Życie bez #ifdefów

Bartosz ’BaSz’ Szurgot

bartek.szurgot@baszerr.eu

2013-10-17

Plan

A teraz...

Prosty kod

```
1 auto answer = 42;
2 // ...
3 ofstream of("tmp/cache.tmp");
4 of << "number:";
5 of << answer;
6 of << endl;
```

Prosty kod

```
1 auto answer = 42;
2 // ...
3 ofstream of("tmp/cache.tmp");
4 of << "number:";
5 of << answer;
6 of << endl;
```

wspierający Windowsa...

```
1  auto answer = 42;
2  // ...
3  #ifdef WIN32
4  ofstream of("tmp\\cache.tmp",
5              ios_base::binary);
6  #else
7  ofstream of("tmp/cache.tmp");
8  #endif
9  of << "number:";
10 of << answer;
11 of << endl;
```

wspierający Windowsa...

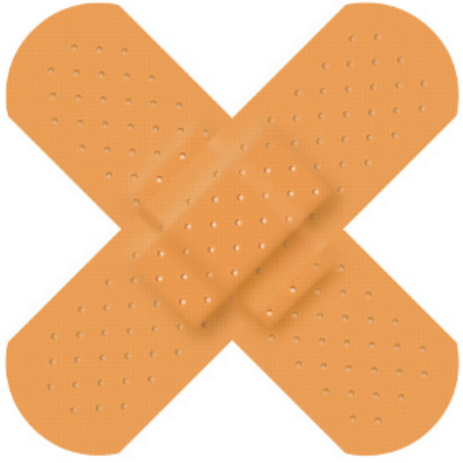
```
1  auto answer = 42;
2  // ...
3  #ifdef WIN32
4  ofstream of("tmp\\cache.tmp",
5              ios_base::binary);
6  #else
7  ofstream of("tmp/cache.tmp");
8  #endif
9  of << "number:";
10 of << answer;
11 of << endl;
```

nowego klienta...

```
1  auto answer = 42;
2  // ...
3  #ifdef WIN32
4  #ifdef DUZY_KLIENT
5  ofstream of("katalog_klienta\\cache.tmp", ios_base::binary);
6  #else
7  ofstream of("tmp\\cache.tmp", ios_base::binary);
8  #endif
9  #else
10 #ifdef DUZY_KLIENT
11 ofstream of("katalog_klienta/cache.tmp");
12 #else
13 ofstream of("tmp/cache.tmp");
14 #endif
15 #endif
16 of << "number:";
17 of << answer;
18 of << endl;
```

nowego klienta...

```
1  auto answer = 42;
2  // ...
3  #ifdef WIN32
4  #ifdef DUZY_KLIENT
5  ofstream of("katalog_klienta\\cache.tmp", ios_base::binary);
6  #else
7  ofstream of("tmp\\cache.tmp", ios_base::binary);
8  #endif
9  #else
10 #ifdef DUZY_KLIENT
11 ofstream of("katalog_klienta/cache.tmp");
12 #else
13 ofstream of("tmp/cache.tmp");
14 #endif
15 #endif
16 of << "number:";
17 of << answer;
18 of << endl;
```

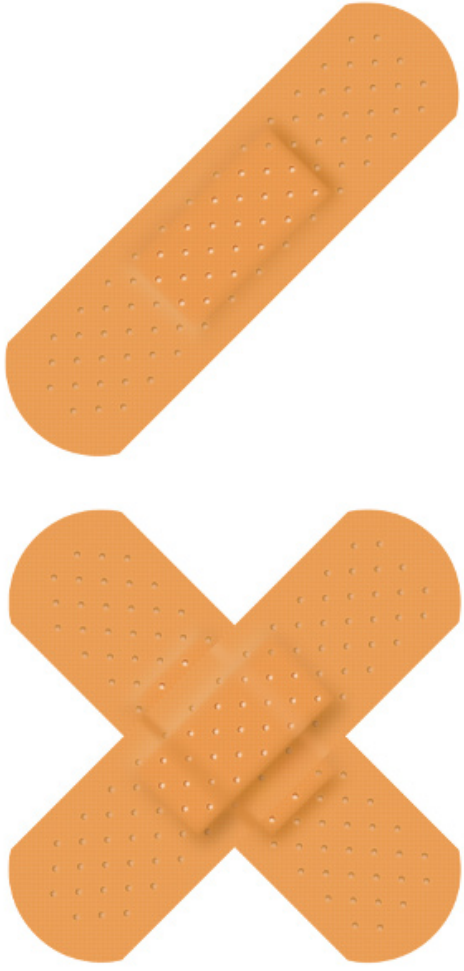


drugiego klienta...

```
1   auto answer = 42;
2   // ...
3   #ifdef WIN32
4   #ifdef DUZY_KLIENT
5   ofstream of("katalog_klienta\\cache.tmp", ios_base::binary);
6   #else
7   #ifdef INNY_KLIENT
8   ofstream of("katalog_innego_klienta\\cache.tmp", ios_base::binary);
9   #else
10  ofstream of("tmp\\cache.tmp", ios_base::binary);
11  #endif
12  #endif
13  #else
14  #ifdef DUZY_KLIENT
15  ofstream of("katalog_klienta/cache.tmp");
16  #else
17  #ifdef INNY_KLIENT
18  ofstream of("katalog_innego_klienta/cache.tmp");
19  #endif
20  ofstream of("tmp/cache.tmp");
21  #endif
22  #endif
23  of << "number:";
24  of << answer;
25  of << endl;
```

drugiego klienta...

```
1   auto answer = 42;
2   // ...
3   #ifdef WIN32
4   #ifdef DUZY_KLIENT
5   ofstream of("katalog_klienta\\cache.tmp", ios_base::binary);
6   #else
7   #ifdef INNY_KLIENT
8   ofstream of("katalog_innego_klienta\\cache.tmp", ios_base::binary);
9   #else
10  ofstream of("tmp\\cache.tmp", ios_base::binary);
11  #endif
12  #endif
13  #else
14  #ifdef DUZY_KLIENT
15  ofstream of("katalog_klienta/cache.tmp");
16  #else
17  #ifdef INNY_KLIENT
18  ofstream of("katalog_innego_klienta/cache.tmp");
19  #endif
20  ofstream of("tmp/cache.tmp");
21  #endif
22  #endif
23  of << "number:";
24  of << answer;
25  of << endl;
```



funkcjonalność++...

```
1 std::string data;
2 ifstream ifs("tmp/cache.tmp");
3 ifs >> data;
4 cout << data << endl;
```

funkcjonalność++...

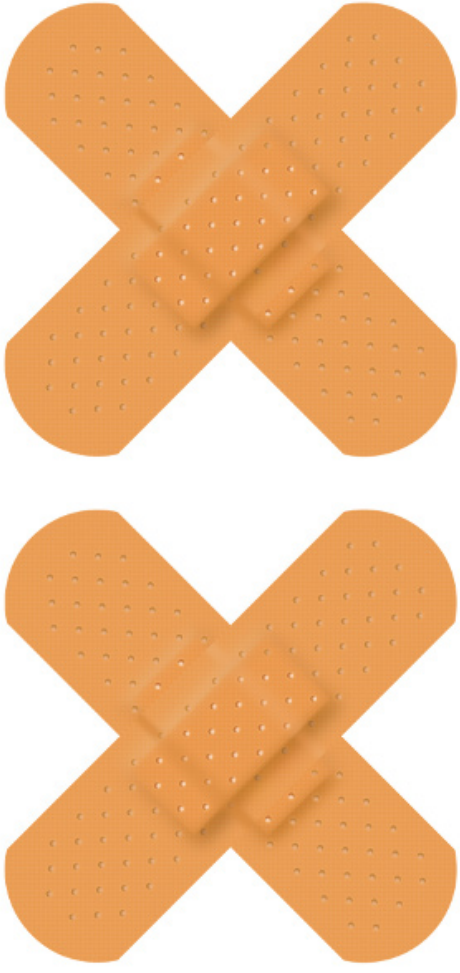
```
1 std::string data;
2 ifstream ifs("tmp/cache.tmp");
3 ifs >> data;
4 cout << data << endl;
```

wspierająca wszystko...

```
1   std::string data;
2   #ifdef WIN32
3   #ifdef DUZY_KLIENT
4   ifstream ifs("katalog_klienta\\cache.tmp", ios_base::binary);
5   #else
6   #ifdef INNY_KLIENT
7   ifstream ifs("katalog_innego_klienta\\cache.tmp", ios_base::binary);
8   #else
9   ifstream ifs("tmp\\cache.tmp", ios_base::binary);
10  #endif
11  #endif
12  #else
13  #ifdef DUZY_KLIENT
14  ifstream ifs("katalog_klienta/cache.tmp");
15  #else
16  #ifdef INNY_KLIENT
17  ifstream ifs("katalog_innego_klienta/cache.tmp");
18  #endif
19  ifstream ifs("tmp/cache.tmp");
20  #endif
21  #endif
22  ifs >> data;
23  cout << data << endl;
```

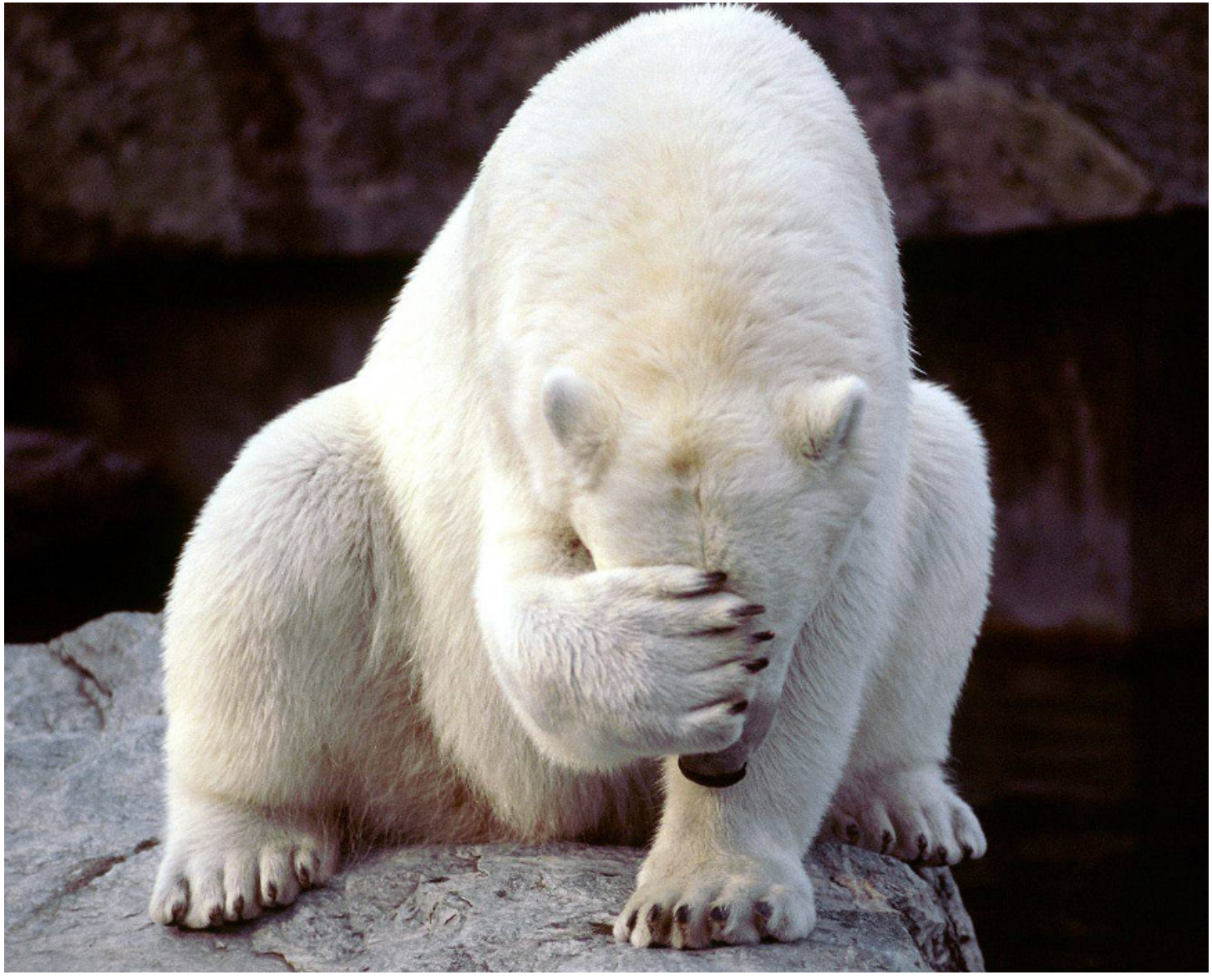
wspierająca wszystko...

```
1   std::string data;
2   #ifdef WIN32
3   #ifdef DUZY_KLIENT
4   ifstream ifs("katalog_klienta\\cache.tmp", ios_base::binary);
5   #else
6   #ifdef INNY_KLIENT
7   ifstream ifs("katalog_innego_klienta\\cache.tmp", ios_base::binary);
8   #else
9   ifstream ifs("tmp\\cache.tmp", ios_base::binary);
10  #endif
11  #endif
12  #else
13  #ifdef DUZY_KLIENT
14  ifstream ifs("katalog_klienta/cache.tmp");
15  #else
16  #ifdef INNY_KLIENT
17  ifstream ifs("katalog_innego_klienta/cache.tmp");
18  #endif
19  ifstream ifs("tmp/cache.tmp");
20  #endif
21  #endif
22  ifs >> data;
23  cout << data << endl;
```

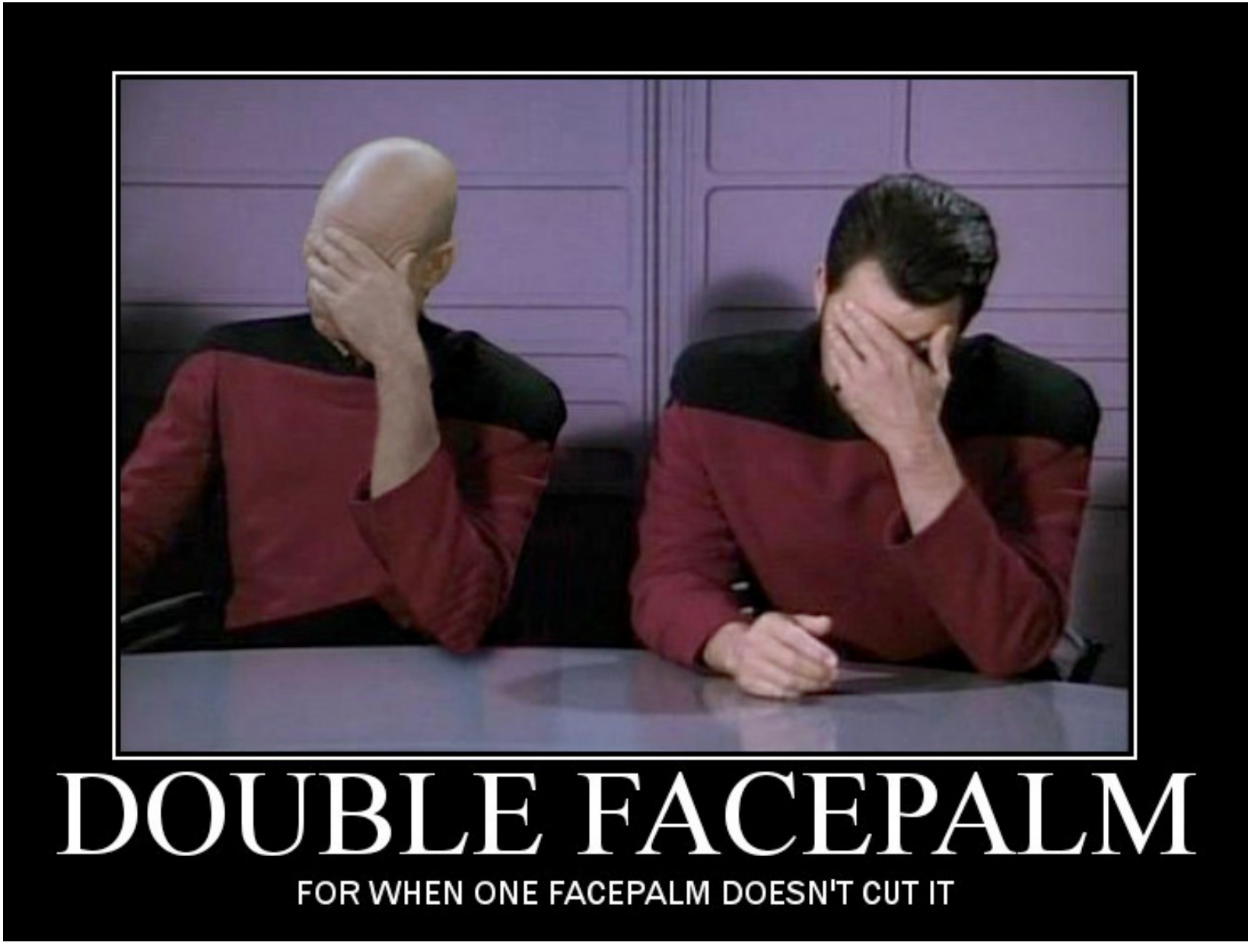


Code review!

...i wstyd



...i to poważny



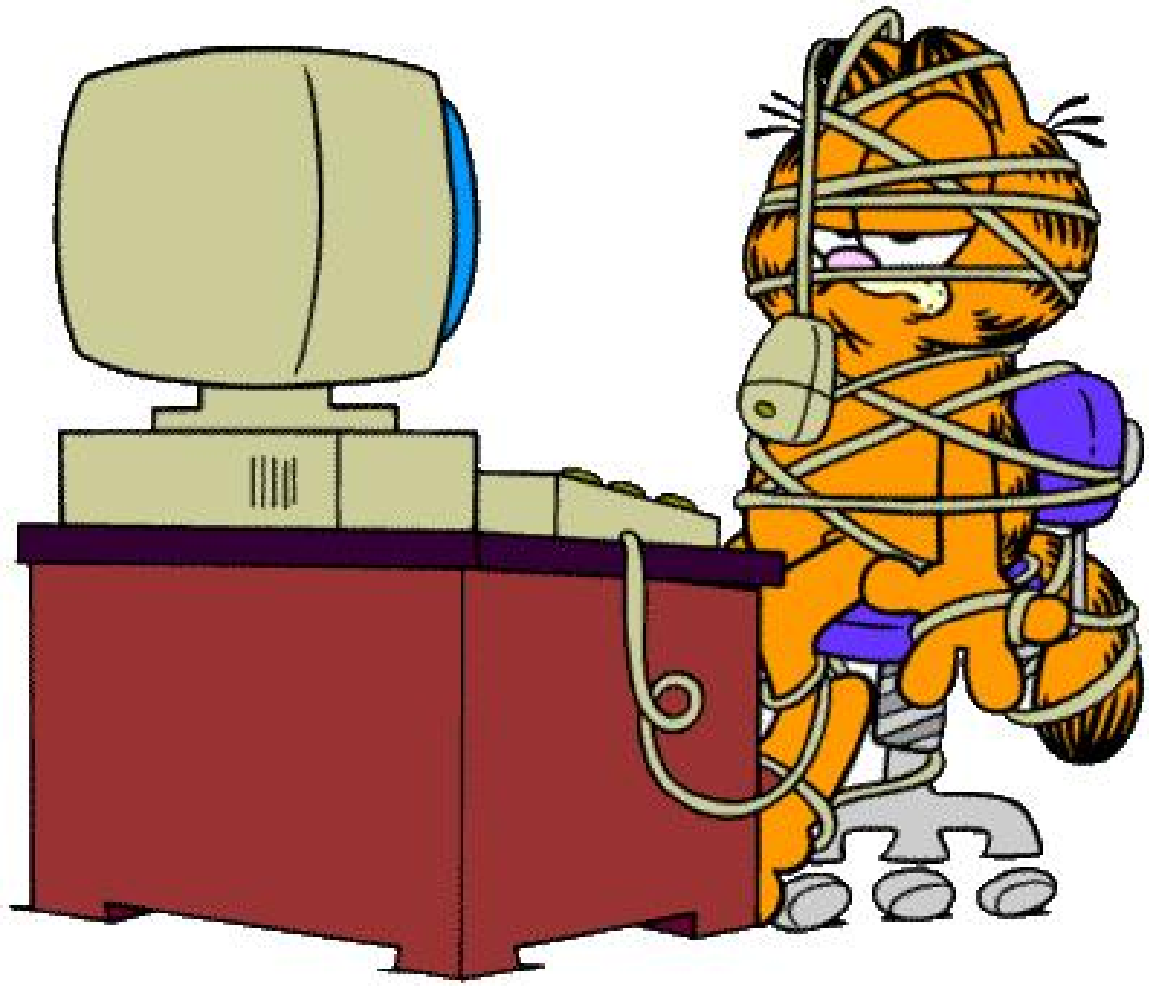
A teraz...

Szybka poprawka

Tylko dla tego klienta

Przecież działa!

Poprawimy później

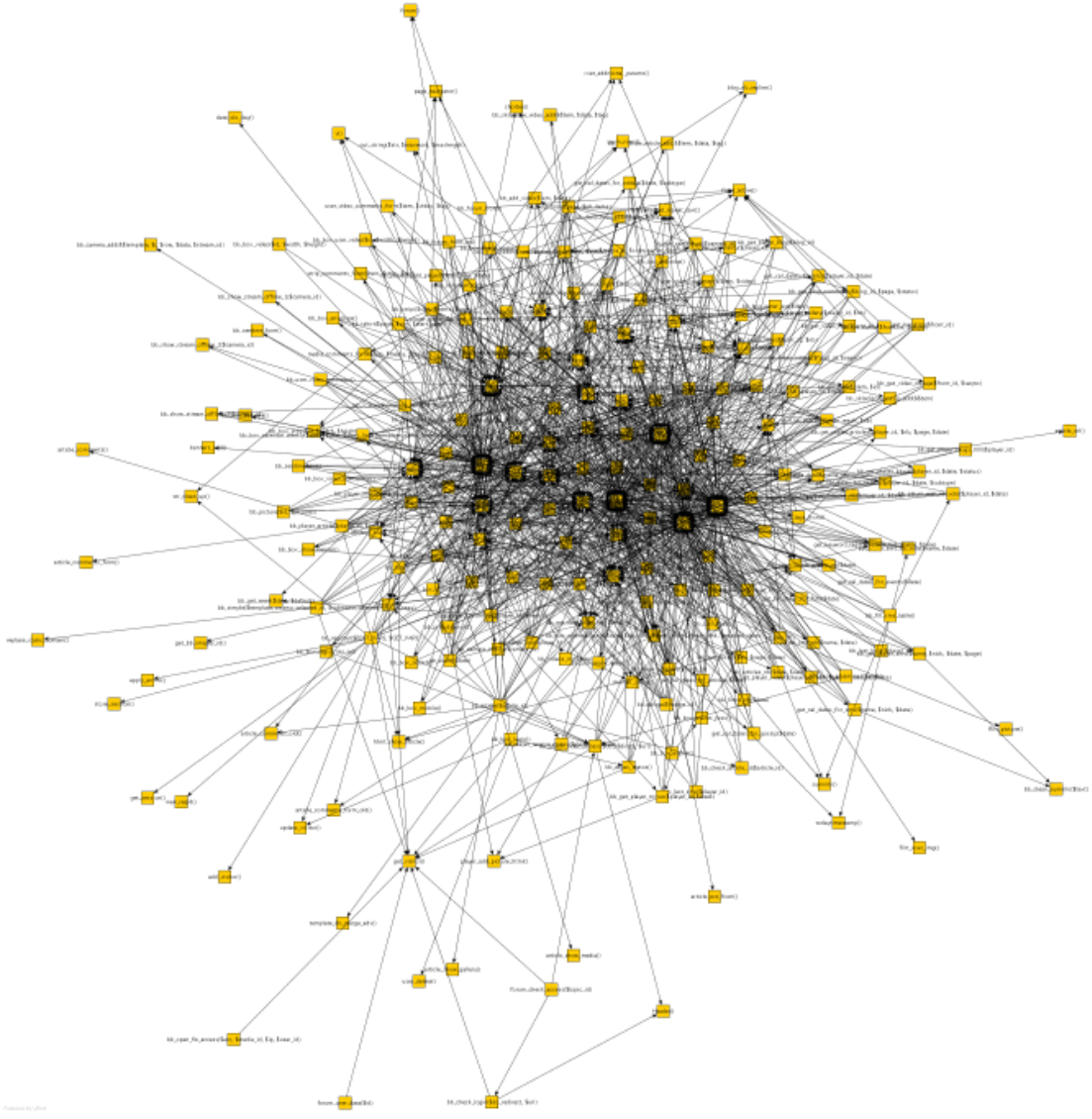


A teraz...

Puzel

```
1  #ifndef GKO_NAMESPACE
2  #if defined(GKO_UL_SWITCH)
3  #define GKO_NAMESPACE GKO_UL
4  #elif !(defined(GKO_MODULETEST_2)||defined(GKO_DUMMY_SWITCH))
5  #define GKO_NAMESPACE GKO_DL
6  #else
7  #define NO_GKO_NAMESPACE
8  #endif
9  #endif
10 #ifdef NO_GKO_NAMESPACE
11 #define GKO_NAMESPACE_BEGIN
12 #define GKO_NAMESPACE_END
13 #else
14 #define GKO_NAMESPACE_BEGIN namespace GKO_NAMESPACE {
15 #define GKO_NAMESPACE_END } using namespace GKO_NAMESPACE;
16 #endif
```

Mija kwartał...



Że jak?

```
1  // logger.hpp:
2  #define LOG_INFO(msg) \
3              do { \
4                cout << "INFO: " << msg << endl;\
5              } while(0)
6
7  // MyClass.cpp:
8  int a = 4;
9  a++;
10 LOG_INFO("a == " << a);
```

Że jak?

```
1  // logger.hpp:
2  #define LOG_INFO(msg) \
3              do { \
4                cout << "INFO: " << msg << endl;\
5              } while(0)
6
7  // MyClass.cpp:
8  int a = 4;
9  a++;
10 LOG_INFO("a == " << a);
```

- GCC 4.8: *invalid operands of types ‘const char [6]’ and ‘int’ to binary ‘operator«’*

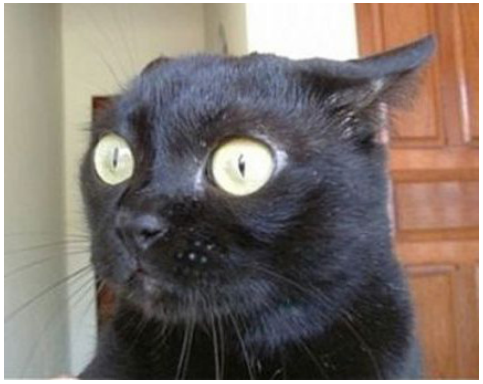
Że jak?

```
1  // logger.hpp:
2  #define LOG_INFO(msg) \
3              do { \
4                cout << "INFO: " << msg << endl;\
5              } while(0)
6
7  // MyClass.cpp:
8  int a = 4;
9  a++;
10 LOG_INFO("a == " << a);
```

- GCC 4.8: *invalid operands of types ‘const char [6]’ and ‘int’ to binary ‘operator«’*
- *GCC 3.3: ‘6’ is not a function*

Że jak?

```
1  // logger.hpp:
2  #define LOG_INFO(msg) \
3              do { \
4                cout << "INFO: " << msg << endl;\
5              } while(0)
6
7  // MyClass.cpp:
8  int a = 4;
9  a++;
10 LOG_INFO("a == " << a);
```



- GCC 4.8: *invalid operands of types ‘const char [6]’ and ‘int’ to binary ‘operator«’*
- *GCC 3.3: ‘6’ is not a function*

Znajdź 2 różnice

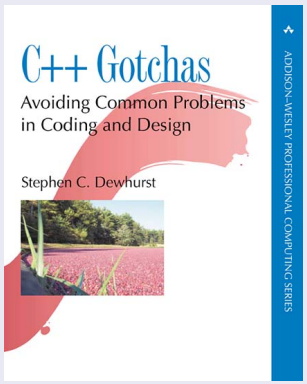
```
1   #define S1 { /* ...*/ }
2   struct Prot { /* ... */ };
3   struct SomeCodec_DecodeErrorInfo
4   {
5     // ...
6     SomeCodec_DecodeErrorInfo(Prot prot, unsigned int noOfBytes = 0)
7     {
8       byteCount = noOfBytes;
9       errorCount = 0;
10      protocol = prot;
11    };
12  #ifdef SOME_UNIT_TEST
13    SomeCodec_DecodeErrorInfo(signed int noOfBytes = 0)
14    {
15      byteCount = noOfBytes;
16      errorCount = 0;
17      protocol = S1;
18    };
19  #endif
20    // ...
21    int byteCount;
22    int errorCount;
23    Prot protocol;
24  };
```

Złożoność

Stephen Dewhurst, „C++ Gotchas”

„I have to use #if to handle the different platform requirements”. To prove your point, such as it is, you display the following code:

```
// some portable code...
#ifdef PLATFORM_A
// do something...
a(); b(); c();
#endif
#ifdef PLATFORM_B
// do same thing...
d(); e();
#endif
```



This code is not platform-independent. It’s multiplatform dependent. Any change to any of the platforms requires not only a recompilation of the source but change to the source for all platforms. You’ve achieved maximal coupling among platforms: a remarkable achievement, if somewhat impractical.

O(MG)

```
1  #ifndef PRODUKCJA
2  ofstream file("stuff.log");
3  #endif
4  ostream& os =
5  #ifdef PRODUKCJA
6  cerr;
7  #else
8  #ifdef TESTY
9  file;
10 #else
11 cout;
12 #endif
13 #endif
14 #ifdef PLATFORMA_A
15 os << "platforma-A" << endl;
16 #endif
17 #ifdef PLATFORMA_B
18 os << "Platforma-B" << endl;
19 #endif
20 #ifdef PLATFORMA_C
21 os << "PLATFORMA-C" << endl;
22 #else
23 os << "nie-C" << endl;
24 #endif
```

O(MG)

```
1  #ifndef PRODUKCJA
2  ofstream file("stuff.log");
3  #endif
4  ostream& os =
5  #ifdef PRODUKCJA
6  cerr;
7  #else
8  #ifdef TESTY
9  file;
10 #else
11 cout;
12 #endif
13 #endif
14 #ifdef PLATFORMA_A
15 os << "platforma-A" << endl;
16 #endif
17 #ifdef PLATFORMA_B
18 os << "Platforma-B" << endl;
19 #endif
20 #ifdef PLATFORMA_C
21 os << "PLATFORMA-C" << endl;
22 #else
23 os << "nie-C" << endl;
24 #endif
```

- Możliwości:

O(MG)

```
1  #ifndef PRODUKCJA
2  ofstream file("stuff.log");
3  #endif
4  ostream& os =
5  #ifdef PRODUKCJA
6  cerr;
7  #else
8  #ifdef TESTY
9  file;
10 #else
11 cout;
12 #endif
13 #endif
14 #ifdef PLATFORMA_A
15 os << "platforma-A" << endl;
16 #endif
17 #ifdef PLATFORMA_B
18 os << "Platforma-B" << endl;
19 #endif
20 #ifdef PLATFORMA_C
21 os << "PLATFORMA-C" << endl;
22 #else
23 os << "nie-C" << endl;
24 #endif
```

- Możliwości:
  - Przeznaczenie ($N = 2$):
    1. PRODUKCJA
    2. TESTY

O(MG)

```
1  #ifndef PRODUKCJA
2  ofstream file("stuff.log");
3  #endif
4  ostream& os =
5  #ifdef PRODUKCJA
6  cerr;
7  #else
8  #ifdef TESTY
9  file;
10 #else
11 cout;
12 #endif
13 #endif
14 #ifdef PLATFORMA_A
15 os << "platforma-A" << endl;
16 #endif
17 #ifdef PLATFORMA_B
18 os << "Platforma-B" << endl;
19 #endif
20 #ifdef PLATFORMA_C
21 os << "PLATFORMA-C" << endl;
22 #else
23 os << "nie-C" << endl;
24 #endif
```

- Możliwości:
  - Przeznaczenie ($N = 2$):
    1. PRODUKCJA
    2. TESTY
  - Platformy ($M = 3$):
    1. PLATFORMA_A
    2. PLATFORMA_B
    3. PLATFORMA_C

O(MG)

```
1   #ifndef PRODUKCJA
2   ofstream file("stuff.log");
3   #endif
4   ostream& os =
5   #ifdef PRODUKCJA
6   cerr;
7   #else
8   #ifdef TESTY
9   file;
10  #else
11  cout;
12  #endif
13  #endif
14  #ifdef PLATFORMA_A
15  os << "platforma-A" << endl;
16  #endif
17  #ifdef PLATFORMA_B
18  os << "Platforma-B" << endl;
19  #endif
20  #ifdef PLATFORMA_C
21  os << "PLATFORMA-C" << endl;
22  #else
23  os << "nie-C" << endl;
24  #endif
```

- Możliwości:
  - Przeznaczenie ($N = 2$):
    1. PRODUKCJA
    2. TESTY
  - Platformy ($M = 3$):
    1. PLATFORMA_A
    2. PLATFORMA_B
    3. PLATFORMA_C
- $O(2^{N+M})$

O(MG)

```
1  #ifndef PRODUKCJA
2  ofstream file("stuff.log");
3  #endif
4  ostream& os =
5  #ifdef PRODUKCJA
6  cerr;
7  #else
8  #ifdef TESTY
9  file;
10 #else
11 cout;
12 #endif
13 #endif
14 #ifdef PLATFORMA_A
15 os << "platforma-A" << endl;
16 #endif
17 #ifdef PLATFORMA_B
18 os << "Platforma-B" << endl;
19 #endif
20 #ifdef PLATFORMA_C
21 os << "PLATFORMA-C" << endl;
22 #else
23 os << "nie-C" << endl;
24 #endif
```

- Możliwości:
  - Przeznaczenie ($N = 2$):
    1. PRODUKCJA
    2. TESTY
  - Platformy ($M = 3$):
    1. PLATFORMA_A
    2. PLATFORMA_B
    3. PLATFORMA_C
- $O(2^{N+M})$
- aka: $O(scary)$

O(MG)

```
1  #ifndef PRODUKCJA
2  ofstream file("stuff.log");
3  #endif
4  ostream& os =
5  #ifdef PRODUKCJA
6  cerr;
7  #else
8  #ifdef TESTY
9  file;
10 #else
11 cout;
12 #endif
13 #endif
14 #ifdef PLATFORMA_A
15 os << "platforma-A" << endl;
16 #endif
17 #ifdef PLATFORMA_B
18 os << "Platforma-B" << endl;
19 #endif
20 #ifdef PLATFORMA_C
21 os << "PLATFORMA-C" << endl;
22 #else
23 os << "nie-C" << endl;
24 #endif
```

- Możliwości:
  - Przeznaczenie ($N = 2$):
    1. PRODUKCJA
    2. TESTY
  - Platformy ($M = 3$):
    1. PLATFORMA_A
    2. PLATFORMA_B
    3. PLATFORMA_C
- $O(2^{N+M})$
- aka: $O(scary)$
- 32 możliwe kompilacje!

A teraz...

Koło. Okrągłe.

- Zły pomysł:

```
1  auto answer = 42;
2  // ...
3  #ifdef WIN32
4  ofstream of("tmp\\cache.tmp",
5              ios_base::binary);
6  #else
7  ofstream of("tmp/cache.tmp");
8  #endif
9  of << "number:";
10 of << answer;
11 of << endl;
```

Koło. Okrągłe.

- Zły pomysł:

```
1  auto answer = 42;
2  // ...
3  #ifdef WIN32
4  ofstream of("tmp\\cache.tmp",
5              ios_base::binary);
6  #else
7  ofstream of("tmp/cache.tmp");
8  #endif
9  of << "number:";
10 of << answer;
11 of << endl;
```

Koło. Okrągłe.

- Zły pomysł:

```
1   auto answer = 42;
2   // ...
3   #ifdef WIN32
4   ofstream of("tmp\\cache.tmp",
5               ios_base::binary);
6   #else
7   ofstream of("tmp/cache.tmp");
8   #endif
9   of << "number:";
10  of << answer;
11  of << endl;
```

- Dobry pomysł:

```
1   auto answer = 42;
2   // ...
3   namespace  fs = boost::filesystem;
4   const auto p  = fs::path("tmp") / "cache.tmp";
5   ofstream of( p.string().c_str(), ios_base::binary);
6   of << "number:";
7   of << answer;
8   of << endl;
```

Przypadek prosty

Wymagane API:

```
1  #pragma once
2  struct Legacy
3  {
4    static void save32(void const*) { /*...*/ }
5    static void save64(void const*) { /*...*/ }
6  };
```

Przypadek prosty

Wymagane API:

```
1  #pragma once
2  struct Legacy
3  {
4    static void save32(void const*) { /*...*/ }
5    static void save64(void const*) { /*...*/ }
6  };
```

Zły pomysł:

```
1  #include "Legacy.hpp"
2  // ...
3  void myCode(void)
4  {
5    auto platformDependentSize = 10l;
6  #ifdef PLATFORM_A
7    Legacy::save64(&platformDependentSize);
8  #else
9    Legacy::save32(&platformDependentSize);
10 #endif
11 }
```

Przypadek prosty

Wymagane API:

```
1  #pragma once
2  struct Legacy
3  {
4    static void save32(void const*) { /*...*/ }
5    static void save64(void const*) { /*...*/ }
6  };
```

Zły pomysł:

```
1  #include "Legacy.hpp"
2  // ...
3  void myCode(void)
4  {
5    auto platformDependentSize = 10l;
6  #ifdef PLATFORM_A
7    Legacy::save64(&platformDependentSize);
8  #else
9    Legacy::save32(&platformDependentSize);
10 #endif
11 }
```

Dobry pomysł:

```
1  #include "Legacy.hpp"
2
3  namespace detail
4  {
5  template<unsigned N>
6  void saveImpl(void const*);
7  template<>
8  void saveImpl<4>(void const* t)
9  { Legacy::save32(t); }
10 template<>
11 void saveImpl<8>(void const* t)
12 { Legacy::save64(t); }
13 } // namespace detail
14
15 // API
16 template<typename T>
17 void save(T const& t)
18 { detail::saveImpl<sizeof(T)>(&t); }
19
20 // ...
21 void func(void)
22 {
23   auto platformDependentSize = 10l;
24   save(platformDependentSize);
25 }
```

Ach ten komplikator

Zły pomysł:

```
1  #ifdef PLATFORMA_A
2  // kod budujacy sie tylko na A:
3  fa();
4  #endif
5  #ifdef PLATFORMA_B
6  // kod budujacy sie tylko na B:
7  fb();
8  #endif
```

Ach ten komplikator

Zły pomysł:

```
1  #ifdef PLATFORMA_A
2  // kod budujacy sie tylko na A:
3  fa();
4  #endif
5  #ifdef PLATFORMA_B
6  // kod budujacy sie tylko na B:
7  fb();
8  #endif
```

- prosty...

Ach ten komplikator

Zły pomysł:

```
1  #ifdef PLATFORMA_A
2  // kod budujacy sie tylko na A:
3  fa();
4  #endif
5  #ifdef PLATFORMA_B
6  // kod budujacy sie tylko na B:
7  fb();
8  #endif
```

- prosty...
- zwięzły...

Ach ten komplikator

Zły pomysł:

```
1  #ifdef PLATFORMA_A
2  // kod budujacy sie tylko na A:
3  fa();
4  #endif
5  #ifdef PLATFORMA_B
6  // kod budujacy sie tylko na B:
7  fb();
8  #endif
```

- prosty...
- zwięzły...
- kompletnie nie skalowalny...

Ach ten komplikator

Zły pomysł:

```
1  #ifdef PLATFORMA_A
2  // kod budujacy sie tylko na A:
3  fa();
4  #endif
5  #ifdef PLATFORMA_B
6  // kod budujacy sie tylko na B:
7  fb();
8  #endif
```

- prosty...
- zwięzły...
- kompletnie nie skalowalny...

Dobry pomysł:

```
1  // PlatformAbstraction.hpp:
2  void f(void);
```

```
1  // PlatformA.cpp:
2  void fa(void);
3  void f(void)
4  {
5    // kod budujacy sie tylko na A:
6    fa();
7  }
```

```
1  // PlatformB.cpp:
2  void fb(void);
3  void f(void)
4  {
5    // kod budujacy sie tylko na B:
6    fb();
7  }
```

Ach ten komplikator

Zły pomysł:

```
1  #ifdef PLATFORMA_A
2  // kod budujacy sie tylko na A:
3  fa();
4  #endif
5  #ifdef PLATFORMA_B
6  // kod budujacy sie tylko na B:
7  fb();
8  #endif
```

- prosty...
- zwięzły...
- kompletnie nie skalowalny...

Dobry pomysł:

```
1  // PlatformAbstraction.hpp:
2  void f(void);
```

```
1  // PlatformA.cpp:
2  void fa(void);
3  void f(void)
4  {
5    // kod budujacy sie tylko na A:
6    fa();
7  }
```

```
1  // PlatformB.cpp:
2  void fb(void);
3  void f(void)
4  {
5    // kod budujacy sie tylko na B:
6    fb();
7  }
```

- build system switch
- aka: „split backed”

Co z wydajnością?

Zastane API:

```
1  #pragma once
2  struct MagicNumber
3  { /* ... */ };
```

Co z wydajnością?

Zastane API:

```
1  #pragma once
2  struct MagicNumber
3  { /* ... */ };
```

Zły pomysł:

```
1  #include "MagicNumber.hpp"
2  void myCode(void)
3  {
4    MagicNumber a;
5    (void)a;
6  #ifdef PLATFORMA_A
7    // kod budujacy sie tylko na A:
8    plusplus(a);
9  #endif
10 #ifdef PLATFORMA_B
11   // kod budujacy sie tylko na B:
12   inc(a);
13 #endif
14 }
```

Co z wydajnością?

Zastane API:

```
1  #pragma once
2  struct MagicNumber
3  { /* ... */ };
```

Zły pomysł:

```
1  #include "MagicNumber.hpp"
2  void myCode(void)
3  {
4    MagicNumber a;
5    (void)a;
6  #ifdef PLATFORMA_A
7    // kod budujacy sie tylko na A:
8    plusplus(a);
9  #endif
10 #ifdef PLATFORMA_B
11   // kod budujacy sie tylko na B:
12   inc(a);
13 #endif
14 }
```

Dobry pomysł:

```
1  // PlatformA/Abstraction.hpp:
2  #include "MagicNumber.hpp"
3  void plusplus(MagicNumber&);
4  inline void increment(MagicNumber& a)
5  {
6    plusplus(a);
7  }
```

```
1  // PlatformB/Abstraction.hpp:
2  #include "MagicNumber.hpp"
3  void inc(MagicNumber&);
4  inline void increment(MagicNumber& a)
5  {
6    inc(a);
7  }
```

Co z wydajnością?

Zastane API:

```
1  #pragma once
2  struct MagicNumber
3  { /* ... */ };
```

Zły pomysł:

```
1  #include "MagicNumber.hpp"
2  void myCode(void)
3  {
4    MagicNumber a;
5    (void)a;
6  #ifdef PLATFORMA_A
7    // kod budujacy sie tylko na A:
8    plusplus(a);
9  #endif
10 #ifdef PLATFORMA_B
11   // kod budujacy sie tylko na B:
12   inc(a);
13 #endif
14 }
```

Dobry pomysł:

```
1  // PlatformA/Abstraction.hpp:
2  #include "MagicNumber.hpp"
3  void plusplus(MagicNumber&);
4  inline void increment(MagicNumber& a)
5  {
6    plusplus(a);
7  }
```

```
1  // PlatformB/Abstraction.hpp:
2  #include "MagicNumber.hpp"
3  void inc(MagicNumber&);
4  inline void increment(MagicNumber& a)
5  {
6    inc(a);
7  }
```

- gcc -I$(PLATFORM)
- aka: „split header”

Większe zmiany

- Co z grubszymi zmianami?

Większe zmiany

- Co z grubszymi zmianami?
  - Inne protokoły?
  - Inne formaty?

Większe zmiany

- Co z grubszymi zmianami?
  - Inne protokoły?
  - Inne formaty?
- Pluginy!
  - Fabryka (abstrakcyjna)
  - Build tworzy *.so per format/protokół
  - Tworzenie fabryki rejestruje *.so

A teraz...

Abstrakcja

- Wymusza podział funkcjonalny
- Wydziela odpowiedzialności
- Poprawia czytelność

Abstrakcja

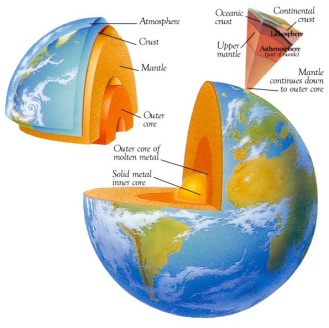
- Wymusza podział funkcjonalny
- Wydziela odpowiedzialności
- Poprawia czytelność

```
1   #include "MagicNumber.hpp"
2   #include "Abstraction.hpp"  // <- split backend/header
3
4   void doStuff(MagicNumber& m);
5
6   void myCode(MagicNumber& m)
7   {
8     increment(m); // <- platform independent!
9     doStuff(m);   // <- common code
10  }
```

Abstrakcja

- Wymusza podział funkcjonalny
- Wydziela odpowiedzialności
- Poprawia czytelność

```
1   #include "MagicNumber.hpp"
2   #include "Abstraction.hpp"  // <- split backend/header
3
4   void doStuff(MagicNumber& m);
5
6   void myCode(MagicNumber& m)
7   {
8     increment(m); // <- platform independent!
9     doStuff(m);   // <- common code
10  }
```



- Samo-tworząca się warstwa abstrakcji!

Podział

- Wyraźne oddzielenie fragmentów:
  - per-klient
  - per-platforma

Podział

- Wyraźne oddzielenie fragmentów:
  - per-klient
  - per-platforma
- Łatwa modyfikacja:
  - krótkie, osobne pliki
  - przenośność (np. nowa platforma)
  - ponowne użycie
  - lokalizacja zmian (sys/linux, sys/windows, sys/mac, etc. . .)
  - mało parametrów do kompilatora

Podział

- Wyraźne oddzielenie fragmentów:
  - per-klient
  - per-platforma
- Łatwa modyfikacja:
  - krótkie, osobne pliki
  - przenośność (np. nowa platforma)
  - ponowne użycie
  - lokalizacja zmian (sys/linux, sys/windows, sys/mac, etc. . .)
  - mało parametrów do kompilatora
- Platformy nie mieszają kodu z klientami

Podział

- Wyraźne oddzielenie fragmentów:
  - per-klient
  - per-platforma
- Łatwa modyfikacja:
  - krótkie, osobne pliki
  - przenośność (np. nowa platforma)
  - ponowne użycie
  - lokalizacja zmian (sys/linux, sys/windows, sys/mac, etc. . .)
  - mało parametrów do kompilatora
- Platformy nie mieszają kodu z klientami
- Wspólne testy do większości

Podział

- Wyraźne oddzielenie fragmentów:
  - per-klient
  - per-platforma
- Łatwa modyfikacja:
  - krótkie, osobne pliki
  - przenośność (np. nowa platforma)
  - ponowne użycie
  - lokalizacja zmian (sys/linux, sys/windows, sys/mac, etc. . .)
  - mało parametrów do kompilatora
- Platformy nie mieszają kodu z klientami
- Wspólne testy do większości
- Separacja kodu powstaje naturalnie. . .

Czytelność

ClientSettings.hpp:

```
1  #pragma once
2  #include <boost/filesystem.hpp>
3  boost::filesystem::path clientCacheDir(void);
```

Czytelność

ClientSettings.hpp:

```
1  #pragma once
2  #include <boost/filesystem.hpp>
3  boost::filesystem::path clientCacheDir(void);
```

```
1  #include "ClientSettings.hpp"
2  void write(void)
3  {
4    const auto answer = 42;
5    const auto f = clientCacheDir() / "cache.tmp";
6    ofstream   of( f.string().c_str(), ios_base::binary | ios_base::out );
7    of << "number:";
8    of << answer;
9    of << endl;
10 }
```

Czytelność

ClientSettings.hpp:

```
1  #pragma once
2  #include <boost/filesystem.hpp>
3  boost::filesystem::path clientCacheDir(void);
```

```
1  #include "ClientSettings.hpp"
2  void write(void)
3  {
4    const auto answer = 42;
5    const auto f = clientCacheDir() / "cache.tmp";
6    ofstream   of( f.string().c_str(), ios_base::binary | ios_base::out );
7    of << "number:";
8    of << answer;
9    of << endl;
10 }
```

```
1  #include "ClientSettings.hpp"
2  void read(void)
3  {
4    const auto  f = clientCacheDir() / "cache.tmp";
5    ifstream    ifs( f.string().c_str(), ios_base::binary | ios_base::in );
6    std::string data;
7    ifs >> data;
8    cout << data << endl;
9  }
```

O(nice)

```
1   #include "ClientSettings.hpp"
2   void write(void)
3   {
4     const auto answer = 42;
5     const auto f = clientCacheDir() / "cache.tmp";
6     ofstream   of( f.string().c_str(), ios_base::binary | ios_base::out );
7     of << "number:";
8     of << answer;
9     of << endl;
10  }
```

O(nice)

```
1   #include "ClientSettings.hpp"
2   void write(void)
3   {
4     const auto answer = 42;
5     const auto f = clientCacheDir() / "cache.tmp";
6     ofstream   of( f.string().c_str(), ios_base::binary | ios_base::out );
7     of << "number:";
8     of << answer;
9     of << endl;
10  }
```

- *N* platform (2: linux + windows)
- *M* klientów (3)

O(nice)

```
1   #include "ClientSettings.hpp"
2   void write(void)
3   {
4     const auto answer = 42;
5     const auto f = clientCacheDir() / "cache.tmp";
6     ofstream   of( f.string().c_str(), ios_base::binary | ios_base::out );
7     of << "number:";
8     of << answer;
9     of << endl;
10  }
```

- N platform (2: linux + windows)
- M klientów (3)
- budujemy: 1 z N oraz 1 z M.

O(nice)

```
1   #include "ClientSettings.hpp"
2   void write(void)
3   {
4     const auto answer = 42;
5     const auto f = clientCacheDir() / "cache.tmp";
6     ofstream   of( f.string().c_str(), ios_base::binary | ios_base::out );
7     of << "number:";
8     of << answer;
9     of << endl;
10  }
```

- N platform (2: linux + windows)
- M klientów (3)
- budujemy: 1 z N oraz 1 z M.
- $O(N * M)$
- poprzednio: $O(2^{N*M})$...

O(nice)

```
1   #include "ClientSettings.hpp"
2   void write(void)
3   {
4     const auto answer = 42;
5     const auto f = clientCacheDir() / "cache.tmp";
6     ofstream   of( f.string().c_str(), ios_base::binary | ios_base::out );
7     of << "number:";
8     of << answer;
9     of << endl;
10  }
```

- N platform (2: linux + windows)
- M klientów (3)
- budujemy: 1 z N oraz 1 z M.
- $O(N * M)$
- poprzednio: $O(2^{N*M})$...
- build na straży porządku! :-)

Nie będziesz używał #ifdefów!

A teraz...

Ale ja naprawdę potrzebuję #ifdefa

Chyba że...

Header guard

```
1  #ifndef INCLUDE_FOO_BAR_MYCLASS_HPP_FILE
2  #define INCLUDE_FOO_BAR_MYCLASS_HPP_FILE
3
4
5  namespace Foo
6  {
7  namespace Bar
8  {
9  class MyClass
10 {
11   // ...
12 };
13 }
14 }
15 #endif
```

Header guard

```
1  #ifndef INCLUDE_FOO_BAR_MYCLASS_HPP_FILE
2  #define INCLUDE_FOO_BAR_MYCLASS_HPP_FILE
3                //  ~^~  ~^~  ~~~^~~~
4                //   |    |       |
5  namespace Foo//--+    |       |
6  {             //       |       |
7  namespace Bar//-------+       |
8  {             //               |
9  class MyClass//---------------+
10 {
11   // ...
12 };
13 }
14 }
15 #endif
```

Header guard

```
1  #ifndef INCLUDE_FOO_BAR_MYCLASS_HPP_FILE
2  #define INCLUDE_FOO_BAR_MYCLASS_HPP_FILE
3                 // ~^~ ~^~ ~~~^~~~
4                 //  |   |      |
5  namespace Foo//--+   |      |
6  {                    //      |      |
7  namespace Bar//------+      |
8  {                    //             |
9  class MyClass//-------------+
10 {
11   // ...
12 };
13 }
14 }
15 #endif
```

Logowanie

```
1  #define MY_LOGGER_INFO(msg) \
2              cout << "INFO␣"; \
3              cout << __FILE__ << ":" << __LINE__; \
4              cout << "␣" << msg << endl
5
6  void myCode(string const& name)
7  {
8    MY_LOGGER_INFO("hello␣" << name);
9  }
```

Logowanie

```
1  #define MY_LOGGER_INFO(msg) \
2            do { \
3              cout << "INFO␣"; \
4              cout << __FILE__ << ":" << __LINE__; \
5              cout << "␣" << msg << endl; \
6            } while(false)
7
8  void myCode(string const& name)
9  {
10   MY_LOGGER_INFO("hello␣" << name);
11 }
```

Logowanie

```
1  #define MY_LOGGER_INFO(msg) \
2            do { \
3              try { \
4              cout << "INFO␣"; \
5              cout << __FILE__ << ":" << __LINE__; \
6              cout << "␣" << msg << endl; \
7              } catch(...) { /* sorry... */ } \
8            } while(false)
9
10 void myCode(string const& name)
11 {
12   MY_LOGGER_INFO("hello␣" << name);
13 }
```

Logowanie

```
1  #ifdef NDEBUG // O RLY?
2  #define MY_LOGGER_DEBUG(msg) do { } while(false}
3  #else
4  #define MY_LOGGER_DEBUG(msg) \
5            do { \
6              try { \
7              cout << "DEBUG"; \
8              cout << __FILE__ << ":" << __LINE__; \
9              cout << "␣" << msg << endl; \
10             } catch(...) { /* sorry... */ } \
11           } while(false)
12 #endif
13
14 void myCode(string const& name)
15 {
16   MY_LOGGER_DEBUG("hello␣" << name);
17 }
```

Standardowe

- Nazwa pliku (__FILE__)
- Numer linii (__LINE__)

Standardowe

- Nazwa pliku (__FILE__)
- Numer linii (__LINE__)
- Data kompilacji (__DATE__)
- Czas kompilacji (__TIME__)

Standardowe

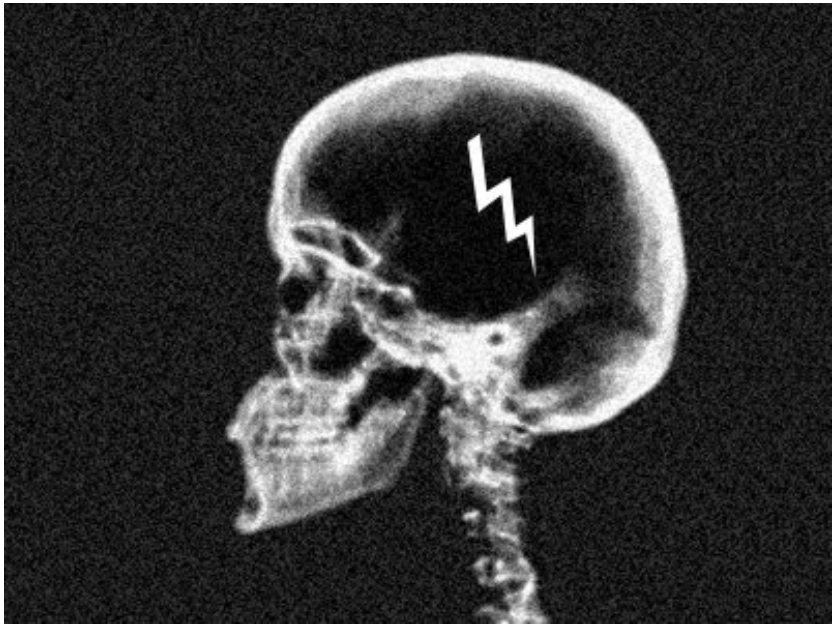
- Nazwa pliku (__FILE__)
- Numer linii (__LINE__)
- Data kompilacji (__DATE__)
- Czas kompilacji (__TIME__)
- assert()

Standardowe

- Nazwa pliku (__FILE__)
- Numer linii (__LINE__)
- Data kompilacji (__DATE__)
- Czas kompilacji (__TIME__)
- assert()
- NDEBUG – bardzo sporadycznie i z duża rozwagą!

A teraz...

Mem::store()



Modrzew

!#ifdef

Happy end