C++ Traps and Pitfalls

Petr Zemek

Principal Developer
Security/Engineering/VirusLab
http://petrzemek.net

C++ Developers' Guild, 2016-04-22

Introduction

What is a pitfall?

Introduction

What is a pitfall?

Example:

```
1 std::ifstream inputFile("C:\test.txt");
2 if (!inputFile) {
3     std::cerr << "Failed to open file!";
4 }
```

Virtual Functions and Default Parameters

```
 1 class A {
 2 public:
 3     virtual void foo(int i = 1) {
 4         std::cout << i << '\n';
 5     }
 6 };
 7
 8 class B: public A {
 9 public:
10     virtual void foo(int i = 2) override {
11         std::cout << i << '\n';
12     }
13 };
14
15 std::unique_ptr<A> p(new B);
16 p->foo();
```

Virtual Functions Inside Ctors/Dtors

```
 1 class A {
 2 public:
 3     A() { foo(); }
 4     virtual ~A() { foo(); }
 5
 6     virtual void foo() {
 7         std::cout << "A::foo()\n";
 8     }
 9 };
10
11 class B: public A {
12 public:
13     virtual void foo() override {
14         std::cout << "B::foo()\n";
15     }
16 };
17
18 std::unique_ptr<A> p(new B);
```

Evaluation Order

```
1 auto result = filterInput(
2     std::shared_ptr<Filter>(new Filter),
3     getInput()
4 );
```

Evaluation Order

```
1 auto result = filterInput(
2     std::shared_ptr<Filter>(new Filter),
3     getInput()
4 );
```

- `auto tmp = new Filter;`

Evaluation Order

```
1 auto result = filterInput(
2     std::shared_ptr<Filter>(new Filter),
3     getInput()
4 );
```

- `auto tmp = new Filter;`
- `auto input = getInput();`

Evaluation Order

```
1 auto result = filterInput(
2     std::shared_ptr<Filter>(new Filter),
3     getInput()
4 );
```

- `auto tmp = new Filter;`
- `auto input = getInput();`
- `auto filter = std::shared_ptr<Filter>(tmp);`

Evaluation Order

```
1 auto result = filterInput(
2     std::shared_ptr<Filter>(new Filter),
3     getInput()
4 );
```

- `auto tmp = new Filter;`
- `auto input = getInput();`
- `auto filter = std::shared_ptr<Filter>(tmp);`
- `auto result = filterInput(filter, input);`

Evaluation Order

```
1 auto result = filterInput(
2     std::shared_ptr<Filter>(new Filter),
3     getInput()
4 );
```

- `auto tmp = new Filter;`
- **`auto input = getInput();`**
- `auto filter = std::shared_ptr<Filter>(tmp);`
- `auto result = filterInput(filter, input);`

Object Construction

```
1 class A {
2 public:
3     // ...
4
5 };


6 A a(1);
7 A b{1};
8 A c = 1;
9 auto d = A(1);
```

Object Construction (Version I)

```
1 class A {
2 public:
3     A(int);
4
5 };


6 A a(1);              // A(int)
7 A b{1};             // A(int)
8 A c = 1;            // A(int), A(const A &)?
9 auto d = A(1);      // A(int), A(const A &)?
```

Object Construction (Version II)

```
1 class A {
2 public:
3     explicit A(int);
4
5 };


6 A a(1);              // A(int)
7 A b{1};             // A(int)
8 A c = 1;            // fails to compile
9 auto d = A(1);      // A(int), A(const A &)?
```

Object Construction (Version III)

```
1 class A {
2 public:
3     A(int);
4     A(const A &) = delete;
5 };

6 A a(1);              // A(int)
7 A b{1};              // A(int)
8 A c = 1;             // fails to compile
9 auto d = A(1);       // fails to compile
```

Object Construction (Version IV)

```
1 class A {
2 public:
3     A(int);
4     A(std::initializer_list<float>);
5 };


6 A a(1);                  // A(int)
7 A b{1};                  // A(std::initializer_list<float>)
8 A c = 1;                 // A(int), A(const A &)?
9 auto d = A(1);           // A(int), A(const A &)?
```

Object Construction (Riddle I)

```
1 class A {
2 public:
3     A();
4     A(int);
5     A(std::initializer_list<int>);
6 };


7 A b{1, 2};          // A(std::initializer_list<int>)
8 A b{1};             // A(std::initializer_list<int>)
9 A a{};              // ?
```

Object Construction (Riddle I)

```
1 class A {
2 public:
3     A();
4     A(int);
5     A(std::initializer_list<int>);
6 };


7 A b{1, 2};          // A(std::initializer_list<int>)
8 A b{1};             // A(std::initializer_list<int>)
9 A a{};              // A()
```

Object Construction (Riddle II)

```
1 auto i{1};
2 auto j = {1};
```

What are the types of `i` and `j`?

Object Construction (Riddle II)

```
1 auto i{1};
2 auto j = {1};
```

What are the types of `i` and `j`?

	`i`	`j`
C++14	`std::initializer_list<int>`	`std::initializer_list<int>`

Object Construction (Riddle II)

```
1 auto i{1};
2 auto j = {1};
```

What are the types of `i` and `j`?

	`i`	`j`
C++14	`std::initializer_list<int>`	`std::initializer_list<int>`
C++1z	`int`	`std::initializer_list<int>`

Object Construction (Riddle II)

```
1 auto i{1};
2 auto j = {1};
```

What are the types of `i` and `j`?

	`i`	`j`
C++14	`std::initializer_list<int>`	`std::initializer_list<int>`
C++1z	`int`	`std::initializer_list<int>`
GCC 4.9 (C++14)	`std::initializer_list<int>`	`std::initializer_list<int>`
GCC 5 (C++14)	`int`	`std::initializer_list<int>`

Object Construction (Riddle II)

```
1 auto i{1};
2 auto j = {1};
```

What are the types of `i` and `j`?

	`i`	`j`
C++14	`std::initializer_list<int>`	`std::initializer_list<int>`
C++1z	`int`	`std::initializer_list<int>`
GCC 4.9 (C++14)	`std::initializer_list<int>`	`std::initializer_list<int>`
GCC 5 (C++14)	`int`	`std::initializer_list<int>`
Clang 3.7 (C++14)	`std::initializer_list<int>`	`std::initializer_list<int>`
Clang 3.8 (C++14)	`int`	`std::initializer_list<int>`

Object Construction (Riddle II)

```
1 auto i{1};
2 auto j = {1};
```

What are the types of `i` and `j`?

	`i`	`j`
C++14	`std::initializer_list<int>`	`std::initializer_list<int>`
C++1z	`int`	`std::initializer_list<int>`
GCC 4.9 (C++14)	`std::initializer_list<int>`	`std::initializer_list<int>`
GCC 5 (C++14)	`int`	`std::initializer_list<int>`
Clang 3.7 (C++14)	`std::initializer_list<int>`	`std::initializer_list<int>`
Clang 3.8 (C++14)	`int`	`std::initializer_list<int>`
MSVC 2015	`int`	`std::initializer_list<int>`

Object Construction (Riddle III)

```
1 class A {
2 public:
3     A();
4     A(const A &) = delete;
5 };
6
7 auto a = A();  // fails to compile
8
9 auto p = std::unique_ptr<int>();  // OK (why?)
```

Move Semantics

```
 1 class Processor {
 2 public:
 3     // ...
 4
 5     void process(const std::string d) {
 6         // ...
 7
 8         data = std::move(d);
 9     }
10
11 private:
12     std::string data;
13
14     // ...
15 };
```

Move Semantics (Huh?)

```
 1 class Processor {
 2 public:
 3     // ...
 4
 5     void process(const std::string d) {
 6         // ...
 7
 8         data = std::move(d);
 9     }
10
11 private:
12     std::string data;
13
14     // ...
15 };
```

Move Semantics (Explanation)

```
 1 class string {
 2 public:
 3     // ...
 4
 5     string &operator=(const string &other);
 6     string &operator=(string &&other);
 7
 8     // ...
 9 };
10
11 // ...
12
13     void process(const std::string d) {
14         // ...
15
16         data = std::move(d); // copies d!
17     }
```

Operations Over Integers

```
std::cout << -sizeof(char) << '\n';
```

Operations Over Integers

```
std::cout << -sizeof(char) << '\n';
```

Prints:

```
4294967295                  (32b system)
18446744073709551615        (64b system)
```

Comparing Signed and Unsigned Integers

```
1 int i = -1;
2 unsigned int j = 1;
3
4 if (i < j) {
5     // ...
6 }
```

Iterating Over Containers

```
1 std::vector<int> v;
2 // ...
3
4 for (unsigned int i = 0; i < v.size(); ++i) {
5     // ...
6 }
```

Iterating Over Containers

```
1 std::vector<int> v;
2 // ...
3
4 for (unsigned int i = 0; i < v.size(); ++i) {
5     // ...
6 }
```

On a 64b system:

```
i = 0           // v.size() == 4294967296 (UINT_MAX + 1)
i = 1           // v.size() == 4294967296
...             // v.size() == 4294967296
i = 4294967294  // v.size() == 4294967296
i = 4294967295  // v.size() == 4294967296
i = 0           // v.size() == 4294967296
i = 1           // v.size() == 4294967296
...
```

Default Lambda Captures

```
 1 class DivFilter {
 2 public:
 3     DivFilter(int divisor);
 4
 5     void addFilter(Filters &filters) {
 6         filters.push_back(
 7             [=](int value) {
 8                 return value % divisor == 0;
 9             }
10         );
11     }
12
13 private:
14     int divisor;
15 };
```

Default Lambda Captures (Continued)

```
 1 class DivFilter {
 2 public:
 3     //...
 4     void addFilter(Filters &filters) {
 5         filters.push_back(
 6             [divisor](int value) {
 7                 return value % divisor == 0;
 8             }
 9         );
10     }
11
12 private:
13     int divisor;
14 };
```

Default Lambda Captures (Continued)

```
 1 class DivFilter {
 2 public:
 3     //...
 4     void addFilter(Filters &filters) {
 5         filters.push_back(
 6             [divisor](int value) {
 7                 return value % divisor == 0;
 8             }
 9         );
10     }
11
12 private:
13     int divisor;
14 };
```

Fails to compile:

```
error: capture of non-variable 'DivFilter::divisor'
    [divisor](int value) {
     ^
```

Default Lambda Captures (Explanation)

```
 1 class DivFilter {
 2 public:
 3     //...
 4     void addFilter(Filters &filters) {
 5         filters.push_back(
 6             [this](int value) {
 7                 return value % this->divisor == 0;
 8             }
 9         );
10     }
11
12 private:
13     int divisor;
14 };
```

Default Lambda Captures (Solution)

```
 1 class DivFilter {
 2 public:
 3     //...
 4     void addFilter(Filters &filters) {
 5         filters.push_back(
 6             [divisor = divisor](int value) {
 7                 return value % divisor == 0;
 8             }
 9         );
10     }
11
12 private:
13     int divisor;
14 };
```

References and Further Information

Scott Meyers

Effective Modern C++

O'Reilly Media, 2014, 336 pages

Miroslav Virius

Pasti a propasti jazyka C++

Computer Press, 2005, 376 pages

- Stephan T. Lavavej: Don't Help the Compiler (GoingNative'13)
  - https://www.youtube.com/watch?v=AKtHxKJRwp4
- Scott Meyers: An Effective C++11/14 Sampler (GoingNative'13)
  - https://www.youtube.com/watch?v=BezbcQluCsY
- Piotr Padlewski: C++ WAT (CppCon'15)
  - https://www.youtube.com/watch?v=rNNnPrMHsAA

Nick Lewycky's realloc

http://blog.regehr.org/archives/767

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 int main() {
 5     int *p = malloc(sizeof(int));
 6     int *q = realloc(p, sizeof(int));
 7     *p = 1;
 8     *q = 2;
 9     if (p == q) {
10         printf("%d %d\n", *p, *q);
11     }
12     return 0;
13 }
```

Nick Lewycky's realloc

http://blog.regehr.org/archives/767

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 int main() {
 5     int *p = malloc(sizeof(int));
 6     int *q = realloc(p, sizeof(int));
 7     *p = 1;
 8     *q = 2;
 9     if (p == q) {
10         printf("%d %d\n", *p, *q);
11     }
12     return 0;
13 }
```

```
$ clang -O2 -o realloc realloc.c
$ ./realloc
1 2
```

Explanation of Nick Lewycky's realloc

Decompiled code (e.g. via retdec.com):

```
 1 #include <stdint.h>
 2 #include <stdio.h>
 3 #include <stdlib.h>
 4
 5 int main() {
 6     char *mem = malloc(4);
 7     char *mem2 = realloc(mem, 4);
 8     *(int32_t *)mem = 1;
 9     *(int32_t *)mem2 = 2;
10     if (mem == mem2) {
11         printf("%d %d\n", 1, 2);
12     }
13     return 0;
14 }
```