Powody wprowadzenia obsługi wyjątków

- Podczas wykonania programu mogą wystąpić przypadki, które nie zostały przewidziane przez programistę
  - Np. użytkownik wprowadził złe dane
  - Kontakt z urządzeniem zewnętrznym został przerwany
- W celu uniknięcia przerwania pracy programu konieczne jest zaimplementowanie obsługi błędów
  - Jedną z możliwości jest obsługa błędów poprzez znaczniki statusów (używane w C)
  - W C++ natomiast wprowadzono nowy znacznie ogólniejszy mechanizm pozwalające na obsługę wyjątków

Obsługa wyjątków

- Wyjątek nie zawsze oznacza błąd
  - Błąd jest niejako podzbiorem wyjątków
- Sytuacją wyjątkową (wyjątkiem) może być wszystko co my programiści za to uznamy
- Obsługa wyjątków stanowi nowy sposób obsługi błędów i sytuacji nazwijmy to niecodziennych
  - Należy stosować kiedy tylko jest to możliwe
- Stanowi wbudowaną własność języka
- Umożliwia obsługę wyjątków w każdym ich znaczeniu za pomocą mechanizmu niezależnego od zasadniczego przepływu sterowania w programie

Obsługa błędów poprzez znaczniki statusów (C)

- Odbywa się poprzez kontrolę wartości zwracanych przez funkcję i wywoływaniu procedur obsługujących błędy
- Błędy są wykrywane i obsługiwane przez kod programu
  - Nie ma różnicy między zwykłym przepływem sterowania programem, a obsługą błędów
    - Standardowy przepływ jest wymieszany z blokami obsługi błędów
  - Wystąpienie błędu sygnalizowane jest jakąś specjalną wartością zwracaną
    - Pojawią się problemy kiedy funkcja jako legalną wartość może zwrócić zbiór pełny (np. wszystkie liczby typu `int` lub znaki, itp.)

Języki obiektowe (C++)

- W takich językach wiele operacji w ogóle nie zwraca żadnej wartości, czyli nie ma możliwości zwrócenia wartości sygnalizującej błąd
  - Np. tworzenie nowych obiektów (wywoływany jest konstruktor)
  - Wykrycie błędu to nie jeden problem, istotne jest również poprawne jego obsłużenie
- Istnieje potrzeba wbudowania mechanizmu, który pozwoliłby na oddzielenie wykrywania błędów od ich obsługi oraz umożliwiał przekazywanie informacji w inny sposób niż parametry zwracane
- Właśnie obsługa wyjątków daje takie możliwości

Koncepcja obsługi wyjątków

- Wyjątki przetwarzane są w języku C++ w następujący sposób
  - Jeżeli niespodziewana sytuacja wystąpi wewnątrz funkcji to zostanie to zakomunikowane za pomocą specjalnej instrukcji
    - Powoduje to przełączenie z normalnego trybu wykonywanie programu do obsługi wyjątków
    - W trybie tym opuszczane są wszystkie wywołane dotąd funkcje lub bloki, aż zostanie napotkany kod obsługi danego wyjątku
  - Dla poszczególnych instrukcji programu można definiować sposób działania jeśli pojawi się wyjątek

Słowa kluczowe służące obsłudze wyjątków

- **`try`** – służy określeniu zakresu instrukcji programu, w których wyjątki są przechwytywane i wysyłane do bloku obsługi błędów
- **`throw`** – umożliwia wyrzucenie obiektu wyjątku do programu
  - Powoduje przełączenie trybu pracy z normalnego do obsługi wyjątków
- **`catch`** – stosowane w celu przyjęcia obiektu wyjątku, a następnie jego obsługi
  - Zdefiniowany zakres wykonuję się podczas opuszczenia normalnego trybu pracy programu

Blok `try` i `catch`

- Wchodząc w programie do obszaru ryzykownego powinniśmy uprzedzić o tym kompilator
  - ```
    try {
            //ryzykowne instrukcje
    }
    catch(…)
    { /*reakcja na wyjątki */ }
    ```
- Wszystko co znajduje się w bloku `try` jest chronione, nawet wywołanie innych funkcji łącznie z bibliotecznymi
  - Niezależnie jak „głęboko" zostanie wyrzucony wyjątek
    ```

Instrukcja `throw`

- Jeżeli dzieje się coś niespodziewanego używamy instrukcji **`throw`**
  - **`throw obiekt;`**
- Możemy wyobrazić sobie dwie sytuacje
  - Rzucamy obiekt, który sam w sobie jest informacją o rodzaju sytuacji wyjątkowej
  - Rzucamy obiekt, który w sobie zawiera dodatkowe informacje o danej sytuacji wyjątkowej
- Różnica jest tylko widoczna od strony obsługi wyjątków, natomiast od strony sygnalizacji żadnej różnicy nie ma

Blok `catch`

- W bloku **`catch`** umieszczamy procedury obsługi wyjątku (-ów)
- Blok **`catch`** może tylko wystąpić bezpośrednio po bloku **`try`** lub innym bloku **`catch`**
- Bloków **`catch`** może być więcej, gdyż mogą one łapać obiekty różnych typów
  - Wtedy każdy blok **`catch`** przystosowany jest od złapania jednego konkretnego typu obiektu
  - Możliwe jest także umieszczenie takiego bloku **`catch`**, który złapie wszystkie wyjątki niezależnie od typu obiektu jak został wyrzucony
- Przykład cpp_8.1

Różnicie między wywołaniem obsługi błędów, a wywołaniem funkcji

- Obiekty zwracane
  - Funkcja może zwracać obiekty ściśle określonego typu i żadne inne
  - Instrukcja **`throw`** może wyrzucać obiekty dowolnego typu
- Różnica w przeniesieniu sterowania
  - Instrukcja **`return`** powoduje powrót do miejsca, skąd funkcja została wywołana
  - Instrukcja **`throw`** powoduje bezpowrotne opuszczenie wszystkich dalszych instrukcji (funkcji) i przenosi wykonanie do bloku **`catch`**

Kolejność bloków `catch`

- Kolejność bloków obsługi wyjątków ma istotne znaczenie
- Sytuacja bardzo podobna do instrukcji warunkowej `if`, `else if` i `else`
- Nie ma znaczenie czy np. w następnym bloku dopasowanie obiektu jest lepsze, zawsze wykonany zostanie ten blok, do którego jako pierwszego rzucany obiekt pasuje
  - Może to mieć szczególne znaczenie jeśli posługujemy się hierarchią klas

Bloki `try` i `catch` można zagnieżdżać

- Czasami może wydawać się lepsze zastosowanie zagnieżdżonej struktury bloków **`try`** i **`catch`**
  - Jeżeli rozróżnimy sytuacje wyjątkowe, z którymi możemy sobie poradzić lokalnie od sytuacji trudniejszych kiedy obsługa wyjątku musi odbyć się w dalszej części programu
- Jeśli instrukcja **`throw`** występuje w zagnieżdżonym bloku **`try`** to najpierw następuje próba obsługi wyjątku w blokach **`catch`** stojących bezpośrednio za nim. Dopiero jeżeli tam nie będzie możliwe obsłużenie wyjątku sprawdzane są bloki znajdujące się za zewnętrznym blokiem **`try`**
- Przykład cpp_8.2

Dopasowywanie typów w blokach `catch`

- Dana procedura obsługi nadaje się do pracy z danym typem jeżeli
  - Typ argumentu rzucanego jest taki sam jak typ argumentu oczekiwanego
  - Jeżeli typ argumentu oczekiwanego ma dodatkowo przydomek `const`
  - Gdy rzucamy dany typ, a oczekiwanym typem jest referencja do niego
  - Typ argumentu oczekiwanego jest publiczną klasą podstawową w stosunku do typu rzucanego
    - Bardzo nietypowe!!!
  - Typ argumentu rzucanego jest wskaźnikiem do jakiegoś typu, a oczekiwany typ jest wskaźnikiem do którego typ rzucany może być skonwertowany za pomocą konwersji standardowej
- Przykład cpp_8.3