C++11…17 i nowsze - rdzeń języka i ulepszenie funkcjonalności

Wykład 10

Pela for w zakresie

- Przydatna do iterowania po zakresie
  - `for ( range_declaration : range_expression ) loop_statement`
  - Często wykorzystywana razem z `auto`
  - Preferowana forma
    - `for(auto&& var : sequence)`
- Posiada bardzie intuicyjną formę w stosunku do standardowego for
  - Oczywiście kiedy mam do dyspozycji zakres
- Przykład 16

Nowa składnia deklaracji i definicji funkcji

- Składania deklaracji przejęta w jeżyka C nie jest już dopasowana wystarczająco dobrze no nowych potrzeb języka
  - Szczególnie problem przy szablonach
    - ```
      template< typename LHS, typename RHS>
      Ret // NIEPRAWIDŁOWE!
      AddingFunc(const LHS &lhs, const RHS &rhs) {return lhs + rhs;}
      ```
      - `Ret` – cokolwiek wynikające z sumy lhs + rhs
    - ```
      template< typename LHS, typename RHS>
      decltype(lhs + rhs) // NIEPRAWIDŁOWE!
      AddingFunc(const LHS &lhs, const RHS &rhs) {return lhs + rhs;}
      ```
      - Jest to nielegalne w C++ ponieważ lhs i rhs nie są jeszcze zdefiniowane
- C++11 wprowadza nową składnię deklaracji i definicji funkcji:
  - ```
    template< typename LHS, typename RHS>
    auto AddingFunc(const LHS &lhs, const RHS &rhs)
      -> decltype(lhs + rhs)
    {return lhs + rhs;}
    ```
- C++14 naprawia problem dedukcji typu zwracanego:
  - ```
    template< typename LHS, typename RHS>
    auto AddingFunc(const LHS &lhs, const RHS &rhs)
    {return lhs + rhs;}
    ```
- Dotyczy to oczywiście zwykłych funkcji i funkcji składowych
- Przykład 15

Wyrażenia lambda

- Inaczej nienazwane obiekty funkcjne
  - `[ capture-list ] ( params ) mutable(optional) exception -> ret { body }`
  - `[ capture-list ] ( params ) -> ret { body }`
  - `[ capture-list ] ( params ) { body }`
  - `[ capture-list ] { body }`
- Znaczenie
  - `mutable` – pozwala na modyfikacje obiektów przesłanych przez wartość wewnątrz funkcji
  - `exception` – pozwala na określenie specyfikacji wyjątków dla operatora()
  - `capture-list` – lista przyjmowanych zmienny automatycznych
    - [a,&b] a jest przesyłane przez wartość a b przez referencję.
    - [&] wszystkie automatyczne zmienne przez referencję
    - [=] wszystkie automatyczne zmienne przez wartość
    - [this] - obsługa wskaźnika this, wskazującego na obiekt obsługiwany przez daną metodę, jest specjalna i musi być wyraźnie zaznaczona w funkcji lambda
    - [] – nic nie jest przekazywanie
  - `params` – lista parametrów (jak w funkcji)
  - `ret` – typ zwracany
  - `body` – ciało funkcji
- Przykład 19 i 20

Inne ciekawostki

- **`extern template`**
  - Zmusza kompilator do nietworzenia instancji szablonu w danej jednostce kompilacji
  - Działa pod warunkiem, że gdzieś indziej taka instancja szablonu zostanie stworzona
    - Znacząco może przyspieszyć proces kompilacji
- Aliasy typów i szablony (**`using`**)
  - W stosunku do typów odpowiednik **`typedef`**
  - Alias do szablonu może uprościć operacje na typach z jednej rodziny
  - Przykład 21

Usprawnienie obsługi wyjątków

- `std::exception_ptr`
  - Jest obiektem podobnym do wskaźnika
  - Zarządza wyrzuconym obiektem, który został złapany przez `std::current_exception`
  - Instancja `std::exception_ptr` może być przekazana do innej funkcji a nawet do innego wątku gdzie wyjątek może zostać wyrzucony ponownie i przetworzony
  - Dwie instancje `std::exception_ptr` są sobie równe jeżeli są puste lub pokazują na ten sam obiekt wyjątku
  - Obiekt wyjątku jest dostępny do czasu kiedy chociaż jeden `std::exception_ptr` pokazuje na niego
    - Jak zmyślny wskaźnik
- `std::current_exception()`
  - Jeśli wywołana w bloku `catch` to łapie obecny wyjątek i tworzy `std::exception_ptr`
- `std::rethrow_exception(std::exception_ptr)`
  - Wyrzuca wyjątek przechowywany w `std::exception_ptr`
- Przykład 22

async, future (i promise, package_task() …)

- Pozwalają na wykonywanie współbieżności zadaniowej
  - Nie wymaga skupiania się na wątkach
    - Problemach synchronizacji, unikaniu race conditions etc.
- **`std::async`**
  - Uruchamia funkcję asynchronicznie (potencjalnie w innym wątku)
  - Zwraca **`std::future`**, który zawierać będzie rezultat
- **`std::future`**
  - Zapewnia mechanizm dostępu do wyniku asynchronicznej operacji
- Przykład 23 i 24