Instytut Informatyki UMCS
Zakład Technologii Informatycznych

Wykład 6

Wyjątki

dr Marcin Denkowski

Lublin, 2019

AGENDA

1. Przestrzenie nazw
2. Wyjątki i bloki **`try..catch`**
3. Klasy wyjątków **`std::exception`**

PRZESTRZENIE NAZW

- Przestrzeń nazw jest mechanizmem służącym do logicznego grupowania

```
namespace nazwa-przestrzeni{
   // deklaracje i definicje
    int zmienna;
}
```

- Dostęp do elementów przestrzeni z kwalifikatorem ::

```
    nazwa-przestrzeni::zmienna = 5;
```

- Przestrzeń nazw jest otwarta – można do niej dodawać nazwy w kilku deklaracjach:

```
namespace nazwa-przestrzeni{
    int kolejna_zmienna;
}
```

DYREKTYWY UŻYCIA

- Dyrektywa użycia

```
using namespace nazwa-przestrzeni;
```

- włącza wszystkie nazwy z tej przestrzeni do przestrzeni globalnej

- Deklaracja użycia

```
using nazwa-przestrzeni::zmienna;
```

- włącza tylko tą konkretną zmienną

- Użycie dyrektywy using może prowadzić do konfliktu nazw
- Użycie deklaracji using przesłania istniejącą zmienną

WYJĄTKI

- Mechanizm obsługi wyjątków – nielokalna struktura sterująca, która korzysta ze zwijania stosu
- Wyjątek jest obiektem pewnego typu
- Kod wykrywający błąd zgłasza (rzuca **`throw`**) ten obiekt
- Wyjątek obsługuje się w bloku **`try..catch`**
- Wynikiem rzucenia wyjątku jest zwinięcie stosu aż do odnalezienia odpowiedniego **`catch`**

WYJĄTKI

- Blok try..catch

```
try
{
   throw type1;
   throw type2;
   throw other;
}
catch(type 1 name)
{
}
catch(type 1 name)
{
}
catch(...)
{
}
```

WYJĄTKI

- Kiedy jest dopasowanie obiektu wyjątku do bloku **`catch`**

1) **`H`** jest tego samego typu co **`E`**

2) **`H`** jest jednoznaczną publiczną klasą podstawową dla **`E`**

3) **`H`** i **`E`** są typami wskaźnikowymi i zachodzi 1) lub 2)

4) **`H`** jest referencją i zachodzi 1) lub 2)

```
try
{
    throw E();
}
catch(H)
{
    // kiedy
}
```

- Rzucany obiekt wyjątku jest kopiowany do obsługi.

PONOWNE ZGŁOSZENIE

- Rozpraszanie obsługi wyjątku

```
try
{
    // kod zglaszajacy wyjatek
}
catch(H)
{
    // obsluga wyjatku lub rzucenie go dalej
    throw;
}
```

KAŻDY WYJĄTEK

- Wyłap każdy wyjątek

```
try
{
    // kod zglaszajacy wyjatek
}
catch(...)
{
    // obsluga wyjatku lub rzucenie go dalej
    throw;
}
```

SPECYFIKOWANIE WYJĄTKÓW

- Funkcja o deklaracji

```
void g();
```

- może zgłosić każdy wyjątek

- Funkcja o deklaracji

```
void g() throw();  lub  void g() noexcept;
```

- nie zgłasza żadnych wyjątków

- Funkcja o deklaracji

```
void g() throw(H, E);
```

- może zgłosić wyjątki H i E

NIEWYŁAPANE WYJĄTKI

- W przypadku niewyłapania wyjątku wywoła się funkcja

```
void std::terminate();
```

- która wywoła funkcję `abort()`

- Działanie funkcji `terminate()` można zmienić ustawiając nową reakcję poprzez:

```
typedef void (*terminate_handler)();
terminate_handler set_terminate(terminate_handler)
```

PLIK <exception>

- W pliku <exception> zdefiniowana jest struktura standardowych wyjątków
- Wszystkie klasy wyjątków w standardowej bibliotece wywodzą się z **`std::exception`**

- `logic_error`
  - `invalid_argument`
  - `domain_error`
  - `length_error`
  - `out_of_range`
  - `future_error`
- `bad_typeid`
- `bad_cast`
- `bad_weak_ptr`
- `bad_function_call`
- `bad_alloc`
  - `bad_array_new_length`
- `bad_exception`

- `runtime_error`
  - `range_error`
  - `overflow_error`
  - `underflow_error`
  - `regex_error`
  - `system_error`
    - `ios_base::failure`

KLASA STD::EXCEPTION

```
namespace std
{
    class exception
    {
    public:
        exception() noexcept;
        exception(const exception& other ) noexcept;

        virtual ~exception();

        exception& operator=(const exception& other )
                                noexcept;

        virtual const char* what() const noexcept;
    };
}
```