Hello Houston!

czyli rzecz o błędów zgłaszaniu

Bartosz ’BaSz’ Szurgot

bartek.szurgot@baszerr.eu

2013-11-07

Plan

A teraz...

Na początku był przykład...

- foo.hpp:

```
1  #pragma once
2  #include <string>
3  void foo(std::string const& str);
```

- foo.cpp:

```
1  #include <cstdio>
2  #include "foo.hpp"
3
4  void foo(std::string const& str)
5  {
6    printf( "Hello␣%s!\n", str.c_str() );
7  }
```

- main.cpp:

```
1  #include "foo.hpp"
2
3  int main(void)
4  {
5    foo("Dr.Evil");
6  }
```

Na początku był przykład...

- foo.hpp:

```
1  #pragma once
2  #include <string>
3  void foo(std::string const& str);
```

- foo.cpp:

```
1  #include <cstdio>
2  #include "foo.hpp"
3
4  void foo(std::string const& str)
5  {
6    printf( "Hello␣%s!\n", str.c_str() );
7  }
```

- main.cpp:

```
1  #include "foo.hpp"
2
3  int main(void)
4  {
5    foo("Dr.Evil");
6  }
```

- ...-fno-exceptions
- main.out => 4976[B]

Na początku był przykład. . .

- foo.hpp:

```
1  #pragma once
2  #include <string>
3  void foo(std::string const& str);
```

- foo.cpp:

```
1  #include <cstdio>
2  #include "foo.hpp"
3
4  void foo(std::string const& str)
5  {
6    printf( "Hello␣%s!\n", str.c_str() );
7  }
```

- main.cpp:

```
1  #include "foo.hpp"
2
3  int main(void)
4  {
5    foo("Dr.Evil");
6  }
```

- . . . -fno-exceptions
- main.out $=>$ 4976$[B]$
- z wyjątkami. . .
- main.out $=>$ 5416$[B]$

Na początku był przykład...

- foo.hpp:

```
1  #pragma once
2  #include <string>
3  void foo(std::string const& str);
```

- foo.cpp:

```
1  #include <cstdio>
2  #include "foo.hpp"
3
4  void foo(std::string const& str)
5  {
6    printf( "Hello␣%s!\n", str.c_str() );
7  }
```

- main.cpp:

```
1  #include "foo.hpp"
2
3  int main(void)
4  {
5    foo("Dr.Evil");
6  }
```

- ...-fno-exceptions
- main.out => 4976[B]
- z wyjątkami...
- main.out => 5416[B]
- 440[B] (~9%)!

Na początku był przykład...

- foo.hpp:

```
1  #pragma once
2  #include <string>
3  void foo(std::string const& str);
```

- foo.cpp:

```
1  #include <cstdio>
2  #include "foo.hpp"
3
4  void foo(std::string const& str)
5  {
6    printf( "Hello␣%s!\n", str.c_str() );
7  }
```

- main.cpp:

```
1  #include "foo.hpp"
2
3  int main(void)
4  {
5    foo("Dr.Evil");
6  }
```

- ...-fno-exceptions
- main.out => 4976[B]
- z wyjątkami...
- main.out => 5416[B]
- 440[B] (~9%)!

Albo inaczej...

- foo.hpp:

```
1  #pragma once
2  #include <string>
3  void foo(std::string const& str);
```

- foo.cpp:

```
1  #include <iostream>
2  #include "foo.hpp"
3
4  void foo(std::string const& str)
5  {
6    // NOTE: cout instead of printf()!
7    std::cout << "Hello␣" << str << "\n";
8  }
```

- main.cpp:

```
1  #include "foo.hpp"
2
3  int main(void)
4  {
5    foo("Dr.Evil");
6  }
```

Albo inaczej...

- foo.hpp:

```
1  #pragma once
2  #include <string>
3  void foo(std::string const& str);
```

- foo.cpp:

```
1  #include <iostream>
2  #include "foo.hpp"
3
4  void foo(std::string const& str)
5  {
6    // NOTE: cout instead of printf()!
7    std::cout << "Hello␣" << str << "\n";
8  }
```

- main.cpp:

```
1  #include "foo.hpp"
2
3  int main(void)
4  {
5    foo("Dr.Evil");
6  }
```

- ...-fno-exceptions
- main.out => 5776[*B*]

Albo inaczej...

- foo.hpp:

```
1  #pragma once
2  #include <string>
3  void foo(std::string const& str);
```

- foo.cpp:

```
1  #include <iostream>
2  #include "foo.hpp"
3
4  void foo(std::string const& str)
5  {
6    // NOTE: cout instead of printf()!
7    std::cout << "Hello␣" << str << "\n";
8  }
```

- main.cpp:

```
1  #include "foo.hpp"
2
3  int main(void)
4  {
5    foo("Dr.Evil");
6  }
```

- ...-fno-exceptions
- main.out => 5776[B]
- z wyjątkami...
- main.out => 6960[B]

Albo inaczej...

- foo.hpp:

```
1  #pragma once
2  #include <string>
3  void foo(std::string const& str);
```

- foo.cpp:

```
1  #include <iostream>
2  #include "foo.hpp"
3
4  void foo(std::string const& str)
5  {
6    // NOTE: cout instead of printf()!
7    std::cout << "Hello␣" << str << "\n";
8  }
```

- main.cpp:

```
1  #include "foo.hpp"
2
3  int main(void)
4  {
5    foo("Dr.Evil");
6  }
```

- ...-fno-exceptions
- main.out => 5776[*B*]
- z wyjątkami...
- main.out => 6960[*B*]
- 1184[*B*] (~20%)!!

Albo inaczej...

- foo.hpp:

```
1  #pragma once
2  #include <string>
3  void foo(std::string const& str);
```

- foo.cpp:

```
1  #include <iostream>
2  #include "foo.hpp"
3
4  void foo(std::string const& str)
5  {
6    // NOTE: cout instead of printf()!
7    std::cout << "Hello␣" << str << "\n";
8  }
```

- main.cpp:

```
1  #include "foo.hpp"
2
3  int main(void)
4  {
5    foo("Dr.Evil");
6  }
```

- ...-fno-exceptions
- main.out => 5776[*B*]
- z wyjątkami...
- main.out => 6960[*B*]
- 1184[*B*] (~20%)!!

Tylko że...

- Co co błędami z *foo()*?

Tylko że...

- Co co błędami z *foo()*?
- *std::bad_alloc*? :-/

Tylko że...

- Co co błędami z *foo()*?
- *std::bad_alloc*? :-/
- Co z błędami *iosreams*? 8-/

Tylko że...

- Co co błędami z *foo()*?
- *std::bad_alloc*? :-/
- Co z błędami *iosreams*? 8-/
- Ignorowanie błędów – delikatna sprawa...

Tylko że...

- Co co błędami z *foo()*?
- *std::bad_alloc*? :-/
- Co z błędami *iosreams*? 8-/
- Ignorowanie błędów – delikatna sprawa...

- Wyjątki vs. Brak-Sprawdzania-Błędów?!...

Złap mnie (jeśli potrafisz)

- Wyjątki:

```
1  #include "foo.hpp"
2
3  int main(void)
4  {
5    try
6    {
7      foo("Dr.Evil");
8    }
9    catch(...)
10   {
11     printf("Oops...\n");
12     return 1;
13   }
14 }
```

Złap mnie (jeśli potrafisz)

- Wyjątki:

```
1  #include "foo.hpp"
2
3  int main(void)
4  {
5    try
6    {
7      foo("Dr.Evil");
8    }
9    catch(...)
10   {
11     printf("Oops...\n");
12     return 1;
13   }
14 }
```

- Brak wyjątków:

```
1  #include "foo.hpp"
2
3  int main(void)
4  {
5    // try?
6    foo("Dr.Evil");
   // ?!
7    // catch()?
8    // idea-oops...
9  }
```

We can handle – or can we?

Ile możliwych ścieżek?

```
1  String f(Employee e)
2  {
3    if( e.Title() == "CEO" || e.Salary() > 100000 )
4    {
5      cout << e.First() << "␣" << e.Last()
6           << "␣is␣overpaid" << endl;
7    }
8    return e.First() + "␣" + e.Last();
9  }
```

Ile możliwych ścieżek?

```
1  String f(Employee e)
2  {
3    if( e.Title() == "CEO" || e.Salary() > 100000 )
4    {
5      cout << e.First() << "␣" << e.Last()
6           << "␣is␣overpaid" << endl;
7    }
8    return e.First() + "␣" + e.Last();
9  }
```

Prawidłowa odpowiedź to **23** (słownie: dwadzieścia trzy)!

A teraz...

Co robić? Jak żyć?

1. Kody powrotu
2. Globalne zmienne
3. Funkcje obsługi błędów
4. Argument wyjściowy

Co robić? Jak żyć?

1. Kody powrotu
2. Globalne zmienne
3. Funkcje obsługi błędów
4. Argument wyjściowy
5. Wyjątki. . . :-)

Co robić? Jak żyć?

1. Kody powrotu
2. Globalne zmienne
3. Funkcje obsługi błędów
4. Argument wyjściowy
5. Wyjątki. . . :-)

Kody powrotu

- Przykład:

```
1 int recLen(int a, int b)
2 {
3   if(a<=0 || b<=0)
4     return -1;
5   return 2*a+2*b;
6 }
```

Kody powrotu

- Przykład:

```
1 int recLen(int a, int b)
2 {
3   if(a<=0 || b<=0)
4     return -1;
5   return 2*a+2*b;
6 }
```

+ Znane
+ Proste
+ Szybkie
+ Przewidywalne

Kody powrotu

- Przykład:

```
1 int recLen(int a, int b)
2 {
3   if(a<=0 || b<=0)
4     return -1;
5   return 2*a+2*b;
6 }
```

+ Znane
+ Proste
+ Szybkie
+ Przewidywalne

– Za proste
– Jaki dokładnie błąd?
– Konwencje...
– Magiczne wartości
– Specjalna wartość == błąd
– Trzeba pamiętać o sprawdzeniu
– Zawsze sprawdzane

Kody powrotu – WAT?!

```
1  int myDiv(int a, int b)
2  {
3    if(b==0)
4      return -1;  // wat?!
5    return a/b;
6  }
```

Kody powrotu – WAT?!

```
1  int myDiv(int a, int b)
2  {
3    if(b==0)
4      return -1;  // wat?!
5    return a/b;
6  }
```

Globalne zmienne

- Przykład:

```
1   extern int errorNo;
2
3   int myDiv(int a, int b)
4   {
5     if(b==0)
6     {
7       errorNo = 42;   // magic!
8       return -1;      // again?!
9     }
10    return a/b;
11  }
12
13  bool use(void)
14  {
15    errorNo = 0;
16    auto r = myDiv(10,2);
17    if(errorNo)
18    {
19      // ...
20      return false;   // int? bool?
21    }
22    assert(r==5);
23    return true;
24  }
```

Globalne zmienne

- Przykład:

```
1   extern int errorNo;
2
3   int myDiv(int a, int b)
4   {
5     if(b==0)
6     {
7       errorNo = 42;   // magic!
8       return -1;      // again?!
9     }
10    return a/b;
11  }
12
13  bool use(void)
14  {
15    errorNo = 0;
16    auto r = myDiv(10,2);
17    if(errorNo)
18    {
19      // ...
20      return false;   // int? bool?
21    }
22    assert(r==5);
23    return true;
24  }
```

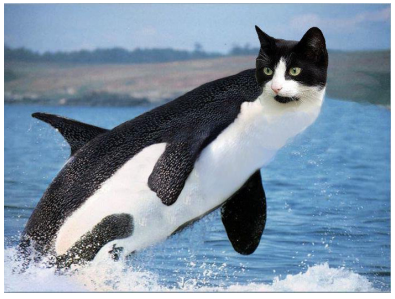
+ Proste
+ Znane (*errno*)

Globalne zmienne

- Przykład:

```
1   extern int errorNo;
2
3   int myDiv(int a, int b)
4   {
5     if(b==0)
6     {
7       errorNo = 42;    // magic!
8       return -1;       // again?!
9     }
10    return a/b;
11  }
12
13  bool use(void)
14  {
15    errorNo = 0;
16    auto r = myDiv(10,2);
17    if(errorNo)
18    {
19      // ...
20      return false;    // int? bool?
21    }
22    assert(r==5);
23    return true;
24  }
```

+ Proste
+ Znane (*errno*)

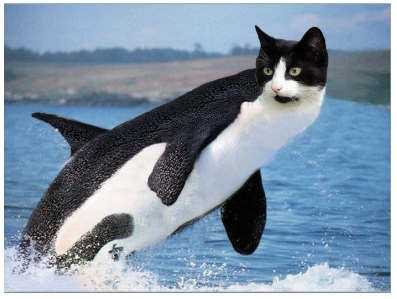


Globalne zmienne

- Przykład:

```
1   extern int errorNo;
2
3   int myDiv(int a, int b)
4   {
5     if(b==0)
6     {
7       errorNo = 42;    // magic!
8       return -1;       // again?!
9     }
10    return a/b;
11  }
12
13  bool use(void)
14  {
15    errorNo = 0;
16    auto r = myDiv(10,2);
17    if(errorNo)
18    {
19      // ...
20      return false;    // int? bool?
21    }
22    assert(r==5);
23    return true;
24  }
```

+ Proste
+ Znane (*errno*)



– Znów magia
– Zawsze sprawdzane
– Trzeba pamiętać o sprawdzeniu
– Wątki – oops. . .
– Sygnały – oops. . .

Funkcje obsługi błędów

- Przykład:

```
1  using errHandle = void(*)(char const* msg);
2
3  float myDiv(float a, float b, errHandle h)
4  {
5    if(b==0)
6    {
7      h("division by zero");
8      return -1;  // how about -42?
9    }
10   return a/b;
11 };
```

Funkcje obsługi błędów

- Przykład:

```
1  using errHandle = void(*)(char const* msg);
2
3  float myDiv(float a, float b, errHandle h)
4  {
5    if(b==0)
6    {
7      h("division by zero");
8      return -1;  // how about -42?
9    }
10   return a/b;
11 };
```

+ Akceptowalne w prostych przypadkach
+ Przyjmuje argumenty

Funkcje obsługi błędów

- Przykład:

```
1  using errHandle = void(*)(char const* msg);
2
3  float myDiv(float a, float b, errHandle h)
4  {
5    if(b==0)
6    {
7      h("division by zero");
8      return -1;  // how about -42?
9    }
10   return a/b;
11 };
```

+ Akceptowalne w prostych przypadkach
+ Przyjmuje argumenty

– „Rozgadane”
– Co jeśli nie *exit()*?
– 2 błędy, 1 funkcja?

Argument wyjściowy

- Przykład:

```
1   int myDiv(int a, int b, int& err)
2   {
3     if(b==0)
4     {
5       err = 42;   // magic!
6       return -1;  // again?!
7     }
8     return a/b;
9   }
10
11  bool use(void)
12  {
13    int err = 0;
14    auto r = myDiv(10,2, err);
15    if(err)
16    {
17      // ...
18      return false;   // int? bool?
19    }
20    assert(r==5);
21    return true;
22  }
```

Argument wyjściowy

- Przykład:

```
1   int myDiv(int a, int b, int& err)
2   {
3     if(b==0)
4     {
5       err = 42;   // magic!
6       return -1;  // again?!
7     }
8     return a/b;
9   }
10
11  bool use(void)
12  {
13    int err = 0;
14    auto r = myDiv(10,2, err);
15    if(err)
16    {
17      // ...
18      return false;   // int? bool?
19    }
20    assert(r==5);
21    return true;
22  }
```

+ Proste
+ Zlokalizowane

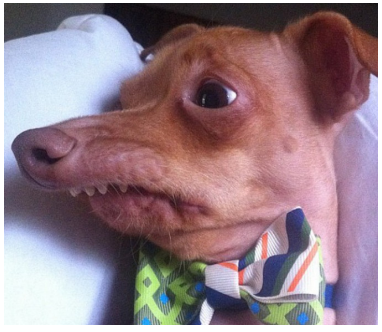
Argument wyjściowy

- Przykład:

```
1   int myDiv(int a, int b, int& err)
2   {
3     if(b==0)
4     {
5       err = 42;    // magic!
6       return -1;   // again?!
7     }
8     return a/b;
9   }
10
11  bool use(void)
12  {
13    int err = 0;
14    auto r = myDiv(10,2, err);
15    if(err)
16    {
17      // ...
18      return false;   // int? bool?
19    }
20    assert(r==5);
21    return true;
22  }
```

\+ Proste

\+ Zlokalizowane



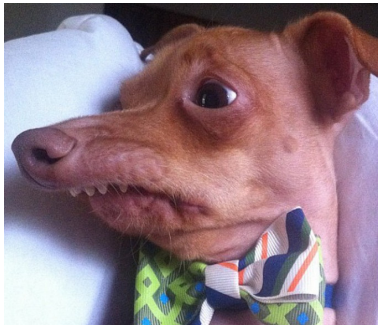
Argument wyjściowy

- Przykład:

```
1   int myDiv(int a, int b, int& err)
2   {
3     if(b==0)
4     {
5       err = 42;    // magic!
6       return -1;   // again?!
7     }
8     return a/b;
9   }
10
11  bool use(void)
12  {
13    int err = 0;
14    auto r = myDiv(10,2, err);
15    if(err)
16    {
17      // ...
18      return false;   // int? bool?
19    }
20    assert(r==5);
21    return true;
22  }
```

\+ Proste

\+ Zlokalizowane



– Znów magia

– Zawsze sprawdzane

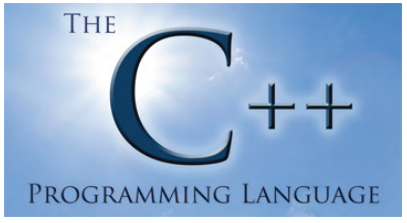
– Trzeba pamiętać o sprawdzeniu

– 2 funkcje, różne argumenty błędów == translacja

A teraz...

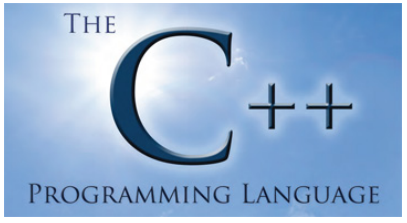
Standard

- Obserwowalne efekty dla:



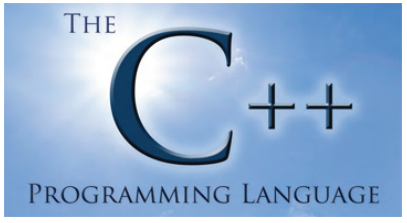
Standard

- Obserwowalne efekty dla:
  - *throw sth*



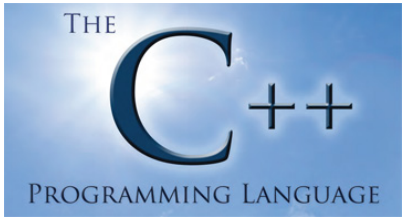
Standard

- Obserwowalne efekty dla:
  - *throw sth*
  - *catch(...)*



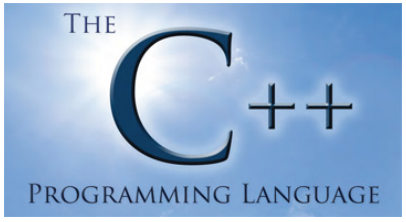
Standard

- Obserwowalne efekty dla:
  - *throw sth*
  - *catch(...)*
  - *std::exception_ptr* i spółka...



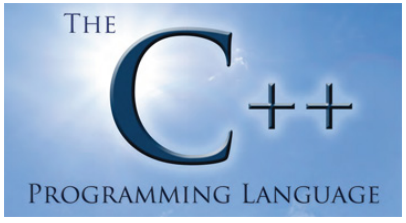
Standard

- Obserwowalne efekty dla:
  - *throw sth*
  - *catch(...)*
  - *std::exception_ptr* i spółka...
  - Nieobsłużonych wyjątków



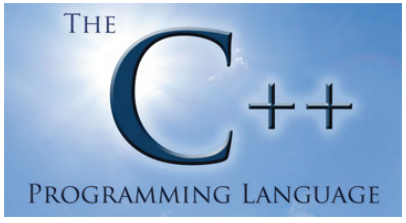
Standard

- Obserwowalne efekty dla:
  - *throw sth*
  - *catch(...)*
  - *std::exception_ptr* i spółka...
  - Nieobsłużonych wyjątków
  - Przypadków szczególnych



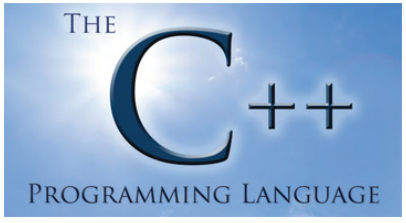
Standard

- Obserwowalne efekty dla:
  - *throw sth*
  - *catch(...)*
  - *std::exception_ptr* i spółka...
  - Nieobsłużonych wyjątków
  - Przypadków szczególnych
- Praktycznie nic o działaniu



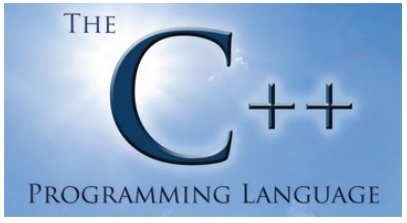
Standard

- Obserwowalne efekty dla:
  - *throw sth*
  - *catch(...)*
  - *std::exception_ptr* i spółka...
  - Nieobsłużonych wyjątków
  - Przypadków szczególnych
- Praktycznie nic o działaniu
  - Gdzie wyjątek rezyduje?



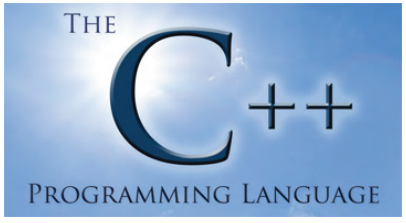
Standard

- Obserwowalne efekty dla:
  - *throw sth*
  - *catch(...)*
  - *std::exception_ptr* i spółka...
  - Nieobsłużonych wyjątków
  - Przypadków szczególnych
- Praktycznie nic o działaniu
  - Gdzie wyjątek rezyduje?
  - Kiedy jest kopiowany/przenoszony (no – prawie nic...)?



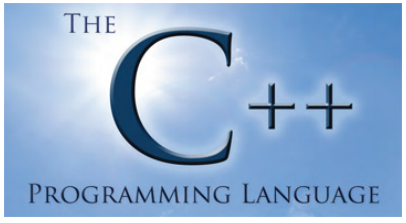
Standard

- Obserwowalne efekty dla:
  - *throw sth*
  - *catch(...)*
  - *std::exception_ptr* i spółka...
  - Nieobsłużonych wyjątków
  - Przypadków szczególnych
- Praktycznie nic o działaniu
  - Gdzie wyjątek rezyduje?
  - Kiedy jest kopiowany/przenoszony (no – prawie nic...)?
- Często używane (np. STL)
- Reszta – implementacja!



Standard

- Obserwowalne efekty dla:
  - *throw sth*
  - *catch(...)*
  - *std::exception_ptr* i spółka...
  - Nieobsłużonych wyjątków
  - Przypadków szczególnych



- Praktycznie nic o działaniu
  - Gdzie wyjątek rezyduje?
  - Kiedy jest kopiowany/przenoszony (no – prawie nic...)?
- Często używane (np. STL)
- Reszta – implementacja!
- Stosowane podejścia:
  - „Code approach”
  - „Table approach”

A teraz...

Sposób działania

- Podejście dynamiczne

Sposób działania

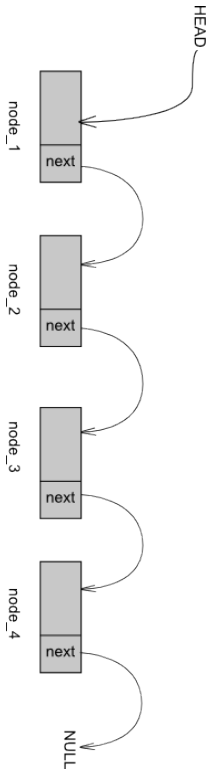
- Podejście dynamiczne
- Stos EH

Sposób działania

- Podejście dynamiczne
- Stos EH
- Lista operacji

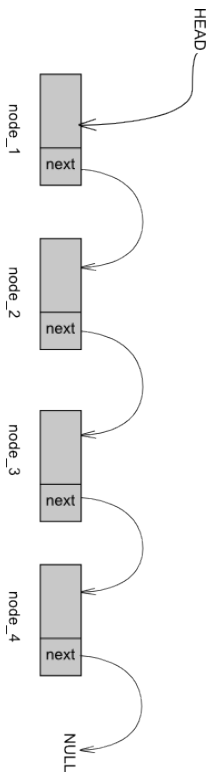
Sposób działania

- Podejście dynamiczne
- Stos EH
- Lista operacji



Sposób działania

- Podejście dynamiczne
- Stos EH
- Lista operacji




```
1   struct MyErr { /* ... */ };
2   struct Data { /* ... */ };
3   bool doSth(void)
4   {
5     Data d1;
6     try {
7       Data d2;
8       { Data d3;
9       }
10      if( time(nullptr)%2==0 ) {
11        throw MyErr{};
12      }
13    }
14    catch(MyErr const&)
15    {
16      std::cerr << "oops...\n";
17      return false;
18    }
19    return true;
20  }
```

Uproszczony przykład

```
1   struct MyErr { /* ... */ };
2   struct Data { /* ... */ };
3   bool doSth(void)
4   {
5     Data d1;
6     // EH->stack->push(&d1)
7     try {
8       // # entering try-catch
9       // tmp = EH->stack
10      // EH->stack->clear()
11      Data d2;
12      // EH->stack->push(&d2)
13      { Data d3;
14        // EH->stack->push(&d3)
15        // # scope exit:
16        // ~d3();
17        // EH->stack->pop();
18      }
19      if( time(nullptr)%2==0 ) {
20        throw MyErr{};
21        // EH->ex = MyErr{}
22        // while( not EH->stack->empty() ) {
23        //   EH->stack->top()->~()
24        //   EH->stack->pop()
25        // }
26        // EH->stack = tmp
27          // jmp NEAREST_CATCH
28      }
29      // ~d2()
30      // EH->stack->pop()
31      // EH->stack = tmp
32    }
33    // NEAREST_CATCH:
34    catch(MyErr const&)
35    // if(typeof(EH->ex)==MyErr)
36    {
37      std::cerr << "oops...\n";
38      // ~d1()
39      // EH->stack->pop()
40      return false;
41    }
42    // catch(...)
43    // {
44    //   ~d1()
45    //   EH->stack->pop()
46    //   jmp NEXT_CATCH # well - sort of... ;-)
47    // }
48    // ~d1()
49    // EH->stack->pop()
50    return true;
51  }
```

Właściwości

- Podejście wymyślone ~1992*r*

Właściwości

- Podejście wymyślone ~1992*r*
- Narzut czasu wykonania (instrukcje)

Właściwości

- Podejście wymyślone ~1992*r*
- Narzut czasu wykonania (instrukcje)
- Narzut pamięciowy (dynamiczne struktury danych)

Właściwości

- Podejście wymyślone ~1992*r*
- Narzut czasu wykonania (instrukcje)
- Narzut pamięciowy (dynamiczne struktury danych)
- Najlepsze implementacje – ~6%

Właściwości

- Podejście wymyślone ~1992*r*
- Narzut czasu wykonania (instrukcje)
- Narzut pamięciowy (dynamiczne struktury danych)
- Najlepsze implementacje – ~6%
- Wyjątki vs. Cfront

Właściwości

- Podejście wymyślone ~1992*r*
- Narzut czasu wykonania (instrukcje)
- Narzut pamięciowy (dynamiczne struktury danych)
- Najlepsze implementacje – ~6%
- Wyjątki vs. Cfront
- Korzenie „złych wyjątków”

Właściwości

- Podejście wymyślone ~1992*r*
- Narzut czasu wykonania (instrukcje)
- Narzut pamięciowy (dynamiczne struktury danych)
- Najlepsze implementacje – ~6%
- Wyjątki vs. Cfront
- Korzenie „złych wyjątków”
- Legendy o zasobożerności

A teraz...

Sposób działania

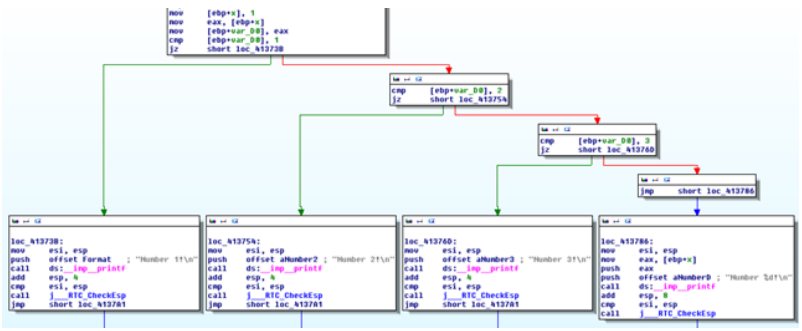
- Podejście statyczne
- Obsługa generowana w czasie kompilacji

Sposób działania

- Podejście statyczne
- Obsługa generowana w czasie kompilacji
- Mapowanie IP na kod obsługi wyjątku

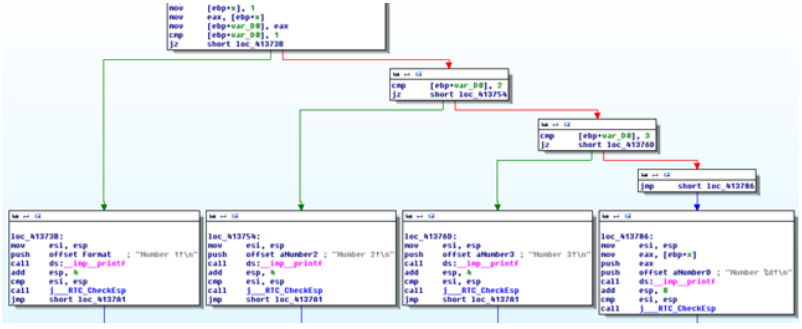
Sposób działania

- Podejście statyczne
- Obsługa generowana w czasie kompilacji
- Mapowanie IP na kod obsługi wyjątku



Sposób działania

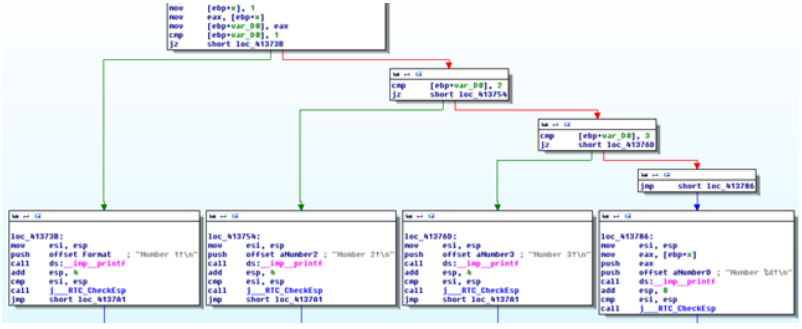
- Podejście statyczne
- Obsługa generowana w czasie kompilacji
- Mapowanie IP na kod obsługi wyjątku



- Osobny kod dla każdego scenariusza

Sposób działania

- Podejście statyczne
- Obsługa generowana w czasie kompilacji
- Mapowanie IP na kod obsługi wyjątku



- Osobny kod dla każdego scenariusza
- Brak kodu w przypadku „niewyjątkowym”

Uproszczony przykład

```
1   struct MyErr { /* ... */ };
2   struct Data { /* ... */ };
3   bool doSth(void)
4   {
5     Data d1;
6     try {
7       Data d2;
8       { Data d3;
9       }
10      if( time(nullptr)%2==0 ) {
11        throw MyErr{};
12      }
13    }
14    catch(MyErr const&)
15    {
16      std::cerr << "oops...\n";
17      return false;
18    }
19    return true;
20  }
```

Uproszczony przykład

```
1   struct MyErr { /* ... */ };
2   struct Data { /* ... */ };
3   bool doSth(void)
4   {
5     Data d1;
6     // if(EH->ex) jmp HANDLE_IP_10 # !
7     try {
8       Data d2;
9       // if(EH->ex) jmp HANDLE_IP_20 # !
10      { Data d3;
11        // if(EH->ex) jmp HANDLE_IP_30 # !
12      }
13      if( time(nullptr)%2==0 ) {
14        throw MyErr{};
15        // EH->ex = MyErr{}
16        // jmp HANDLE_IP_42
17      }
18    }
19    catch(MyErr const&)
20    {
21      // MYERR_EXCEPTION_HANDLE:
22      std::cerr << "oops...\n";
23      // ~d1()
24      return false;
25    }
26    // ~d1()
27    return true;
28  }
29  // HANDLE_IP_30:
30  //   ~d3()
31  // HANDLE_IP_42:
32  // HANDLE_IP_20:
33  //   ~d2()
34  //   if(typeof(EH->ex)==MyErr)
35  //     jmp MYERR_EXCEPTION_HANDLE
36  // HANDLE_IP_10:
37  //   ~d1()
38  //   jmp EH->parent_ex_handle_addr
```

Uproszczony przykład

```
1   struct MyErr { /* ... */ };
2   struct Data { /* ... */ };
3   bool doSth(void)
4   {
5     Data d1;
6     // if(EH->ex) jmp HANDLE_IP_10 # !
7     try {
8       Data d2;
9       // if(EH->ex) jmp HANDLE_IP_20 # !
10      { Data d3;
11        // if(EH->ex) jmp HANDLE_IP_30 # !
12      }
13      if( time(nullptr)%2==0 ) {
14        throw MyErr{};
15        // EH->ex = MyErr{}
16        // jmp HANDLE_IP_42
17      }
18    }
19    catch(MyErr const&)
20    {
21      // MYERR_EXCEPTION_HANDLE:
22      std::cerr << "oops...\n";
23      // ~d1()
24      return false;
25    }
26    // ~d1()
27    return true;
28  }
29  // HANDLE_IP_30:
30  //   ~d3()
31  // HANDLE_IP_42:
32  // HANDLE_IP_20:
33  //   ~d2()
34  //   if(typeof(EH->ex)==MyErr)
35  //     jmp MYERR_EXCEPTION_HANDLE
36  // HANDLE_IP_10:
37  //   ~d1()
38  //   jmp EH->parent_ex_handle_addr
```

- *if(EH->ex)* – lokacja

Właściwości

- Podejście wymyślone ~1994*r*

Właściwości

- Podejście wymyślone ~1994*r*
- Brak narzut czasu wykonania (jeśli brak wyjątku)

Właściwości

- Podejście wymyślone ~1994*r*
- Brak narzut czasu wykonania (jeśli brak wyjątku)
- Brak narzutu pamięciowego

Właściwości

- Podejście wymyślone ~1994*r*
- Brak narzut czasu wykonania (jeśli brak wyjątku)
- Brak narzutu pamięciowego
- Nie da się bez wsparcia kompilatora
- Cfront 1993*r*. – R.I.P.

Właściwości

- Podejście wymyślone ~1994*r*
- Brak narzut czasu wykonania (jeśli brak wyjątku)
- Brak narzutu pamięciowego
- Nie da się bez wsparcia kompilatora
- Cfront 1993*r*. – R.I.P.
- Zwiększa rozmiar programu (~5 – 20%)

Właściwości

- Podejście wymyślone ~1994*r*
- Brak narzut czasu wykonania (jeśli brak wyjątku)
- Brak narzutu pamięciowego
- Nie da się bez wsparcia kompilatora
- Cfront 1993*r*. – R.I.P.
- Zwiększa rozmiar programu ($\sim 5 - 20\%$)
- W przypadku braku wyjątku potencjalnie najszybszy

Właściwości

- Podejście wymyślone ~1994*r*
- Brak narzut czasu wykonania (jeśli brak wyjątku)
- Brak narzutu pamięciowego
- Nie da się bez wsparcia kompilatora
- Cfront 1993*r*. – R.I.P.
- Zwiększa rozmiar programu ($\sim 5 - 20\%$)
- W przypadku braku wyjątku potencjalnie najszybszy
- Trudno przewidywalny czas obsługi wyjątku

Właściwości

- Podejście wymyślone ~1994*r*
- Brak narzut czasu wykonania (jeśli brak wyjątku)
- Brak narzutu pamięciowego
- Nie da się bez wsparcia kompilatora
- Cfront 1993*r*. – R.I.P.
- Zwiększa rozmiar programu ($\sim 5 - 20\%$)
- W przypadku braku wyjątku potencjalnie najszybszy
- Trudno przewidywalny czas obsługi wyjątku
- Znane i lubiane od ~1.5 dekady

A teraz...

O pomiarach

- Wyjątki vs. dowolnie-wybrana-alternatywa

O pomiarach

- Wyjątki vs. dowolnie-wybrana-alternatywa
- Tu: argument wyjściowy

O pomiarach

- Wyjątki vs. dowolnie-wybrana-alternatywa
- Tu: argument wyjściowy
- Zakres („footprintność”):

O pomiarach

- Wyjątki vs. dowolnie-wybrana-alternatywa
- Tu: argument wyjściowy
- Zakres („footprintność”):
  - Rozmiar kody wynikowego

O pomiarach

- Wyjątki vs. dowolnie-wybrana-alternatywa
- Tu: argument wyjściowy
- Zakres („footprintność”):
  - Rozmiar kody wynikowego
  - Czas wykonania:
    - Normalny przebieg
    - Zgłoszony błąd

O pomiarach

- Wyjątki vs. dowolnie-wybrana-alternatywa
- Tu: argument wyjściowy
- Zakres („footprintność”):
  - Rozmiar kody wynikowego
  - Czas wykonania:
    - Normalny przebieg
    - Zgłoszony błąd
  - Zajętość pamięci (stos)

O pomiarach

- Wyjątki vs. dowolnie-wybrana-alternatywa
- Tu: argument wyjściowy
- Zakres („footprintność”):
  - Rozmiar kody wynikowego
  - Czas wykonania:
    - Normalny przebieg
    - Zgłoszony błąd
  - Zajętość pamięci (stos)
- Średnia z 4 wykonań

O pomiarach

- Wyjątki vs. dowolnie-wybrana-alternatywa
- Tu: argument wyjściowy
- Zakres („footprintność”):
  - Rozmiar kody wynikowego
  - Czas wykonania:
    - Normalny przebieg
    - Zgłoszony błąd
  - Zajętość pamięci (stos)
- Średnia z 4 wykonań
- Kompilatory:
  - GCC 4.8.2
  - Clang 3.2

O pomiarach

- Wyjątki vs. dowolnie-wybrana-alternatywa
- Tu: argument wyjściowy
- Zakres („footprintność”):
  - Rozmiar kody wynikowego
  - Czas wykonania:
    - Normalny przebieg
    - Zgłoszony błąd
  - Zajętość pamięci (stos)
- Średnia z 4 wykonań
- Kompilatory:
  - GCC 4.8.2
  - Clang 3.2
- Kompilacja bez LTO
- *stdout* > */dev/null*

Hello world!

- Wyjątki

```
1 #include <cstdio>
2
3 int main(void)
4 {
5   printf("hello world!\n");
6 }
```

Hello world!

- Wyjątki

```
1  #include <cstdio>
2
3  int main(void)
4  {
5    printf("hello world!\n");
6  }
```

- Rozmiar (gcc): 4632[*B*]
- Rozmiar (clang): 4536[*B*]

Hello world!

- Wyjątki

```
1  #include <cstdio>
2
3  int main(void)
4  {
5    printf("hello world!\n");
6  }
```

- Rozmiar (gcc): 4632[*B*]
- Rozmiar (clang): 4536[*B*]

- Brak wyjątków

```
1  #include <cstdio>
2
3  int main(void)
4  {
5    printf("hello world!\n");
6  }
```

Hello world!

- Wyjątki

```
1 #include <cstdio>
2
3 int main(void)
4 {
5   printf("hello world!\n");
6 }
```

- Rozmiar (gcc): 4632[*B*]
- Rozmiar (clang): 4536[*B*]

- Brak wyjątków

```
1 #include <cstdio>
2
3 int main(void)
4 {
5   printf("hello world!\n");
6 }
```

- Rozmiar (gcc): 4632[*B*]
- Rozmiar (clang): 4536[*B*]

Hello funkcja – kod

```
1   #include <cstdio>
2   #include <cstdlib>
3   #include "foo.hpp"
4
5   int main(int argc, char** argv)
6   {
7     try
8     {
9       const long n = atol(argv[1]);
10      for(long i=0; i<n; ++i)
11        foo();
12    }
13    catch(...)
14    {
15      printf("error!\n");
16      return 1;
17    }
18  }
```

Hello funkcja – kod

```
1  #include <cstdio>
2  #include <cstdlib>
3  #include "foo.hpp"
4
5  int main(int argc, char** argv)
6  {
7    try
8    {
9      const long n = atol(argv[1]);
10     for(long i=0; i<n; ++i)
11       foo();
12   }
13   catch(...)
14   {
15     printf("error!\n");
16     return 1;
17   }
18 }
```

```
1  #include <cstdio>
2  #include "foo.hpp"
3
4  void foo(void)
5  {
6    printf("hello world!\n");
7  }
```

Hello funkcja – kod

```
1  #include <cstdio>
2  #include <cstdlib>
3  #include "foo.hpp"
4
5  int main(int argc, char** argv)
6  {
7    try
8    {
9      const long n = atol(argv[1]);
10     for(long i=0; i<n; ++i)
11       foo();
12   }
13   catch(...)
14   {
15     printf("error!\n");
16     return 1;
17   }
18 }
```

```
1  #include <cstdio>
2  #include "foo.hpp"
3
4  void foo(void)
5  {
6    printf("hello world!\n");
7  }
```

```
1  #include <cstdio>
2  #include <cstdlib>
3  #include "foo.hpp"
4
5  int main(int argc, char** argv)
6  {
7    const long n = atol(argv[1]);
8    for(long i=0; i<n; ++i)
9    {
10     if( not foo() )
11     {
12       printf("error!\n");
13       return 1;
14     }
15   }
16 }
```

Hello funkcja – kod

```cpp
#include <cstdio>
#include <cstdlib>
#include "foo.hpp"

int main(int argc, char** argv)
{
  try
  {
    const long n = atol(argv[1]);
    for(long i=0; i<n; ++i)
      foo();
  }
  catch(...)
  {
    printf("error!\n");
    return 1;
  }
}
```

```cpp
#include <cstdio>
#include "foo.hpp"

void foo(void)
{
  printf("hello world!\n");
}
```

```cpp
#include <cstdio>
#include <cstdlib>
#include "foo.hpp"

int main(int argc, char** argv)
{
  const long n = atol(argv[1]);
  for(long i=0; i<n; ++i)
  {
    if( not foo() )
    {
      printf("error!\n");
      return 1;
    }
  }
}
```

```cpp
#include <cstdio>
#include "foo.hpp"

bool foo(void)
{
  printf("hello world!\n");
  return true;
}
```

Hello funkcja – wyniki

Parametr	Wersja	GCC	Clang
Rozmiar [B]	Wyjątki	5472	5272
	Kody powrotu	4856	4752
Czas [s] (N=200M)	Wyjątki	5.12	5.08
	Kody powrotu	5.43	5.44

StackMem – implementacja

```
1  #pragma once
2  #include <cinttypes>
3
4  struct StackMem
5  {
6    StackMem(void) = delete;
7
8    static uint64_t start(void)
9    {
10     static const auto s = now();
11     return s;
12   }
13
14   static uint64_t size(void)
15   {
16     const auto as = now();
17     const auto at = start();
18     return at>as?at-as:as-at;
19   }
20
21 private:
22   static uint64_t now(void)
23   {
24     const char sp{0};
25     static_assert( sizeof(&sp)<=sizeof(uint64_t), "pointer too big" );
26     const auto addr = reinterpret_cast<uint64_t>(&sp);
27     return addr;
28   }
29 };
```

StackMem – implementacja

```
1  #pragma once
2  #include <cinttypes>
3
4  struct StackMem
5  {
6    StackMem(void) = delete;
7
8    static uint64_t start(void)
9    {
10     static const auto s = now();
11     return s;
12   }
13
14   static uint64_t size(void)
15   {
16     const auto as = now();
17     const auto at = start();
18     return at>as?at-as:as-at;
19   }
20
21 private:
22   static uint64_t now(void)
23   {
24     const char sp{0};
25     static_assert( sizeof(&sp)<=sizeof(uint64_t), "pointer too big" );
26     const auto addr = reinterpret_cast<uint64_t>(&sp);
27     return addr;
28   }
29 };
```

Hello „mem” funkcja – kod

```
1   #include <cstdio>
2   #include <cstdlib>
3   #include "foo.hpp"
4   #include "StackMem.hpp"
5   int main(int argc, char** argv)
6   {
7     try
8     {
9       StackMem::start();
10      const long n = atol(argv[1]);
11      for(long i=0; i<n; ++i)
12        foo();
13    }
14    catch(...)
15    {
16      printf("error!\n");
17      return 1;
18    }
19  }
```

Hello „mem” funkcja – kod

```
1  #include <cstdio>
2  #include <cstdlib>
3  #include "foo.hpp"
4  #include "StackMem.hpp"
5  int main(int argc, char** argv)
6  {
7    try
8    {
9      StackMem::start();
10     const long n = atol(argv[1]);
11     for(long i=0; i<n; ++i)
12       foo();
13   }
14   catch(...)
15   {
16     printf("error!\n");
17     return 1;
18   }
19 }
```

```
1  #include <cstdio>
2  #include "foo.hpp"
3  #include "StackMem.hpp"
4  void foo(void)
5  {
6    const auto now = StackMem::size();
7    printf("hello␣world!␣(%lu[B])\n", now);
8  }
```

Hello „mem” funkcja – kod

```
1  #include <cstdio>
2  #include <cstdlib>
3  #include "foo.hpp"
4  #include "StackMem.hpp"
5  int main(int argc, char** argv)
6  {
7    try
8    {
9      StackMem::start();
10     const long n = atol(argv[1]);
11     for(long i=0; i<n; ++i)
12       foo();
13   }
14   catch(...)
15   {
16     printf("error!\n");
17     return 1;
18   }
19 }
```

```
1  #include <cstdio>
2  #include "foo.hpp"
3  #include "StackMem.hpp"
4  void foo(void)
5  {
6    const auto now = StackMem::size();
7    printf("hello␣world!␣(%lu[B])\n", now);
8  }
```

```
1  #include <cstdio>
2  #include <cstdlib>
3  #include "foo.hpp"
4  #include "StackMem.hpp"
5  int main(int argc, char** argv)
6  {
7    StackMem::start();
8    const long n = atol(argv[1]);
9    for(long i=0; i<n; ++i)
10   {
11     if( not foo() )
12     {
13       printf("error!\n");
14       return 1;
15     }
16   }
17 }
```

Hello „mem” funkcja – kod

```
1  #include <cstdio>
2  #include <cstdlib>
3  #include "foo.hpp"
4  #include "StackMem.hpp"
5  int main(int argc, char** argv)
6  {
7    try
8    {
9      StackMem::start();
10     const long n = atol(argv[1]);
11     for(long i=0; i<n; ++i)
12       foo();
13   }
14   catch(...)
15   {
16     printf("error!\n");
17     return 1;
18   }
19 }
```

```
1  #include <cstdio>
2  #include "foo.hpp"
3  #include "StackMem.hpp"
4  void foo(void)
5  {
6    const auto now = StackMem::size();
7    printf("hello␣world!␣(%lu[B])\n", now);
8  }
```

```
1  #include <cstdio>
2  #include <cstdlib>
3  #include "foo.hpp"
4  #include "StackMem.hpp"
5  int main(int argc, char** argv)
6  {
7    StackMem::start();
8    const long n = atol(argv[1]);
9    for(long i=0; i<n; ++i)
10   {
11     if( not foo() )
12     {
13       printf("error!\n");
14       return 1;
15     }
16   }
17 }
```

```
1  #include <cstdio>
2  #include "foo.hpp"
3  #include "StackMem.hpp"
4  bool foo(void)
5  {
6    const auto now = StackMem::size();
7    printf("hello␣world!␣(%lu[B])\n", now);
8    return true;
9  }
```

Hello „mem” funkcja – wyniki

Parametr	Wersja	GCC	Clang
Rozmiar [B]	Wyjątki	5912	5688
	Kody powrotu	5328	5200
Czas [s] (N=50M)	Wyjątki	6.04	6.08
	Kody powrotu	6.12	6.18
Stos [B]	Wyjątki	32	25[1]
	Kody powrotu	32	17

[1] wygenerowane nieużywane zmienne?

Fibonacci – kod (wyjątki)

```
1   #include <cstdio>
2   #include <cstdlib>
3   #include "fib.hpp"
4   #include "StackMem.hpp"
5
6   int main(int argc, char** argv)
7   {
8     try
9     {
10      StackMem::start();
11      const auto n = atoi(argv[1]);
12      const auto f = fib(n);
13      printf("fib(%d)=%d\n", n, f);
14    }
15    catch(...)
16    {
17      printf("error!\n");
18      return 1;
19    }
20  }
```

Fibonacci – kod (wyjątki)

```
1   #include <cstdio>
2   #include <cstdlib>
3   #include "fib.hpp"
4   #include "StackMem.hpp"
5
6   int main(int argc, char** argv)
7   {
8     try
9     {
10      StackMem::start();
11      const auto n = atoi(argv[1]);
12      const auto f = fib(n);
13      printf("fib(%d)=%d\n", n, f);
14    }
15    catch(...)
16    {
17      printf("error!\n");
18      return 1;
19    }
20  }
```

```
1   #include <cstdio>
2   #include "fib.hpp"
3   #include "StackMem.hpp"
4
5   bool g_done = false;
6
7   int fib(const int n)
8   {
9     if(n<2 && not g_done)
10    {
11      const auto now = StackMem::size();
12      printf("stack size is %lu[B]\n", now);
13      g_done = true;
14    }
15
16    if(n<0)
17      throw Negative{};
18    if(n<2)
19      return n;
20    return fib(n-2) + fib(n-1);
21  }
```

Fibonacci – kod (kody powrotu)

```
1   #include <cstdio>
2   #include <cstdlib>
3   #include "fib.hpp"
4   #include "StackMem.hpp"
5
6   int main(int argc, char** argv)
7   {
8     StackMem::start();
9     const auto n = atoi(argv[1]);
10    bool       ret;
11    const auto f = fib(n, ret);
12    if(not ret)
13    {
14      printf("error!\n");
15      return 1;
16    }
17    printf("fib(%d)=%d\n", n, f);
18  }
```

Fibonacci – kod (kody powrotu)

```cpp
#include <cstdio>
#include <cstdlib>
#include "fib.hpp"
#include "StackMem.hpp"

int main(int argc, char** argv)
{
  StackMem::start();
  const auto n = atoi(argv[1]);
  bool       ret;
  const auto f = fib(n, ret);
  if(not ret)
  {
    printf("error!\n");
    return 1;
  }
  printf("fib(%d)=%d\n", n, f);
}
```

```cpp
#include <cstdio>
#include "fib.hpp"
#include "StackMem.hpp"

bool g_done = false;

int fib(int n, bool& ret)
{
  if(n<2 && not g_done)
  {
    const auto now = StackMem::size();
    printf("stack size is %lu[B]\n", now);
    g_done = true;
  }

  if(n<0)
  {
    ret = false;
    return 0;
  }
  else
    ret = true;
  if(n<2)
    return n;
  return fib(n-2,ret) + fib(n-1,ret);
}
```

Fibonacci – wyniki

Parametr	Wersja	GCC	Clang
Rozmiar [B]	Wyjątki	7952	6872
	Kody powrotu	7408	5344
Czas [s] (N=42)	Wyjątki	1.89	3.80
	Kody powrotu	2.25	5.03
Stos [B]	Wyjątki	480	713
	Kody powrotu	560	1057

Wystąpienie błędów – kod (wyjątki)

```
1  #include <cstdio>
2  #include <cstdlib>
3  #include "foo.hpp"
4
5  int main(int argc, char** argv)
6  {
7    const int max = atoi(argv[2]);
8    const int n   = atoi(argv[1]);
9    int ok  = 0;
10   int err = 0;
11   for(int i=0; i<max; ++i)
12   {
13     try
14     {
15       if( foo(n) )
16         ++ok;
17     }
18     catch(...)
19     {
20       printf("error!\n");
21       ++err;
22     }
23   }
24   printf("%d, %d\n", ok, err);
25 }
```

Wystąpienie błędów – kod (wyjątki)

```cpp
#include <cstdio>
#include <cstdlib>
#include "foo.hpp"

int main(int argc, char** argv)
{
  const int max = atoi(argv[2]);
  const int n   = atoi(argv[1]);
  int ok  = 0;
  int err = 0;
  for(int i=0; i<max; ++i)
  {
    try
    {
      if( foo(n) )
        ++ok;
    }
    catch(...)
    {
      printf("error!\n");
      ++err;
    }
  }
  printf("%d, %d\n", ok, err);
}
```

```cpp
#include <cstdio>
#include "foo.hpp"

int foo(int n)
{
  if((n%2)==0)
    throw Oops{};
  printf("hello number %d!\n", n);
  return 1;
}
```

Wystąpienie błędów – kod (kody powrotu)

```
1  #include <cstdio>
2  #include <cstdlib>
3  #include "foo.hpp"
4
5  int main(int argc, char** argv)
6  {
7    const int max = atoi(argv[2]);
8    const int n   = atoi(argv[1]);
9    int ok  = 0;
10   int err = 0;
11   for(int i=0; i<max; ++i)
12   {
13     bool       ret;
14     const auto out = foo(n, ret);
15     if(not ret)
16     {
17       printf("error!\n");
18       ++err;
19       continue;
20     }
21     if(out)
22       ++ok;
23   }
24   printf("%d, %d\n", ok, err);
25 }
```

Wystąpienie błędów – kod (kody powrotu)

```cpp
#include <cstdio>
#include <cstdlib>
#include "foo.hpp"

int main(int argc, char** argv)
{
  const int max = atoi(argv[2]);
  const int n   = atoi(argv[1]);
  int ok  = 0;
  int err = 0;
  for(int i=0; i<max; ++i)
  {
    bool       ret;
    const auto out = foo(n, ret);
    if(not ret)
    {
      printf("error!\n");
      ++err;
      continue;
    }
    if(out)
      ++ok;
  }
  printf("%d, %d\n", ok, err);
}
```

```cpp
#include <cstdio>
#include "foo.hpp"

int foo(int n, bool& ret)
{
  if((n%2)==0)
  {
    ret = false;
    return -1;
  }
  ret = true;
  printf("hello number %d!\n", n);
  return 1;
}
```

Wystąpienie błędów – wyniki

Parametr	Wersja	GCC	Clang
Rozmiar [B]	Wyjątki	6016	5776
	Kody powrotu	5104	4968
Czas [s] (N=1M)	Wyjątki	3.56	3.34
	Kody powrotu	0.03	0.03

A teraz...

Blitzkrieg

System czasu rzeczywistego

System, dla którego poprawność odpowiedzi składa się z wartość oraz czasu jej udzielenia.

Blitzkrieg

System czasu rzeczywistego

System, dla którego poprawność odpowiedzi składa się z wartość oraz czasu jej udzielenia.

- Odpowiedź po czasie == zła

Blitzkrieg

System czasu rzeczywistego

System, dla którego poprawność odpowiedzi składa się z wartość oraz czasu jej udzielenia.

- Odpowiedź po czasie $==$ zła
- Miękki RT
  - Strumieniowanie multimediów
  - Gry akcji

Blitzkrieg

System czasu rzeczywistego

System, dla którego poprawność odpowiedzi składa się z wartość oraz czasu jej udzielenia.

- Odpowiedź po czasie $==$ zła
- Miękki RT
  - Strumieniowanie multimediów
  - Gry akcji
- Twardy RT
  - ABS
  - Generator PWM

Blitzkrieg

System czasu rzeczywistego

System, dla którego poprawność odpowiedzi składa się z wartość oraz czasu jej udzielenia.

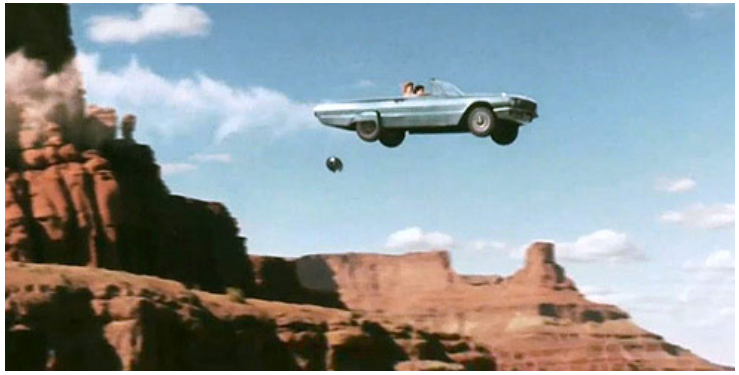
- Odpowiedź po czasie == zła
- Miękki RT
  - Strumieniowanie multimediów
  - Gry akcji
- Twardy RT
  - ABS
  - Generator PWM

Blitzkrieg

System czasu rzeczywistego

System, dla którego poprawność odpowiedzi składa się z wartość oraz czasu jej udzielenia.

- Odpowiedź po czasie == zła
- Miękki RT
  - Strumieniowanie multimediów
  - Gry akcji
- Twardy RT
  - ABS
  - Generator PWM
- RT == Przewidywalność!



Wyjątki

- Zakładamy „table approach”

[2] Translation Lookaside Buffer

Wyjątki

- Zakładamy „table approach”
- Bardzo szybkie wykonanie bezbłędne

[2] Translation Lookaside Buffer

Wyjątki

- Zakładamy „table approach”
- Bardzo szybkie wykonanie bezbłędne
- Pojawia się *throw*...

[2] Translation Lookaside Buffer

Wyjątki

- Zakładamy „table approach”
- Bardzo szybkie wykonanie bezbłędne
- Pojawia się *throw*...
  - Inny blok kodu

[2] Translation Lookaside Buffer

Wyjątki

- Zakładamy „table approach”
- Bardzo szybkie wykonanie bezbłędne
- Pojawia się *throw*...
  - Inny blok kodu
  - Możliwy nielokalny skok

[2] Translation Lookaside Buffer

Wyjątki

- Zakładamy „table approach”
- Bardzo szybkie wykonanie bezbłędne
- Pojawia się *throw*...
  - Inny blok kodu
  - Możliwy nielokalny skok
  - Wysoce prawdopodobne nietrafienie w cache

[2] Translation Lookaside Buffer

Wyjątki

- Zakładamy „table approach”
- Bardzo szybkie wykonanie bezbłędne
- Pojawia się *throw*...
  - Inny blok kodu
  - Możliwy nielokalny skok
  - Wysoce prawdopodobne nietrafienie w cache
  - Równie prawdopodobne nietrafienie w TLB[2]

[2] Translation Lookaside Buffer

Wyjątki

- Zakładamy „table approach”
- Bardzo szybkie wykonanie bezbłędne
- Pojawia się *throw*...
  - Inny blok kodu
  - Możliwy nielokalny skok
  - Wysoce prawdopodobne nietrafienie w cache
  - Równie prawdopodobne nietrafienie w TLB[2]
  - Wysoce prawdopodobne zaczytanie kodu z RAMu

[2] Translation Lookaside Buffer

Wyjątki

- Zakładamy „table approach”
- Bardzo szybkie wykonanie bezbłędne
- Pojawia się *throw*...
  - Inny blok kodu
  - Możliwy nielokalny skok
  - Wysoce prawdopodobne nietrafienie w cache
  - Równie prawdopodobne nietrafienie w TLB[2]
  - Wysoce prawdopodobne zaczytanie kodu z RAMu
  - Możliwa konieczność zaczytania kodu z dysku (swapa)

[2] Translation Lookaside Buffer

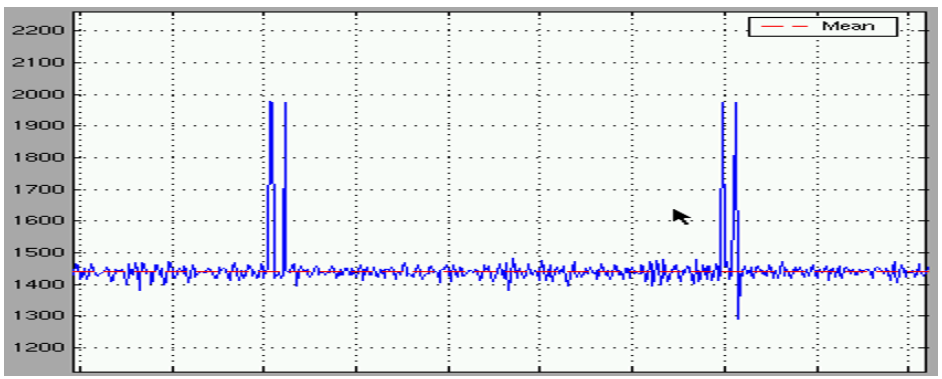
Wyjątki

- Zakładamy „table approach”
- Bardzo szybkie wykonanie bezbłędne
- Pojawia się *throw*...
  - Inny blok kodu
  - Możliwy nielokalny skok
  - Wysoce prawdopodobne nietrafienie w cache
  - Równie prawdopodobne nietrafienie w TLB[2]
  - Wysoce prawdopodobne zaczytanie kodu z RAMu
  - Możliwa konieczność zaczytania kodu z dysku (swapa)
  - Wersja (potencjalnie N razy) wolniejsza

[2] Translation Lookaside Buffer

Wyjątki

- Zakładamy „table approach”
- Bardzo szybkie wykonanie bezbłędne
- Pojawia się *throw*...
  - Inny blok kodu
  - Możliwy nielokalny skok
  - Wysoce prawdopodobne nietrafienie w cache
  - Równie prawdopodobne nietrafienie w TLB[2]
  - Wysoce prawdopodobne zaczytanie kodu z RAMu
  - Możliwa konieczność zaczytania kodu z dysku (swapa)
  - Wersja (potencjalnie N razy) wolniejsza



[2] Translation Lookaside Buffer

A teraz...

Obsługa błędów

- Wyjątki:

Obsługa błędów

- Wyjątki:
+ Umożliwiają RAII
+ Nie da się przegapić
+ Nielokalna obsługa błędów
+ Nie wymagają umownych konwencji
+ Wymagają mniej pamięci

Obsługa błędów

- Wyjątki:
+ Umożliwiają RAII
+ Nie da się przegapić
+ Nielokalna obsługa błędów
+ Nie wymagają umownych konwencji
+ Wymagają mniej pamięci
- Zwiększają rozmiar programu
- Dłuższa obsługa błędu

Obsługa błędów

- Wyjątki:
  - \+ Umożliwiają RAII
  - \+ Nie da się przegapić
  - \+ Nielokalna obsługa błędów
  - \+ Nie wymagają umownych konwencji
  - \+ Wymagają mniej pamięci
  - \- Zwiększają rozmiar programu
  - \- Dłuższa obsługa błędu
- Kody błędów:

Obsługa błędów

- Wyjątki:
  + Umożliwiają RAII
  + Nie da się przegapić
  + Nielokalna obsługa błędów
  + Nie wymagają umownych konwencji
  + Wymagają mniej pamięci
  - Zwiększają rozmiar programu
  - Dłuższa obsługa błędu

- Kody błędów:
  + Przewidywalne
  + Szybkie w przypadku błędu
  + Zajmują mało miejsca programu

Obsługa błędów

- Wyjątki:
  + Umożliwiają RAII
  + Nie da się przegapić
  + Nielokalna obsługa błędów
  + Nie wymagają umownych konwencji
  + Wymagają mniej pamięci
  - Zwiększają rozmiar programu
  - Dłuższa obsługa błędu

- Kody błędów:
  + Przewidywalne
  + Szybkie w przypadku błędu
  + Zajmują mało miejsca programu
  - Wymagają więcej pamięci
  - Magiczne wartości
  - Łatwo zapomnieć
  - Niekompatybilne z RAII
  - Zaciemniają kod
  - Wymagają umownych konwencji

Uwagi do pomiarów

- Bardzo proste przykłady
- Jeden przykład – jeden aspekt

Uwagi do pomiarów

- Bardzo proste przykłady
- Jeden przykład – jeden aspekt
- Testowanie przypadków patologicznych
  - Rekurencyjnych Fibonacci
  - Zgłaszanie w pętli

Uwagi do pomiarów

- Bardzo proste przykłady
- Jeden przykład – jeden aspekt
- Testowanie przypadków patologicznych
  - Rekurencyjnych Fibonacci
  - Zgłaszanie w pętli
- W rzeczywistości – dużo interakcji

Uwagi do pomiarów

- Bardzo proste przykłady
- Jeden przykład – jeden aspekt
- Testowanie przypadków patologicznych
  - Rekurencyjnych Fibonacci
  - Zgłaszanie w pętli
- W rzeczywistości – dużo interakcji
  - LTO (Link Time Optimization)

Uwagi do pomiarów

- Bardzo proste przykłady
- Jeden przykład – jeden aspekt
- Testowanie przypadków patologicznych
  - Rekurencyjnych Fibonacci
  - Zgłaszanie w pętli
- W rzeczywistości – dużo interakcji
  - LTO (Link Time Optimization)
  - *throw* obsługujący wielokrotne zagnieżdżenia

Uwagi do pomiarów

- Bardzo proste przykłady
- Jeden przykład – jeden aspekt
- Testowanie przypadków patologicznych
  - Rekurencyjnych Fibonacci
  - Zgłaszanie w pętli
- W rzeczywistości – dużo interakcji
  - LTO (Link Time Optimization)
  - *throw* obsługujący wielokrotne zagnieżdżenia
  - Wpływ cache na pomiary

Uwagi do pomiarów

- Bardzo proste przykłady
- Jeden przykład – jeden aspekt
- Testowanie przypadków patologicznych
  - Rekurencyjnych Fibonacci
  - Zgłaszanie w pętli
- W rzeczywistości – dużo interakcji
  - LTO (Link Time Optimization)
  - *throw* obsługujący wielokrotne zagnieżdżenia
  - Wpływ cache na pomiary
  - Przykłady mieszanie (np. *throw* co 10k zawołań)

Uwagi do pomiarów

- Bardzo proste przykłady
- Jeden przykład – jeden aspekt
- Testowanie przypadków patologicznych
  - Rekurencyjnych Fibonacci
  - Zgłaszanie w pętli
- W rzeczywistości – dużo interakcji
  - LTO (Link Time Optimization)
  - *throw* obsługujący wielokrotne zagnieżdżenia
  - Wpływ cache na pomiary
  - Przykłady mieszanie (np. *throw* co 10k zawołań)
  - Dużo możliwych błędów (23 możliwości w 8 linii?)

Uwagi do pomiarów

- Bardzo proste przykłady
- Jeden przykład – jeden aspekt
- Testowanie przypadków patologicznych
  - Rekurencyjnych Fibonacci
  - Zgłaszanie w pętli
- W rzeczywistości – dużo interakcji
  - LTO (Link Time Optimization)
  - *throw* obsługujący wielokrotne zagnieżdżenia
  - Wpływ cache na pomiary
  - Przykłady mieszanie (np. *throw* co $10k$ zawołań)
  - Dużo możliwych błędów (23 możliwości w 8 linii?)
- *throw* w przykładzie $< 3\mu s$

Kompresja do 1 slajdu

- Wyjątki:
  - „Table approach”
  - Najszybsze w normalnym przebiegu

Kompresja do 1 slajdu

- Wyjątki:
  - „Table approach”
  - Najszybsze w normalnym przebiegu
  - Trudno przewidywalny czas zgłaszania
  - Program większy o $5 - 20\%$

Kompresja do 1 slajdu

- Wyjątki:
  - „Table approach”
  - Najszybsze w normalnym przebiegu
  - Trudno przewidywalny czas zgłaszania
  - Program większy o $5 - 20\%$
- Domyślnie stosuj wyjątki

Kompresja do 1 slajdu

- Wyjątki:
  - „Table approach”
  - Najszybsze w normalnym przebiegu
  - Trudno przewidywalny czas zgłaszania
  - Program większy o $5 - 20\%$
- Domyślnie stosuj wyjątki
- . . . za wyjątkiem twardego RT
  - Wyjątki $+$ RT $==$ zły pomysł. . .
  - „Real-Time C++”, Chris Kormanyos

Kompresja do 1 slajdu

- Wyjątki:
  - „Table approach”
  - Najszybsze w normalnym przebiegu
  - Trudno przewidywalny czas zgłaszania
  - Program większy o $5 - 20\%$
- Domyślnie stosuj wyjątki
- . . . za wyjątkiem twardego RT
  - Wyjątki $+$ RT $==$ zły pomysł. . .
  - „Real-Time C++”, Chris Kormanyos
- Złoty młotek nie istnieje (sorry)

Kompresja do 1 slajdu

- Wyjątki:
  - „Table approach”
  - Najszybsze w normalnym przebiegu
  - Trudno przewidywalny czas zgłaszania
  - Program większy o $5 - 20\%$
- Domyślnie stosuj wyjątki
- . . . za wyjątkiem twardego RT
  - Wyjątki $+$ RT $==$ zły pomysł. . .
  - „Real-Time C++”, Chris Kormanyos
- Złoty młotek nie istnieje (sorry)
- Narzędzia do potrzeb

Kompresja do 1 slajdu

- Wyjątki:
  - „Table approach”
  - Najszybsze w normalnym przebiegu
  - Trudno przewidywalny czas zgłaszania
  - Program większy o $5 - 20\%$
- Domyślnie stosuj wyjątki
- . . . za wyjątkiem twardego RT
  - Wyjątki $+$ RT $==$ zły pomysł. . .
  - „Real-Time C++”, Chris Kormanyos
- Złoty młotek nie istnieje (sorry)
- Narzędzia do potrzeb
- Mierz. . .

Kompresja do 1 slajdu

- Wyjątki:
  - „Table approach”
  - Najszybsze w normalnym przebiegu
  - Trudno przewidywalny czas zgłaszania
  - Program większy o $5 - 20\%$
- Domyślnie stosuj wyjątki
- . . . za wyjątkiem twardego RT
  - Wyjątki $+$ RT $==$ zły pomysł. . .
  - „Real-Time C++”, Chris Kormanyos
- Złoty młotek nie istnieje (sorry)
- Narzędzia do potrzeb
- Mierz. . .
- Mierz. . .

Kompresja do 1 slajdu

- Wyjątki:
  - „Table approach”
  - Najszybsze w normalnym przebiegu
  - Trudno przewidywalny czas zgłaszania
  - Program większy o $5 - 20\%$
- Domyślnie stosuj wyjątki
- . . . za wyjątkiem twardego RT
  - Wyjątki $+$ RT $==$ zły pomysł. . .
  - „Real-Time C++”, Chris Kormanyos
- Złoty młotek nie istnieje (sorry)
- Narzędzia do potrzeb
- Mierz. . .
- Mierz. . .
- Mierz. . .

Do poczytania

- http://www.open-std.org/jtc1/sc22/wg21/docs/TR18015.pdf
- http://channel9.msdn.com/Shows/Going+Deep/C-and-Beyond-2012-Andrei-Alexandrescu-Systematic-Error-Handling-in-C
- http://baszerr.eu/doku.php/blog/2012/02/26/1
- http://www.hexblog.com/wp-content/uploads/2012/06/Recon-2012-Skochinsky-Compiler-Internals.pdf

Dziękuję za uwagę

Dziękuję za uwagę

Autor pragnie podziękować wszystkim którzy nie chrapali zbyt głośno, w imieniu tych, którym chrapanie innych przeszkadzało spać.