Wyjątki w C++

Wyjątki (exceptions)

- sygnalizują problem powstały w czasie wykonywania programu
- występują rzadko

Obsługa wyjątków

- może rozwiązać problem
  - pozwala kontynuować wykonanie programu lub
  - powiadamia użytkownika o problemie i
  - zamyka aplikację w kontrolowany sposób
- Sprawia, że programy wytrzymałe i odporne na uszkodzenia

Obsługa wyjątków w C++

Standardowy mechanizm do przetwarzania błędów

- Szczególnie ważne podczas pracy nad projektem z dużym zespołem programistów

Obsługa wyjątków w C++ jest podobna do obsługi wyjątków w Java

Obsługa wyjątków w Java jest podobna do obsługi wyjątków w C++

Podstawowe informacje

Mechanizm wysyłania sygnału wyjątku na stos wywołań

- Niezależnie od wywołań przerywających

Uwaga: mechanizm działa na tej samej zasadzie co w języku C

- **setjmp(), longjmp()**
- patrz manual do C

Tradycyjna obsługa wyjątków

Przemieszanie programu i logiki obsługi błędów

Perform a task
If the preceding task did not execute correctly
  Perform error processing
Perform next task
If the preceding task did not execute correctly
  Perform error processing
…

Sprawia, że programy są trudne do odczytania, modyfikowania, utrzymania i debugowania

Wpływa na wydajność

Podstawowe założenia

Usunięcie kodu obsługi błędów z "głównej linii" w trakcie realizacji programu

Programiści mogą obsługiwać wszelkie wyjątki w zależności ich wyboru:

- Wszystkie wyjątki
- Wszystkie wyjątki określonego typu
- Wszystkie wyjątki od grupy powiązanych typów

Program powinien:

- odzyskać działanie po błędzie
- ukrywać błędy
- przekazywać informację o błędach w górę łańcucha poleceń
- ignorować pewne wyjątki i pozwolić aby ktoś inny się nimi zajął

Istnienie klasa wyjątków - *exception*

- Zazwyczaj pochodzi z jednej z klas bazowych obsługujących wyjątki systemowe

- W przypadku wystąpienia wyjątku lub błędu program generuje obiekt tej klasy
- Obiekt jest umieszczany na górze stosu

Program wywołujący może wybrać czy przechwycić wyjątki określonych klas

- Akcja podejmowana w zależności od klasy obiektu generującego wyjątek

Klasa exception

Standardowa klasa bazowa języka C++ dla wszystkich wyjątków

Zapewnia klasom pochodnym wirtualną funkcję what

— Zwraca zapisany komunikat o błędzie

Przykład: dzielenie przez zero

```
// definicja klasy plik zero.h
#include <stdexcept> // stdexcept plik nagowkowy zawierajacy runtime_error
using std::runtime_error;
// obiekty DzieleniePrzezZero powinny być wyrzucane
// przez funkcje po wykryciu dzielenia przez zero
class DzieleniePrzezZeroException : public runtime_error
{
public:
// konstruktor specyfikuje domyslna wiadaomosc
DzieleniePrzezZeroException::DzieleniePrzezZeroException()
: runtime_error( „Dzielenie przez zero jest niemożliwe!" ) {}
}; // koniec klasy
```

Przykład: dzielenie przez zero

```
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
#include „zero.h" // klasa DzieleniePrzezZeroException
double dzielenie( int licznik, int mianownik )
{
//wygeneruj wyjątek DzieleniePrzezZeroException przy dzieleniu przez zero
  if ( mianownik == 0 )
     throw DzieleniePrzezZeroException(); // zakończ funkcję
  //w przeciwnym wypadku zwroc rezultat
```

```
  return static_cast<double>( licznik ) / mianownik;
}
int main()
{
  int liczba1; // licznik
  int liczba2; // mianownik
  double wynik;
  cout << "Podaj dwie liczby całkowite (EOF aby skończyć): ";
```

Przykład: dzielenie przez zero

```
  while ( cin >> liczba1 >> liczba2 )
  {
    // uzycie bloku try do przechwycenia wyjatku
    try
    {
      wynik = dzielenie( liczba1, liczba2 );
      cout << „Wynik dzielenia to: " << wynik << endl;
    }
    // obsluga zdarzenia zwiazanego z dzieleniem przez zero
    catch ( DzieleniePrzezZeroException &dzieleniePrzezZeroException )
    {
      cout << „Wystąpił błąd: "
        << dzieleniePrzezZeroException.what() << endl;
    }
    cout << "\nPodaj dwie liczby całkowite (end-of-file to end): ";
  } // koniec while
  cout << endl;
  return 0; //poprawne wyjscie
} // koniec main
```

Blok try

Słowo kluczowe try po którym następują nawiasy klamrowe {}

Blok powinien zawierać

- Polecenia, które mogą spowodować wyjątki
- Polecenia, które powinny zostać pominięte w przypadku wystąpienia wyjątku

Inżynieria oprogramowania a wyjątki

Wyjątki mogą wpływać na działanie programu:

- poprzez wyraźnie wymieniony kod w bloku **try**,
- poprzez wywołania innych funkcji,
- przez głęboko zagnieżdżone wywołania funkcji inicjowanych przez kod w bloku **try.**

Obsługa Catch

Powinna się zawsze znaleźć po bloku try

- Jeden lub więcej programów obsługi **catch** dla każdego bloku **try**

Słowo kluczowe catch

Parametr wyjątku umieszczony w nawiasach

- reprezentuje typ wyrażenia do przetworzenia
- może zapewnić opcjonalny parametr do interakcji z przechwyconym wyjątkiem obiektu

Wykonuje się wtedy gdy typ parametru wyjątku zgadza się z tym który wystąpił w bloku try

- może być to klasa bazowa dla klasy generującej wyjątek

Obsługa catch

```
try {
        // kod do bloku try
}
catch (exceptionClass1 &name1) {
        // obsługa wyjątków klasy exceptionClass1
}
catch (exceptionClass2 &name2) {
        // obsługa wyjątków klasy exceptionClass2
}
catch (exceptionClass3 &name3) {
        // obsługa wyjątków klasy exceptionClass3
}
...
// kod wykonywany gdy nie zaszedł wyjątek lub
// zaszedł wyjątek w kodzie catch
```

Obsługa catch c.d.

```
void funkcja()
{
 try
  {
   // obliczenia
   throw ‘x’;
   // dalsze obliczenia
   throw -1;
  }
 catch (int i)
  {
   cout << "wykryto błąd o numerze " << i;
  }
 catch (char)
  {
   cout << "wykryto błąd typu char";
  }
}
```

Typowe błędy programistyczne

Błąd składni, aby umieścić kod między blokiem try i jego odpowiednikiem catch

Każda obsługa wyjątku catch powinna posiadać pojedynczy parametr

- Określanie listy parametrów wyjątków oddzielonych przecinkami jest błędem składni

Błąd logiczny, aby obsłużyć ten sam typ wyjątków w dwóch różnych blokach catch przy jednym bloku try

Podstawowe założenia c.d.

Model wygaśnięcia obsługi wyjątków

- blok **try** *wygasa* kiedy pojawia się wyjątek
  - Zmienne lokalne w bloku try wychodzą poza zakres
- Wykonywany jest kod w odpowiadającej sekcji **catch**
- Kontrola jest wznawiana przez pierwsze polecenie po ostatnim bloku **catch** poprzedzanym przez blok **try**

Odwijanie stosu

- Zachodzi gdy nie znaleziono pasującego bloku **catch**
- Program stara się zlokalizować inny przykryty blok **try** w funkcji wywołującej

Odwijanie stosu

Zachodzi wtedy gdy rzucony wyjątek nie jest złapany w odpowiednim zakresie

Odwijanie funkcji kończy jej działanie:

- Wszystkie lokalne zmienne i funkcje są niszczone
  - Wywołanie destruktorów
- Kontrola programu zostaje zwrócona do punktu gdzie została wywołana funkcja

Próby przechwycenia wyjątku są podejmowane w zewnętrznych blokach try…catch

Jeżeli wyjątek nie zostaje przechwycony wtedy wywoływana jest funkcja terminate

Odwijanie stosu

Wędrówka wyjątku

Uwagi

Z obsługą wyjątków, program może kontynuować wykonywanie (a nie kończyć działanie) po obsłużeniu problemu

Przyczynia się to do wspierania stabilnych aplikacji, które są aplikacjami typu mission-critical lub business-critical

Gdy nie pojawia się wyjątek, nie ma wtedy utraty wydajności

Uwagi

Instrukcje throw a return:

- **return** przerwanie zawsze tylko jednej funkcji i powrót do miejsca, z którego ją wywołano.

- **throw** może nie przerywać wykonywania funkcji (jeżeli znajdzie w niej pasującą instrukcję **catch**), równie dobrze może przerwać działanie wielu funkcji lub całego programu
- **return** umożliwia rzucenie obiektu należącego tylko do jednego, ściśle określonego typu.
- **throw** może wyrzucać obiekt dowolnego typu, zależnie od potrzeb
- **return** - normalny sposób powrotu z funkcji
- **throw** - używany w sytuacjach wyjątkowych

Rzucanie wyjątku

Należy użyć słowa throw następnie argumentu reprezentującą typ wyjątku

- Argument **throw** może być dowolnego typu
- Jeżeli argument **throw** jest obiektem, nazywa się wtedy wyjątkiem obiektowym

Argument throw inicjalizuje parametr wyjątku w odpowiadającym bloku catch, jeżeli blok ten został odnaleziony

Uwagi

Łapanie wyjątku obiektu przez referencje eliminuje narzut kopiowania obiektu reprezentującego rzucony wyjątek

Przypisanie każdemu rodzajowi błędu wykonawczego odpowiedniej nazwany wyjątku obiektu poprawia przejrzystość programu

Kiedy używać obsługi wyjątków

Nie używać do rutynowych czynności typu EOF czy sprawdzania null string

Do przetwarzania błędów synchronicznych

- Występują, gdy dana instrukcja jest wykonywana

Nie do przetwarzania błędów asynchronicznych

- Występują zarazem równolegle jak i niezależnie od wykonywania programu

Aby przetwarzać problemy powstałe w określonych elementach oprogramowania:

- takich jak predefiniowane funkcje i klasy,
- obsługa błędów może być przeprowadzona przez kod programu i być dostosowana w zależności od potrzeb

Inżynieria oprogramowania – uwagi

Dobrze jest włączyć strategii obsługi wyjątków w projektowaniu systemu od początku

- Bardzo trudno to zmienić gdy system został wdrożony!

Obsługa wyjątków zapewnia jednolitą technikę przetwarzania problemów

Należy unikać korzystania z obsługi wyjątków jako alternatywnej formy sterowania przepływem

- Te "dodatkowe" wyjątki mogą powodować zakłócenia natywnej obsługi błędów

Odrzucenie wyjątku

Po złapaniu wyjątku przez bardziej wewnętrzny catch nie potrafimy podjąć wszystkich akcji, jakie byłyby dla niego konieczne.

- Np. można zarejestrować wyjątek w dzienniku błędów -> bardziej użyteczna reakcja powinna być gdzieś wyżej

Należy użyć instrukcji throw bez parametrów

Przydaje się, gdy obsługa catch nie może lub może tylko częściowo przetworzyć wyjątek

Następny blok try próbuje dopasować wyjątek z jednego z jego bloków obsługi catch

Popularny błąd programistyczny

Wykonywanie pustej instrukcji **throw** poza obsługą bloku **catch** powoduje wywołanie funkcji kończącej (terminate)

- Porzuca przetwarzania wyjątku i natychmiast kończy działanie programu

Specyfikacja typów wyjątków
Nazywana często listą **throw**
Polecenie **throw**

- oddzielona przecinkami lista klas wyjątków w nawiasach

Przykład

```
int jakasFunkcja( double wartosc )
  throw ( WyjatekA, WyjatekB,
        WyjatekC )
{
  ...
}
```

- oznacza, że **jakasFunkcja** może rzucać (**throw)** typy **WyjatekA**, **WyjatekB** i **WyjatekC**

Specyfikacja wyjątków

Funkcja może rzucać **throw** tylko wyjątkami typów ze specyfikacji (lub typów pochodnych)

- Jeśli funkcja zgłasza wyjątek bez specyfikacji, wtedy wywoływana jest funkcja **unexpected**
  - Działanie to kończy funkcjonowanie aplikacji

Brak specyfikacji typu wyjątków wskazuje, że funkcja może rzucić wyjątek dowolnego typu
Pusta lista komendy **throw()**, zakłada że funkcja nie może rzucić żadnego wyjątku

Błędy

Kompilator nie wygeneruje błędu kompilacji, jeśli funkcja zawiera wyrażenie **throw** dla wyjątku nie wymienionego w specyfikacji funkcji

Błąd pojawia się tylko wtedy, gdy funkcja ta próbuje rzucić ten wyjątek w czasie wykonania aplikacji
Aby uniknąć niespodzianek w czasie wykonywania, należy dokładnie sprawdzić kod, aby upewnić się, że funkcje nie generują wyjątków nie wymienione w ich specyfikacji wyjątków

Konstruktory i desktruktory

Wyjątki i kontruktory

- Wyjątki umożliwiają konstruktorom zgłaszanie błędów
  - Nie można zwrócić wartości
- Wyjątki rzucane przez konstruktory powodują to, że każdy już zbudowany obiekty wywołuje destruktor
  - Tylko te obiekty które istnieją zostają zniszczone

Wyjątki i destruktory

- Destruktory są wywoływane dla wszystkich automatycznych obiektów w terminowanym bloku **try**, gdy jest rzucony wyjątek
  - Nabyte zasoby mogą być umieszczone w lokalnych obiektach aby automatycznie zwalniać zasobów, gdy wystąpi wyjątek
- Jeśli destruktor wywołany został przez odwijanie stosu to zgłasza wyjątek i jest wywoływana funkcja **terminate**

Uwaga

Gdy jest wyjątek jest rzucony od konstruktora dla obiektu, który jest tworzony poleceniem **new** …

… dynamicznie zaalokowana pamięć dla tego obiektu jest zwalniana.

Wyjątki i dziedziczenie

Nowe klasy wyjątków można tworzyć przez dziedziczenie z istniejących klas wyjątków
Obsługa **catch** dla konkretnej klasy wyjątków może złapać wyjątki klas pochodzących z tej klasy

- Umożliwia przechwytywanie podobnych błędów w zwięzłej notacji

Niepowodzenie wywołań **new**
Niektóre kompilatory rzucają (**throw)** wyjątek **bad_alloc**

- Zgodność ze standardem specyfikacji C++

Niektóre zwracają wartość **0**

- Cześć kompilatorów C++ posiada również wersję **new** która zwraca **0**
  - Należy w tym przypadku użyć wyrażenia **new**(**nothrow**), gdzie **nothrow** jest typu **nothrow_t**

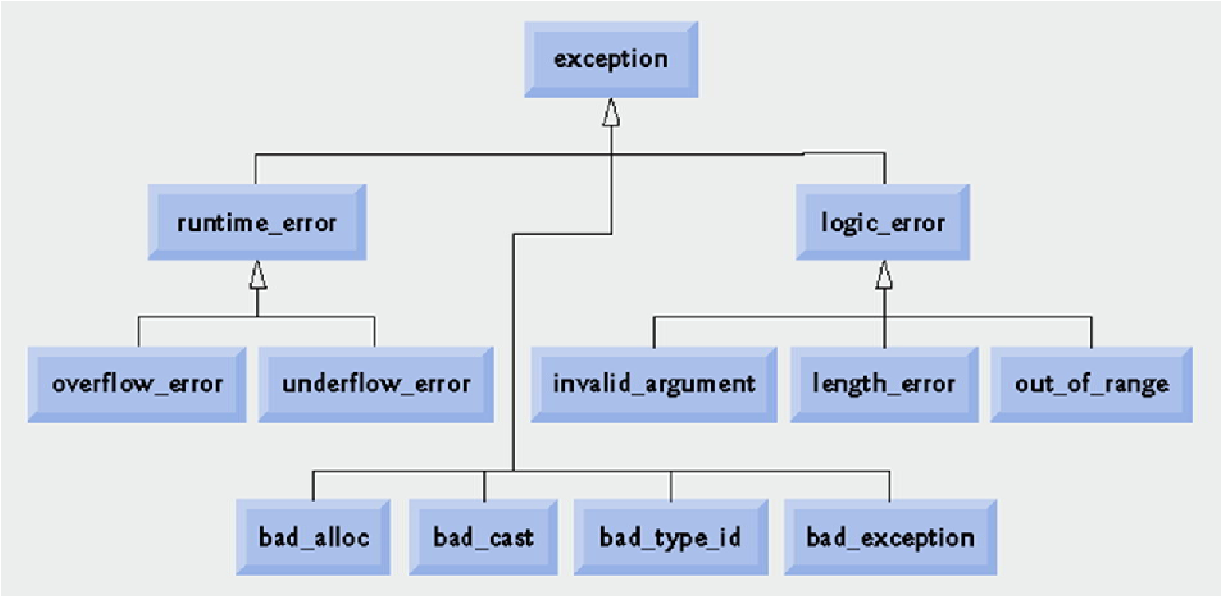
Cześć kompilatrów zwraca (**throw) bad_alloc** jeżeli **<new>** jest dołączone
Hierarchia standardowej biblioteki wyjątków

- Klasa bazowa **exception**
  - Zawiera wirtualną (**virtual)** funkcję **what** dla przechowywania komunikatów o błędach

Klasy wyjątków dziedziczące z **exception**

- **bad_alloc** – rzucone przez **new**
- **bad_cast** – rzucone przez **dynamic_cast**
- **bad_typeid** – rzucone przez **typeid**
- **bad_exception** – rzucone przez **unexpected**
  - Zamiast zakończenia programu lub wywołania funkcji określonej przez **set_unexpected**
  - Użyte tylko gdy **bad_exception** jest na liście funkcji **throw**

Standardowe klasy wyjątków



Wyjątki w bibliotece standardowej

- **bad_alloc**
  - wyrzuca **new** w przypadku niepowodzenia alokacji
- **bad_cast**
  - rzuca **dynamic_cast** gdy nie powiedzie się konwersja referencji
- **bad_exception**
  - niezgodność typów wyjątku z **catch**, lub do typów wyspecyfikowanych w nagłówku funkcji rzucającej (po wywołaniu funkcji unexpected())
- **bad_typeid**
  - rzuca **typeid** np. gdy otrzyma jako argument NULL
- **runtime_error**
  - klasa bazowa dla klas definiujących wyjątki czasu wykonania programu (**range_error, overflow_error, underflow_error**)
- **logic_error**
  - dla błędów wewnętrznej logiki programu, stanowi klasę bazową dla innych klas (**domain_error, invalid_argument, length_error, out_of_range**)
- **ios_base::failure**
  - błędy zgłaszane przez bibliotekę **iostream**

Podsumowanie

- Wyjątki pochodzą z klasy **exception**
- Wyjątek lub błąd jest sygnalizowany przez rzucenie obiektu tej klasy
  - Stworzony przez konstruktor w wyrażeniu **throw**
- Programy wywołujące mogą sprawdzać wyjątki z użyciem konstrukcji **try...catch**

- Uniwersalna metoda obsługi wyjątków
    - Znacznie lepszy od kodowania obsługi wyjątków w dalszej perspektywie
- Brak wpływu na wydajność jak nie zachodzą wyjątki