

WYDZIAŁ FIZYKI
UNIWERSYTET WARSZAWSKI

Programowanie zaawansowane FM i NI

Wykład 13

Obsługa wyjątków

Krzysztof Piasecki

Semestr letni roku akad. 2023-24


krzysztof.piasecki@fuw.edu.pl

- **Wyjątki.** Obserwowaliśmy w kodach sytuacje niepożądane.
  - ▷ np. gdy `vector::at(..)` sięgał po element poza zakresem
  - ▷ np. gdy prosimy o $\sqrt{}$ z liczby, ale liczba jest < 0.

  Zwykle skutkiem było przerwanie kodu.

Wyjątki pozwalają na kontrolowane obsłużenie błędów. Kod się nie wywraca: to my określamy, co ma zrobić.

⊙ Obejmujemy dany obszar nasłuchem (otaczamy go przez **`try {…}`**).
W środku, komendą **`throw`** możemy przerwać blok i wyrzucić „*wyjątek*” (*exception*).
Po bloku **`try {…}`**, polecenie **`catch (…) {…}`** chwyta wyjątek i go obsługujemy.

⊙ **`throw`** wyrzuca zmienną/obiekt, która zawsze ma jakiś typ.

`catch` w **`(..)`** wystawia zmienną danego typu.
Tak chwytamy każdy wyrzucony obiekt tego typu.
Po **`catch(..)`**, w **`{..}`** piszemy blok kodu, na wypadek schwytaniu wyjątku.

⊙ W tym przykładzie, **`throw`** uprzedza próbę dzielenia przez 0, wyrzucając string z napisem.
`catch` to wychwytuje i wypisuje komunikat.
Sytuacja niechciana jest pod naszą kontrolą.

[Link]

```
5  int main () {
6    double x, y;
7    cout << "Podaj 2 składniki dzielenia: ";
8    try {
9      cin >> x >> y;
10     if (y == 0.)
11       throw string ("[Proba dzielenia p/0]");
12     else
13       cout << "x/y = " << x/y << endl;
14   }
15   catch (string& s) {
16     cerr << "<Blad> " << s << endl;
17     return 1;
18   }
19   cout << "[Main: end]";
20 }
```

- **Wyjątki – typy.** Z obszaru nasłuchiwanego przez `try` można wyemitować wyjątki różnych typów.

  Dlatego można wpisać wiele × `catch`, każdy łowiący wyjątki jednego typu.

⊙ Wystawiając w `catch ()` tylko typ, nie dysponujemy obiektem wyjątku.

⊙ Gdy wystawimy:

`catch (typ obiekt)` ,

to wyjątek przejdzie do obiektu i można go używać (referencja podepnie oryginał).

⊙ Wystawiając:

`catch (...)` ,

schwytamy wszystkie pozostałe typy wyjątków.

⊙ Gdy dla wyrzuconego typu – nie będzie adekwatnego typu w `catch`, to program się przerwie.

⊙ Ale nie może być `try` nie zakończone żadnym `catch`.

[Link]

```
 5  int main () {
 6    cout << "Wpisz liczbe 1-4 lub inna: ";
 7    try {
 8      int i;
 9      cin >> i;
10      if      (i == 1) throw i ;
11      else if (i == 2) throw 'a' ;
12      else if (i == 3) throw string ("abc");
13      else if (i == 4) throw -1.234 ;
14      else
15        cout << "[blok try: ok]\n";
16    }
17    catch (int i)    { cerr << "[int] " << i << endl; }
18    catch (char )    { cerr << "[any char] " << endl; }
19    catch (string s) { cerr << "[str] " << s << endl; }
20    catch (...)      { cerr << "[any other exception]\n"; }
21
22    cout << "[Main: end]";
23  }
```

Wyjątki w funkcjach

[Link]

- W funkcji można poprzez `throw` wyrzucić wyjątek.
  To rodzaj pobocznego `return` :
  funkcja się tu zatrzyma i zwrócimy obiekt.

- Ale jeżeli nie obejmiemy miejsca wywołania funkcji przez

```
try {
  …
} catch () {…}
```

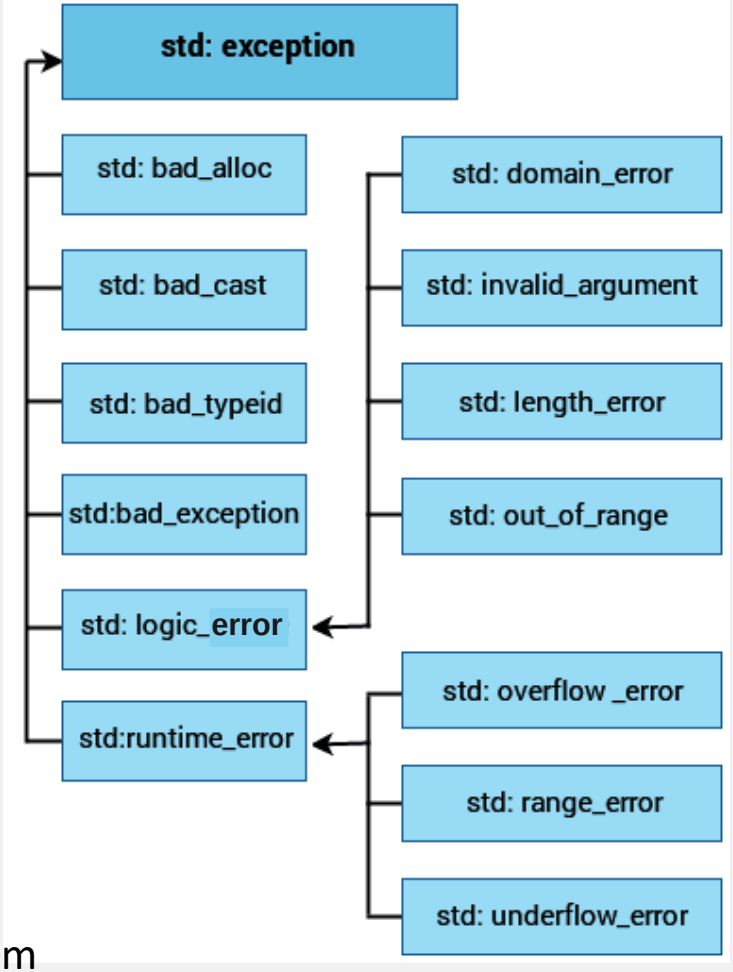
  (⊕ `catch` ma zawierać wyrzucony typ)
  → to program się przerwie.

- Jeśli to miejsce obejmiemy blokiem `try/catch`
  ⊕ jest `catch` z pasującym typem,
  to możemy przejąć wyrzucony obiekt
  i obsłużyć sytuację.

```
5  void myFunction () {
6    cout << "[myFunction]\n";
7    int i = -1 ;
8    if (i < 0)
9      throw string ("[negative i]");
10
11   cout << "[myFunction - end]\n";
12 }
13
14 int main () {
15   try {
16     myFunction ();
17   }
18   catch (string& s) {
19     cerr << "Caught: "  << s << endl ;
20   }
21
22   cout << "[main - end]\n";
23 }
```

Wyjątki klasy exception i dziedziczących

- W C++ gotowa jest klasa bazowa **`exception`** i klasy dziedziczące. Nazwy tych klas specjalnie wskazują na rodzaj problemu. Dzięki temu, gdy jako wyjątek wyrzucamy obiekt o danej nazwie, to w kodzie widać kategorię problemu.
  - Biblioteka **`<exception>`** ma tylko klasę `exception`.
  - Biblioteka **`<stdexcept>`** ma wszystkie klasy dziedziczące i załącza <exception> .
    - ⇒ Dlatego zwykle załączamy tylko **`<stdexcept>`**.
- Obiekty `exception` (i dziedziczące) mają pole komunikatu. Pisząc np. **`throw logic_error ("Hey, this was wrong") ;`** utworzymy obiekt z takim komunikatem. Metoda **`what()`** zwraca treść tego komunikatu.
- Przykładowo:



`out_of_range`	wyrzuci metoda `at`, gdyby indeks był poza zakresem
`logic_error`	pasuje, gdy jeszcze na etapie kodu coś jest źle
`runtime_error`	pasuje, gdy błąd pojawi się dopiero, gdy działa kod
`domain_error`	pasuje, gdy funkcja matematyczna ma dostać argument poza dziedziną
`range_error`	pasuje przy próbie wstawienia do zmiennej – wartości poza zakresem

Wyjątki w bibliotece <stdexcept>

[Link]

- Przykład:
  gdy metodę `vector<..> :: at`
  poprosimy o element poza zakresem,
  to wyrzuci ona wyjątek klasy
  `out_of_range`.

- W środku `catch()` możemy wystawić
  (referencję na) obiekt typu `out_of_range`.

  Można też skorzystać z polimorfizmu
  i dać referencję na obiekt typu `exception`.

- Klasa `out_of_range`
  dziedziczy po klasie `logic_failure`,
  a ta z kolei – po klasie `exception`.

  Zatem dziedziczy metodę **`what()`**
  która zwraca treść komunikatu.

```
 6  int main () {
 7    vector<double> tab(10);
 8    try {
 9      int i = 20;
10
11  //  if (i < 0 || i >= tab.size())
12  //    throw out_of_range ("Przesadziles.\n");
13
14      tab.at(i) = 100;
15    }
16    catch (const out_of_range& myException ) {
17      cerr << "[Wyjatek wyrzucony przez vector] "
18           << myException.what () << endl;
19    }
20    cout << "[Main: end]" ;
21  }
```

Wyjątki w bibliotece <stdexcept>

- Przykład: gdy operator alokacji dynamicznej `new` poprosimy o zaalokowanie obszaru większego, niż pozwala system operacyjny, to `new` wyrzuci wyjątek klasy **`bad_alloc`**.

- W środku `catch()` możemy wystawić (referencję na) obiekt typu `bad_alloc`.
  Albo – korzystając z polimorfizmu – referencję na obiekt typu `exception`.

- Klasa `bad_alloc` dziedziczy po klasie `exception`.
  Zatem dziedziczy metodę `what()`, która zwraca treść komunikatu.

[Link]

```
5  int main () {
6    int* tab;
7    try {
8      for (long int size = 1;  ; size += 2000*1024 ) {
9        tab = new int[size];
10       cout << " Tablica o rozmiarze "
11            << size/1000/1024 << " MB : " << tab << endl;
12       delete[] tab;
13     }
14   }
15   catch (const bad_alloc& myException) {
16     cerr << "[Schwytany bad_alloc o tresci: "
17          << myException.what() << endl;
18   }
19 }
```

Wyjątki w konstruktorach klas

⊙ Zasadniczo nie ma problemu, aby konstruktor klasy wyrzucał wyjątek. Również, w liście inicjalizacyjnej przypisywany obiekt może go wyrzucić.

Jednak miejsce deklaracji obiektu trzeba otoczyć blokiem `try/catch` .

⊙ I tu pojawia się problem, bo w C/C++ każdy obiekt objęty { … } istnieje tylko w tych nawiasach.

⇒ Proste obejście tego ograniczenia: **alokacja dynamiczna**.

Tworzymy wskaźnik na obiekt, a w bloku `try/catch` alokujemy obiekt poprzez `new`. Dzięki temu, po opuszczeniu `try`, obiekt jest dostępny pod wskaźnikiem.

[Link]

```
5  struct MyClass {
6   double x;
7   int* ptr;
8   MyClass () : ptr ( new int[9999999999] )  { }
9
10  MyClass (double _x) {
11   if (_x < 0)
12    throw runtime_error ("<E-MyClass> x<0." );
13   else
14    x = _x ;
15   }
16  };
17
18  int main () {
19   MyClass* M1, * M2;
20   try {
21    M1 = new MyClass ( -1. );
22   } catch (runtime_error& re) {
23    cerr << "[M(int) error] " << re.what() << endl; }
24
25   try {
26    M2 = new MyClass ;
27   } catch (bad_alloc& ba) {
28    cerr << "[M      error] " << ba.what() << endl; }
29  }
```

Przepływ wyjątku przy dziedziczeniu

- Przypuśćmy, że klasa **B** dziedziczy po **A**. Niech konstruktor klasy bazowej wyrzuca wyjątek.

  My chcemy utworzyć obiekt klasy B, który w konstruktorze, dzięki liście inicjalizacyjnej, wywoła konstruktor klasy bazowej.

  W trybie domyślnym, wyjątek z „głębszej" klasy A przepłynie automatycznie do miejsca deklaracji obiektu klasy B.

- Ale jeśli chcemy, to listę inicjalizacyjną konstruktora B możemy objąć nasłuchem `try/catch` (por. przykład). Wtedy przechwytujemy wyjątek jeszcze na poziomie konstruktora B. Tam obsługujemy sytuację – i posyłamy (lub nie) ten sam (lub inny) wyjątek.

[Link]

```
5  struct A {
6   int x;
7   A (int _x) {
8     if (_x < 0) throw runtime_error ("[A problem: x<0]");
9     else        x = _x ;
10  }
11 };
12
13 struct B : A {
14  int y;
15  B (int _x, int _y) try : A (_x) , y(_y) {
16  } catch (runtime_error& RE) {
17     string Msg = "[Via B]" + string(RE.what()) ;
18     throw runtime_error ( Msg ) ;
19  }
20 };
21
22 int main () {
23   try {
24     B objB ( -1. , 1.);
25   } catch (runtime_error& RE) {
26     cerr << "[B caught] " << RE.what() << endl;
27   }
28 }
```