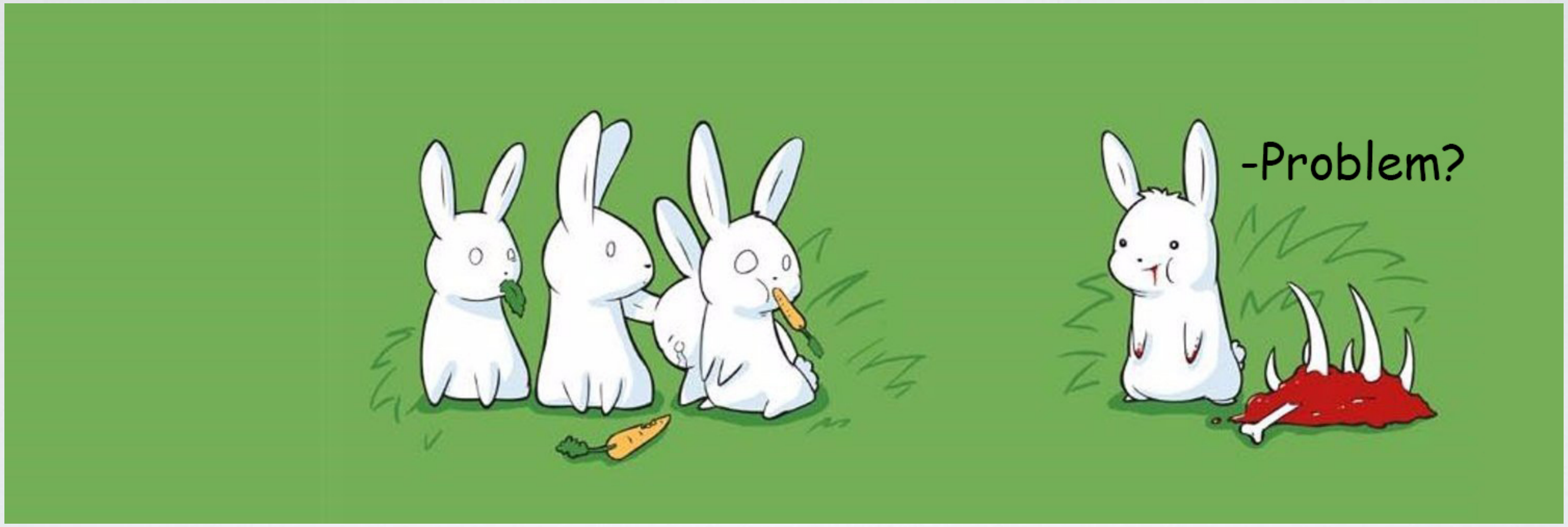
ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

EXCEPTIONS



ABNORMAL PROGRAM TERMINATION ABORT()

```
...

double hmean(double a, double b)
{
    if (std::abs(a+b) < epsilon)
    {
        std::cout << "untenable arguments to hmean()\n";
        std::abort();
    }
    return 2.0 * a * b / (a + b);
}

...
```

ABNORMAL PROGRAM TERMINATION ABORT()

```
...
int main()
{
   double x, y, z;

   std::cout << "Enter two numbers: ";
   while (std::cin >> x >> y)
   {
      z = hmean(x, y);
      std::cout << "Harmonic mean of " << x << " and " << y
         << " is " << z << '\n';
      std::cout << "Enter next set of numbers <q to quit>: ";
   }

   std::cout << "Bye!\n";

   return 0;
}
```

RETURN ERROR CODES

```
...

bool hmean(double a, double b, double* ans)
{
    if (std::abs(a+b) < epsilon)
    {
        *ans = std::numeric_limits<double>::infinity();
        return false;
    }

    *ans = 2.0 * a * b / (a + b);
    return true;
}

...
```

MACRO ERRNO

```
...
int main()
{
   double not_a_number = std::log(-1.0);
   if (errno == EDOM) {
      std::cout << "log(-1) failed: "
                << std::strerror(errno) << '\n';
   }

   return 0;
}
```

EXCEPTIONS

- Сигнализирование об ошибках в C:
  - Код возврата функции.
  - Глобальная переменная (**errno**).
  - **Можно проигнорировать!**
- Исключения **нельзя** проигнорировать! Muahahaha

EXCEPTIONS

- Генерация исключения throw.
- Перехват исключения обработчиком catch.
- Использование блока try.

EXCEPTIONS

```
...

double hmean(double a, double b)
{
    if (std::abs(a+b) < epsilon)
    {
        throw "bad hmean() arguments: a = -b not allowed";
    }
    return 2.0 * a * b / (a + b);
}

...
```

EXCEPTIONS

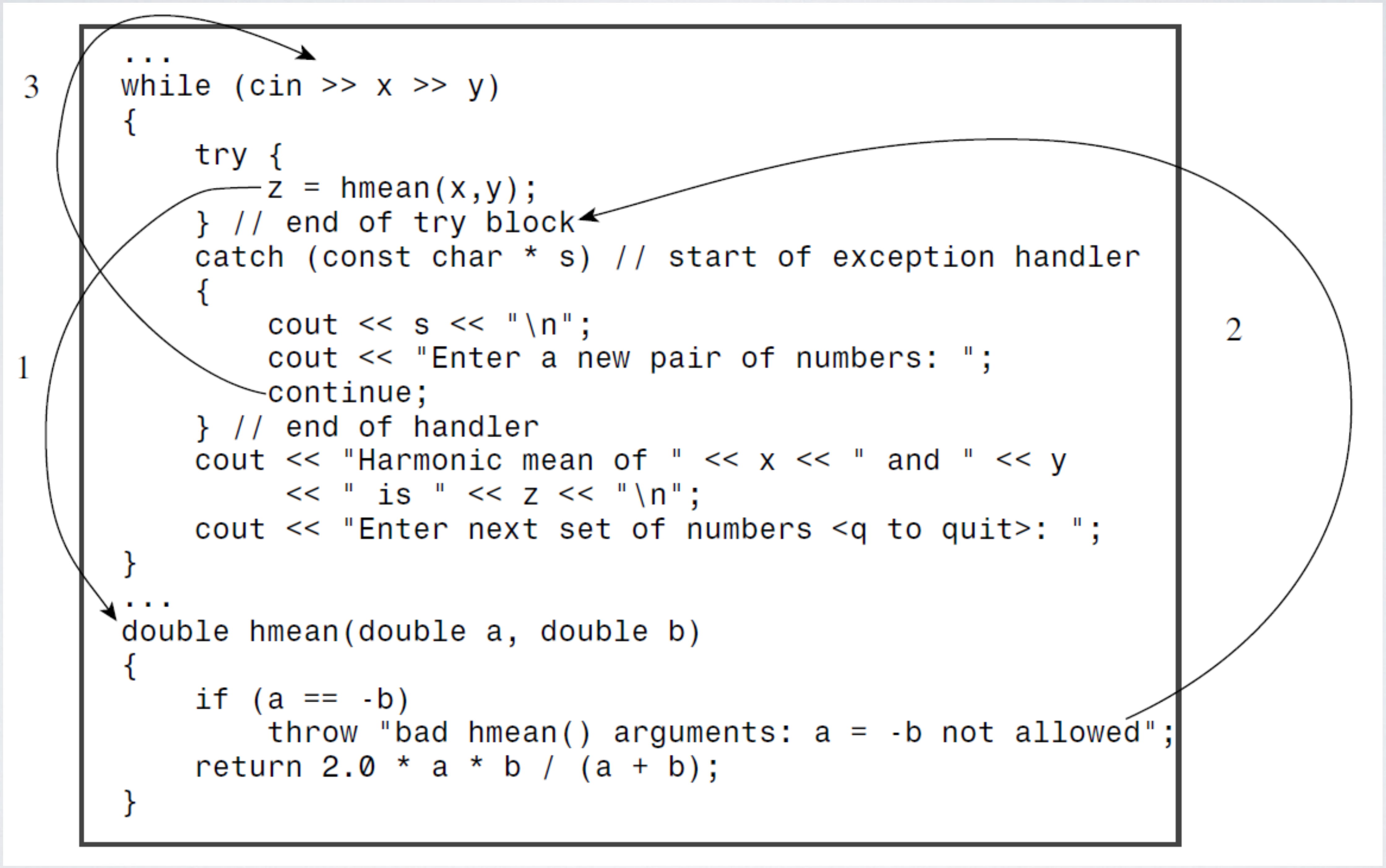
```
...
int main()
{
    double x, y, z;

    std::cout << "Enter two numbers: ";
    while (std::cin >> x >> y)
    {
        try {                        // start of try block
            z = hmean(x,y);
        }                            // end of try block
        catch (const char* s)        // start of exception handler
        {
            std::cout << s << '\n';
            std::cout << "Enter a new pair of numbers: ";
            continue;
        }                            // end of handler
        std::cout << "Harmonic mean of " << x << " and " << y << " is " << z << '\n';
        std::cout << "Enter next set of numbers <q to quit>: ";
    }

    std::cout << "Bye!\n";

    return 0;
}
```

EXCEPTIONS



```
...
while (cin >> x >> y)
{
    try {
        z = hmean(x,y);
    } // end of try block
    catch (const char * s) // start of exception handler
    {
        cout << s << "\n";
        cout << "Enter a new pair of numbers: ";
        continue;
    } // end of handler
    cout << "Harmonic mean of " << x << " and " << y
         << " is " << z << "\n";
    cout << "Enter next set of numbers <q to quit>: ";
}
...
double hmean(double a, double b)
{
    if (a == -b)
        throw "bad hmean() arguments: a = -b not allowed";
    return 2.0 * a * b / (a + b);
}
```

USING OBJECTS AS EXCEPTIONS

```cpp
class bad_hmean final
{
private:
    double v1;
    double v2;
public:
    bad_hmean(double a = 0, double b = 0) : v1(a), v2(b){}
    void msg() const;
};

inline void bad_hmean::msg() const
{
    std::cout << "hmean(" << v1 << ", " << v2 << "): "
              << "invalid arguments: a = -b\n";
}
```

USING OBJECTS AS EXCEPTIONS

```
...

double hmean(double a, double b)
{
    if (std::abs(a+b) < epsilon)
    {
        throw bad_hmean(a, b);
    }
    return 2.0 * a * b / (a + b);
}

...
```

USING OBJECTS AS EXCEPTIONS

```
...
int main()
{
    double x, y, z;

    std::cout << "Enter two numbers: ";
    while (std::cin >> x >> y)
    {
        try {                     // start of try block
            z = hmean(x,y);
            std::cout << "Harmonic mean of " << x << " and " << y << " is " << z << '\n';
            std::cout << "Enter next set of numbers <q to quit>: ";

        }                               // end of try block
        catch (const bad_hmean& bh)     // start of exception handler
        {
            bh.msg();
            std::cout << "Try again.\n";
        }                         // end of handler
    }

    std::cout << "Bye!\n";

    return 0;
}
```

USING OBJECTS AS EXCEPTIONS

```
...

double gmean(double a, double b)
{
    if (a * b < 0)
    {
        throw bad_gmean(a, b);
    }
    return std::sqrt(a * b);
}

...
```

USING OBJECTS AS EXCEPTIONS

```
class bad_gmean final
{
private:
    double v1;
    double v2;
public:
    bad_gmean(double a = 0, double b = 0) : v1(a), v2(b){}
    std::string msg() const;
};

inline std::string bad_gmean::msg() const
{
    return "gmean() arguments should be >= 0\n";
}
```

USING OBJECTS AS EXCEPTIONS

```
int main()
{
    double x, y, z;

    std::cout << "Enter two numbers: ";
    while (std::cin >> x >> y)
    {
        try {                       // start of try block
            z = hmean(x,y);
            z = gmean(x,y);
        }                                 // end of try block
        catch (const bad_hmean& bh)       // start of catch block
        {
            bh.msg();
            std::cout << "Try again.\n";
           continue;
        }
        catch (const bad_gmean& bg)
        {
            std::cout << bg.msg();
            break;
        }                                 // end of catch block
    }
    ...
}
```

EXCEPTION SPECIFICATIONS C++98

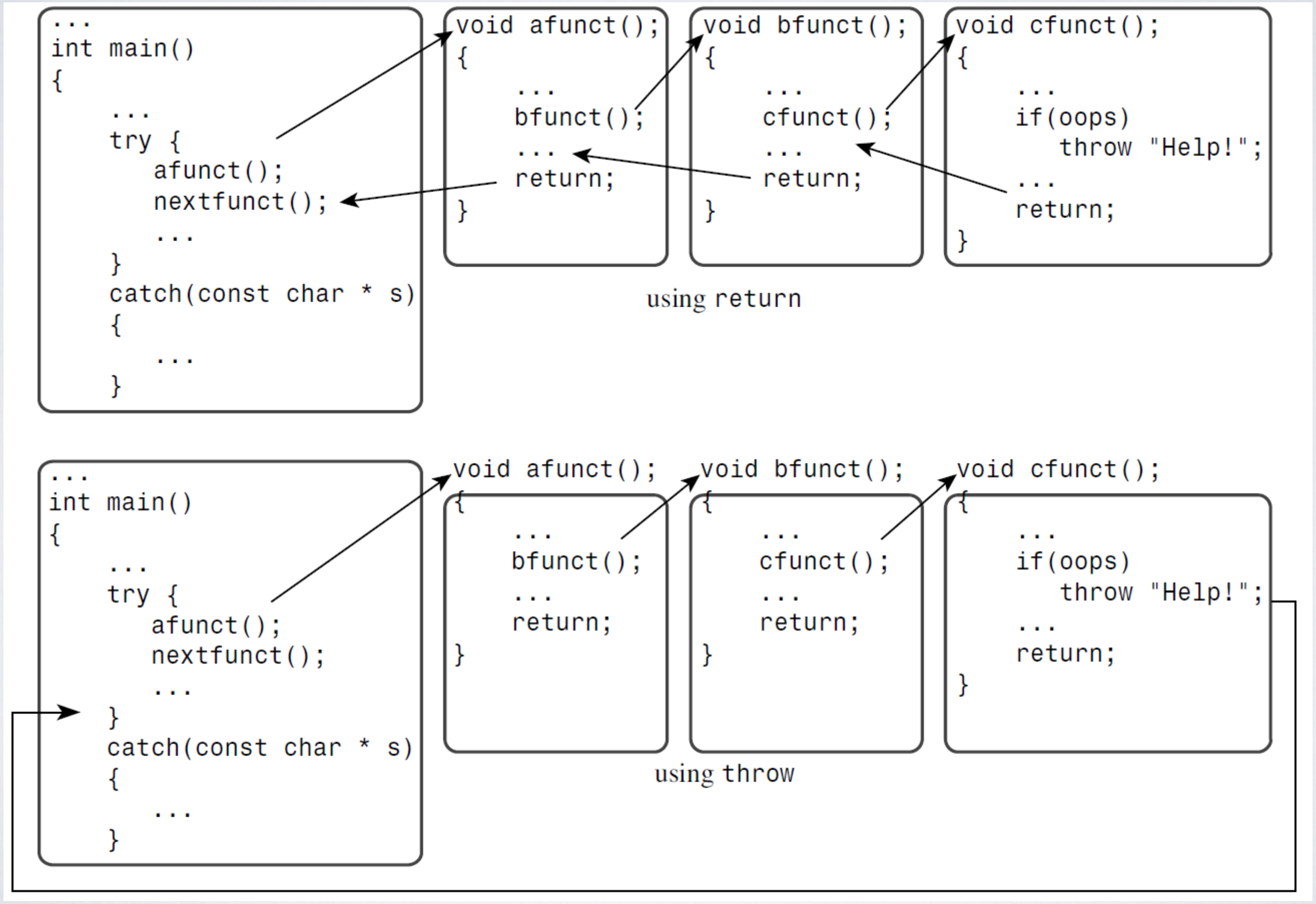
```
// Deprecated since C++11.
// Removed since C++17: compiler issues warning C5040.
double bad_gmean(double a) throw(bad_gmean); // may throw bad_thing
                                             // exception

// Since C++17: is alias for noexcept(true)
double func(double a) throw(); // doesn't throw an exception
```

EXCEPTION SPECIFICATIONS C++11

```
// Since C++11
double func(double a) noexcept; // doesn't throw an exception
double func(double a) noexcept(true); // doesn't throw an exception
```

UNWINDING THE STACK



```
...
int main()
{
    ...
    try {
       afunct();
       nextfunct();
       ...
    }
    catch(const char * s)
    {
       ...
    }
```

```
void afunct();
{
    ...
    bfunct();
    ...
    return;
}
```

```
void bfunct();
{
    ...
    cfunct();
    ...
    return;
}
```

```
void cfunct();
{
    ...
    if(oops)
       throw "Help!";
    ...
    return;
}
```

using return

```
...
int main()
{
    ...
    try {
       afunct();
       nextfunct();
       ...
    }
    catch(const char * s)
    {
       ...
    }
```

```
void afunct();
{
    ...
    bfunct();
    ...
    return;
}
```

```
void bfunct();
{
    ...
    cfunct();
    ...
    return;
}
```

```
void cfunct();
{
    ...
    if(oops)
       throw "Help!";
    ...
    return;
}
```

using throw

КАК ЭТО РАБОТАЕТ?

- При генерации исключения с помощью **throw** начинается раскрутка стека.
- У всех объектов всех функций на стеке вызываются деструкторы.
- Раскрутка останавливается, если найден подходящий обработчик исключения.

UNWINDING THE STACK

```
class demo final
{
private:
    std::string word;
public:
    demo (const std::string & str): word(str){
        std::cout << "demo " << word << " created\n";
    }
    ~demo(){
        std::cout << "demo " << word << " destroyed\n";
    }
    void show() const{
        std::cout << "demo " << word << " lives!\n";
    }
};
```

UNWINDING THE STACK

```
double means(double a, double b)
{
    double am, hm, gm;
    demo d2("found in means()");
    am = (a + b) / 2.0;     // arithmetic mean
    try
    {
        hm = hmean(a,b);
        gm = gmean(a,b);
    }
    catch (const bad_hmean & bh) // start of catch block
    {
        bh.msg();
        std::cout << "Caught in means()\n";
        throw;              // rethrows the exception
    }
    d2.show();
    return (am + hm + gm) / 3.0;
}
```

UNWINDING THE STACK

```
int main()
{
    double x, y, z;

    demo d1("found in main()");

    while (std::cin >> x >> y)
    {
        try {
            z = means(x,y);
        }
        catch (const bad_hmean& bh)
        {
            bh.msg();
            std::cout << "Try again.\n";
        }
        catch (const bad_gmean& bg)
        {
            std::cout << bg.msg();
            break;
        }
    }

    d1.show();

    return 0;
}
```

Console:

```
demo found in main() created
6 12
???
```

UNWINDING THE STACK

```
int main()
{
    double x, y, z;

    demo d1("found in main()");

    while (std::cin >> x >> y)
    {
        try {
            z = means(x,y);
        }
        catch (const bad_hmean& bh)
        {
            bh.msg();
            std::cout << "Try again.\n";
        }
        catch (const bad_gmean& bg)
        {
            std::cout << bg.msg();
            break;
        }
    }

    d1.show();

    return 0;
}
```

Console:

```
demo found in main() created
6 12
demo found in means() created
demo found in means() lives!
demo found in means() destroyed
6 -6
???
```

UNWINDING THE STACK

```cpp
int main()
{
    double x, y, z;

    demo d1("found in main()");

    while (std::cin >> x >> y)
    {
        try {
            z = means(x,y);
        }
        catch (const bad_hmean& bh)
        {
            bh.msg();
            std::cout << "Try again.\n";
        }
        catch (const bad_gmean& bg)
        {
            std::cout << bg.msg();
            break;
        }
    }

    d1.show();

    return 0;
}
```

Console:

```
demo found in main() created
6 12
demo found in means() created
demo found in means() lives!
demo found in means() destroyed
6 -6
demo found in means() created
hmean (6, -6) : invalid arguments: a = -b
Caught in means()
demo found in means() destroyed
hmean (6, -6) : invalid arguments: a = -b
Try again.
6 -8
???
```

UNWINDING THE STACK

```cpp
int main()
{
    double x, y, z;

    demo d1("found in main()");

    while (std::cin >> x >> y)
    {
        try {
            z = means(x,y);
        }
        catch (const bad_hmean& bh)
        {
            bh.msg();
            std::cout << "Try again.\n";
        }
        catch (const bad_gmean& bg)
        {
            std::cout << bg.msg();
            break;
        }
    }

    d1.show();

    return 0;
}
```

Console:

```
demo found in main() created
6 12
demo found in means() created
demo found in means() lives!
demo found in means() destroyed
6 -6
demo found in means() created
hmean (6, -6) : invalid arguments: a = -b
Caught in means()
demo found in means() destroyed
hmean (6, -6) : invalid arguments: a = -b
Try again.
6 -8
demo found in means() created
demo found in means() destroyed
gmean() arguments should be >= 0
demo found in main() lives!
demo found in main() destroyed
```

```cpp
class A {
public:
    ~A() { std::cout << "~A()" << std::endl; }
};

class B {
public:
    ~B() { std::cout << "~B()" << std::endl; }
};

void cc() {
    throw 123;
}

void bb() {
    B b;
    cc();
}

void aa() {
    A a;
    bb();
}

int main() {
    try {
        aa();
    }
    catch (int v) {
        std::cout << "Caught " << v << std::endl;
    }
}
```

```
~B()
~A()
Caught 123
```

MORE EXCEPTION FEATURES

```
class problem final {...}

void func(){
   ...
   if(oh_no){
      problem oops;
      throw oops;
   }
   ...
}
...

try{
     func();
}
catch(problem& ex){
   ...
}
```

Different objects.

```cpp
#include <iostream>
#include <string>
#include <cctype>
#include <exception>

using namespace std;

class INNException           : public std::exception {};
class WrongLengthException    : public INNException {};
class WrongCharsException     : public INNException {};
class WrongChecksumException : public INNException {};

unsigned long long int parseINN(const string &s) {
    if (s.length() != 10)
        throw WrongLengthException();

    if (!all_of(s.begin(), s.end(), ::isdigit))
        throw WrongCharsException();

    static int coeffs[] = { 2, 4, 10, 3, 5, 9, 4, 6, 8 };

    int res = 0;
    for (int i = 0; i < 9; ++i)
        res += (int(s[i]) - int('0')) * coeffs[i];

    if (int(s[9]) - int('0') != (res % 11) % 10)
        throw WrongChecksumException();

    return stoull(s);
}
```

INN validation

```
int main() {
    string s;
    if (getline(cin, s)) {
        try {
            auto inn = parseINN(s); // например, 5445264092

            cout << "Хороший, годный ИНН: " << inn << endl;
        }
        catch (WrongLengthException &) {
            cerr << "ИНН имеет неверную длину!" << endl;
        }
        catch (WrongCharsException &) {
            cerr << "ИНН содержит недопустимые символы!" << endl;
        }
        catch (WrongChecksumException &) {
            cerr << "У ИНН неверная контрольная сумма!" << endl;
        }
        catch (INNException &) {
            cerr << "ИНН никуда не годится!" << endl;
        }
    }
}
```

Exception Handling

```cpp
// ...
using namespace std;

class INNException             : public std::exception {};

class WrongLengthException    : public INNException {
public:
    const char *what() const override noexcept { return "неверная длина"; }
};

class WrongCharsException     : public INNException {
public:
    const char *what() const override noexcept { return "неверные символы"; }
};

class WrongChecksumException : public INNException {
public:
    const char *what() const override noexcept { return "неверная контр. сумма"; }
};

unsigned long long int parseINN(const string &s);

int main() {
    string s;
    if (getline(cin, s)) {
        try {
            auto inn = parseINN(s); // например, 5445264092

            cout << "Хороший, годный ИНН: " << inn << endl;
        }
        catch (INNException &exc) {
            cerr << "Ошибочный ИНН (" << exc.what() << ")!" << endl;
        }
    }
}
```

Other way

MORE EXCEPTION FEATURES

```
void process_file(const char *fn) {
    FILE *fp = fopen(fn, "r");

    try {
        // file processing
    }
    catch (...) {
        fclose(fp);
        throw;
    }

    fclose(fp);
}
```

← Catch any type exception

RESOURCE ACQUISITION IS INITIALIZATION RAII

```
class FilePtr {
    FILE *fp;
public:
    FilePtr(const char *fn, const char *access) {
        fp = fopen(fn, access);
    }
    FilePtr(FILE *fp) { this->fp = fp; }
    ~FilePtr()        { if (fp) fclose(fp); }

    operator FILE *() { return fp; }
};

void process_file(const char *fn) {
    // Принцип RAII: получение ресурса есть инициализация
    // техника управления ресурсами через локальные объекты
    FilePtr f(fn, "r");

    // …просто используем f
    // файл в любом случае будет закрыт автоматически
}
```

EXCEPTIONS IN CONTRUCTORS

- Деструктор вызывается только для полностью сконструированного объекта!
- Если в конструкторе было исключение, то будет утечка памяти:

Bad code!

```
class Y {
    int *p;
    void init() { /* здесь бабах! */ }

public:
    Y(int s) { p = new int[s]; init(); }
    ~Y() { delete p; }
};
```

EXCEPTIONS IN CONTRUCTORS

Good code!

```
class Z {
    std::vector<int> p;
    void init() { /* здесь бабах! */ }

public:
    Z(int s): p(s) { init(); }
    ~Z() = default;
};
```

if "init" throws an exception, the destructor of "p" will be called.

BEST PRACTICES

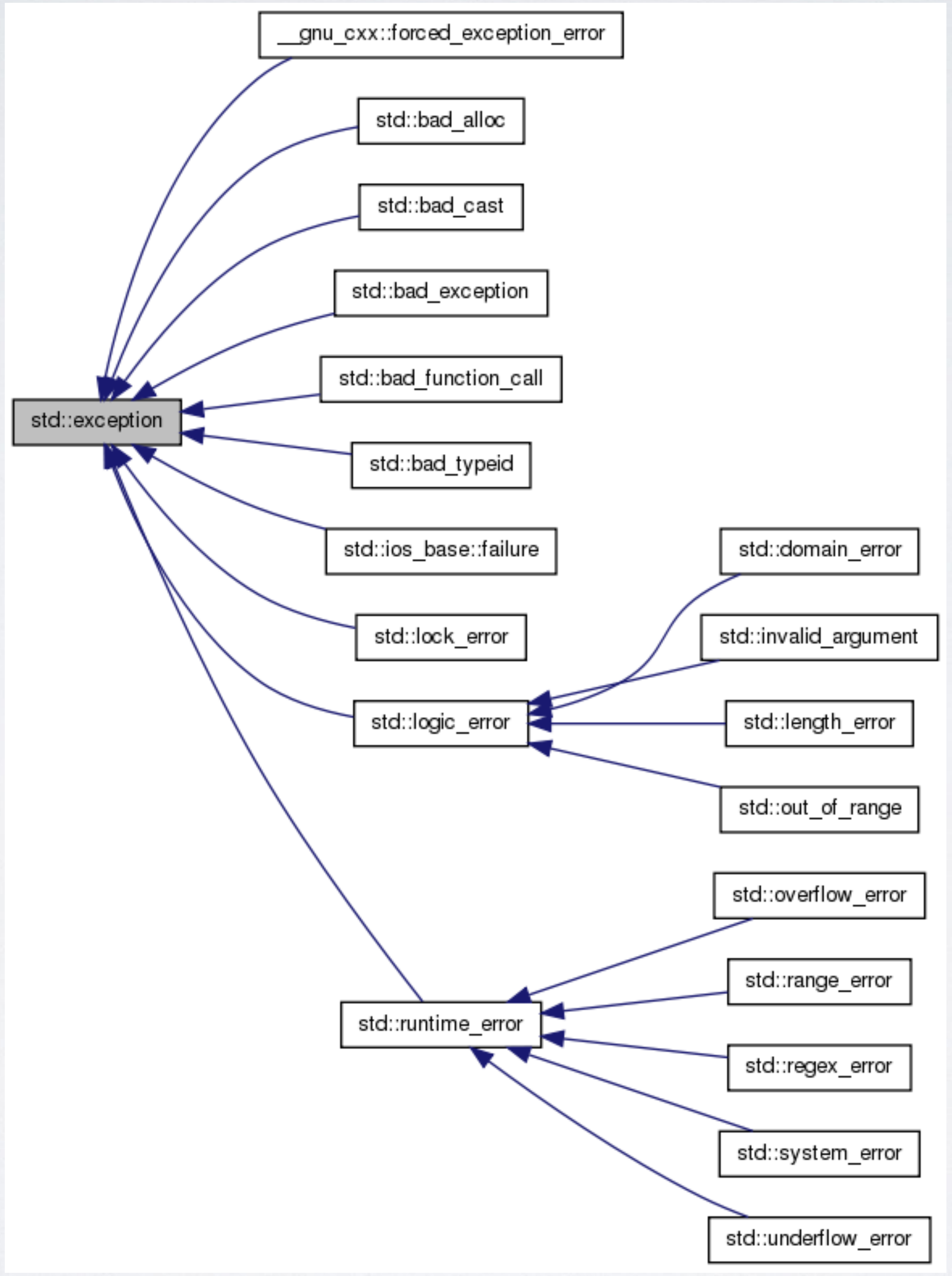
- Использовать иерархию классов исключений: не кидать **int**,
  const char * и т.п.

- Делать классы исключений простыми.

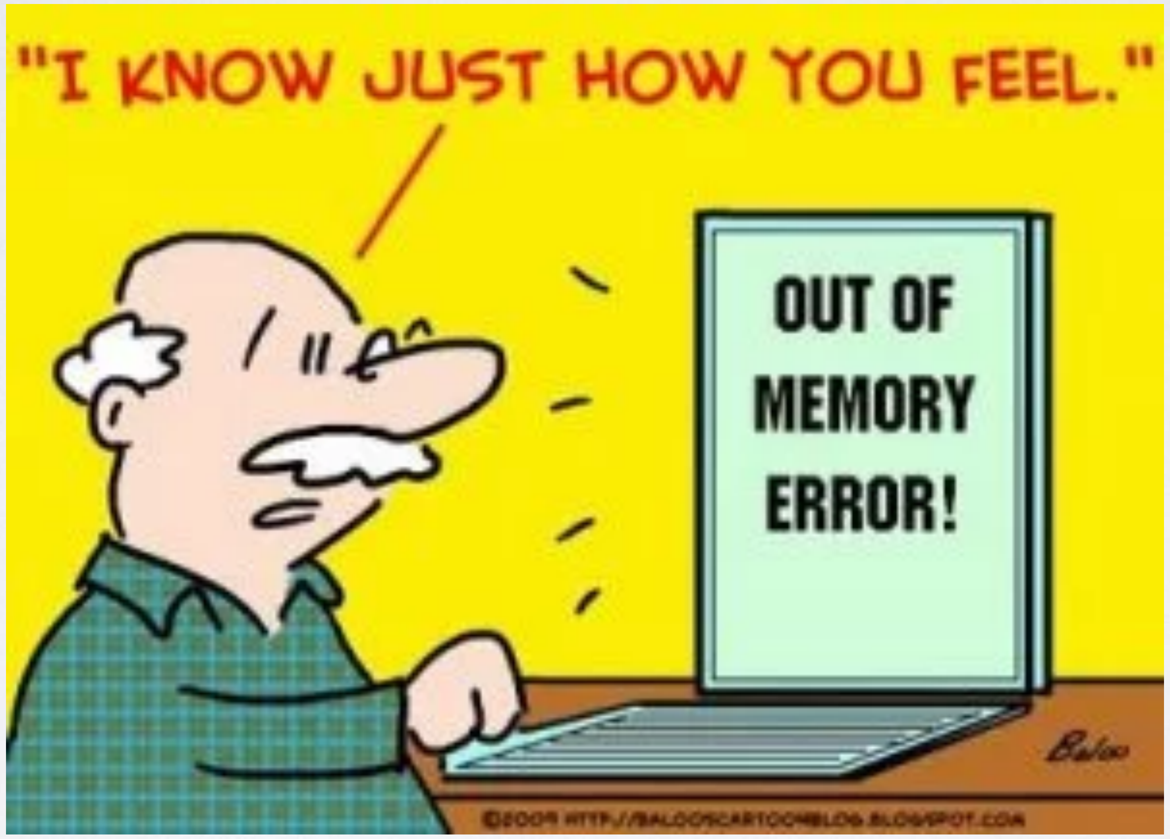
- Использовать исключения только для *исключительных* ситуаций.

- Генерируя исключение, понимать где и кем оно будет обработано.

STD EXCEPTIONS



КОНЕЦ ВОСЬМОЙ ЛЕКЦИИ


"I KNOW JUST HOW YOU FEEL."
OUT OF MEMORY ERROR!
Baloo
©2009 HTTP://BALOOSCARTOONBLOG.BLOGSPOT.COM

```
throw EndOfLecture();
```