ostream & out,
    const bar::Foo & foo);
}
```

~/myprog/bar/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }
}

std::ostream & bar::operator<<(
  std::ostream & out,
  const bar::Foo & foo) {
  return out << foo.value();
}
```

~/myprog/main.cpp

```
#include "bar/foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i)
    foo.calc(i);
  std::cout << "the answer="
            << foo << std::endl;
}
```

32. use include guards in header files

~/myprog/bar/foo.hpp

```
#ifndef BAR_FOO_HPP
#define BAR_FOO_HPP

#include <iosfwd>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
  private:
    int value_;
  };
  std::ostream & operator<<(
    std::ostream & out,
    const bar::Foo & foo);
}

#endif
```

~/myprog/main.cpp

```
#include "bar/foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i)
    foo.calc(i);
  std::cout << "the answer="
            << foo << std::endl;
}
```

~/myprog/bar/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }
}

std::ostream & bar::operator<<(
  std::ostream & out,
  const bar::Foo & foo) {
  return out << foo.value();
}
```

33. real professionals indent by four spaces

~/myprog/bar/foo.hpp

```
#ifndef BAR_FOO_HPP
#define BAR_FOO_HPP

#include <iosfwd>

namespace bar {
  class Foo {
  public:
    explicit Foo(int seed);
    void calc(int number);
    int value() const;
  private:
    int value_;
  };
  std::ostream & operator<<(
    std::ostream & out,
    const bar::Foo & foo);
}

#endif
```

~/myprog/main.cpp

```
#include "foo.hpp"
#include <iostream>

int main() {
  bar::Foo foo(2);
  for (int i = 0; i != 4; ++i)
    foo.calc(i);
  std::cout << "the answer="
            << foo << std::endl;
}
```

~/myprog/bar/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
  int myMagic(int a, int b) {
    return a + b - 3 + 8 - 5;
  }
}

namespace bar {
  Foo::Foo(int seed) : value_(seed) {
  }

  void Foo::calc(int number) {
    value_ += myMagic(value_, number);
  }

  int Foo::value() const {
    return value_;
  }
}

std::ostream & bar::operator<<(
  std::ostream & out,
  const bar::Foo & foo) {
  return out << foo.value();
}
```

33. real professionals indent by four spaces

~/myprog/bar/foo.hpp

```
#ifndef BAR_FOO_HPP
#define BAR_FOO_HPP

#include <iosfwd>

namespace bar {
    class Foo {
    public:
        explicit Foo(int seed);
        void calc(int number);
        int value() const;
    private:
        int value_;
    };
    std::ostream & operator<<(
        std::ostream & out,
        const bar::Foo & foo);
}

#endif
```

~/myprog/main.cpp

```
#include "bar/foo.hpp"
#include <iostream>

int main() {
    bar::Foo foo(2);
    for (int i = 0; i != 4; ++i)
        foo.calc(i);
    std::cout << "the answer="
              << foo << std::endl;
}
```

~/myprog/bar/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
    int myMagic(int a, int b) {
        return a + b - 3 + 8 - 5;
    }
}

namespace bar {
    Foo::Foo(int seed) : value_(seed) {
    }

    void Foo::calc(int number) {
        value_ += myMagic(value_, number);
    }

    int Foo::value() const {
        return value_;
    }
}

std::ostream & bar::operator<<(
    std::ostream & out,
    const bar::Foo & foo) {
    return out << foo.value();
}
```

~/myprog/bar/foo.hpp

```
#ifndef BAR_FOO_HPP
#define BAR_FOO_HPP

#include <iosfwd>

namespace bar {
    class Foo {
    public:
        explicit Foo(int seed);
        void calc(int number);
        int value() const;
    private:
        int value_;
    };
    std::ostream & operator<<(
        std::ostream & out,
        const bar::Foo & foo);
}

#endif
```

~/myprog/main.cpp

```
#include "bar/foo.hpp"
#include <iostream>

int main() {
    bar::Foo foo(2);
    for (int i = 0; i != 4; ++i)
        foo.calc(i);
    std::cout << "the answer="
              << foo << std::endl;
}
```

~/myprog/bar/foo.cpp

```
#include "foo.hpp"
#include <ostream>

namespace {
    int myMagic(int a, int b) {
        return a + b - 3 + 8 - 5;
    }
}

namespace bar {
    Foo::Foo(int seed) : value_(seed) {
    }

    void Foo::calc(int number) {
        value_ += myMagic(value_, number);
    }

    int Foo::value() const {
        return value_;
    }
}

std::ostream & bar::operator<<(
    std::ostream & out,
    const bar::Foo & foo) {
    return out << foo.value();
}
```

Anything else?

Now the looks like it is written by someone who cares... but the anwer is still incorrect. It should be 42, of course. Take this as a reminder about the importance of testing and validating your code properly.

0. Show that you care. (Or: Do sweat the small stuff)
1. always compile with -Wall and -Werror
2. always use tools to support the building process
3. do not _prefix member variables, use postfix_ if you must
4. public stuff should be declared first, then the private stuff
5. single argument constructors should usually be explicit
6. initialize the state of the object properly
7. use a consistent naming convention, camelCase or under_score
8. do not prefix queries and modifiers with get/set
9. do not import namespaces
10. query functions should be declared const
11. non-const functions are modifiers
12. prefer free-standing functions
13. use anonymous namespaces for private free-standing functions
14. do not inline stuff in the class definition
15. by default keep your stuff in the implementation file
16. avoid member functions that both modifies and queries
17. default arguments are depreciated, use delegation if you must
18. the K&R vs BS war is over, use an extra space around & and *
19. by not specifying const you say that something will change
20. reduce scope of variables
21. for-loops in C++ are often not written like this
22. in C++ you do not need to explicitly return from main
23. inject side-effects if you must have them
24. make sure headers compile by itself
25. include your own header file first, standard libraries last
26. prefer forward declarations in header files
27. don't need braces here
28. avoid side-effects if you can, prefer free-standing functions
29. do not open a namespace when implementing free-standing functions
30. operator overloading is sometimes a nice thing
31. namespaces usually corresponds to directories, and vice versa
32. use include guards in header files
33. real professionals indent by four spaces