(Średnio) zaawansowane programowanie w C++ (ZPR)

Wykład 15 - Uwagi końcowe

Robert Nowak

24L

Standard C++

Standard C++, język C++ jest zgodny wstecz

- ISO/IEC 14882, opublikowany w 1998 (C++98)
- ISO/IEC 14882:2003, zmodyfikowany w 2003 (C++03)
- ISO/IEC 14882:2011, C++11, c++0x (III 2011), draft:N3290
  `http://www.open-std.org/jtc1/sc22/wg21`
- ISO/IEC 14882:2014, C++14, c++1y (VIII 2014), draft:N3797
- ISO/IEC 14882:2017, C++17, c++1z (XII 2017), draft:N4661
- ISO/IEC 14882:2020, C++20, c++2a (XII 2020), draft:N4878
- jeszcze (2024.06.09) nie ogłoszony, C++23, draft:N4950 (XII 2023)
  `git clone https://github.com/cplusplus/draft.git`
- C++26, draft:N4981 (2024.04.16)

Boost – zbiór $\approx$ 100 bibliotek eksperymentalnych związanych ze standardem C++.
`http://boost.org`, wersja 1.85, kwiecień 2024

Standard Python, Python Software Fundation

3 wersje standardu (niezgodne wstecz!):
Python 1 (nie używana), Python 2 (wycofana od 2020), Python 3

Definicja standardu w Python Enhancement Proposals (PEP), `http://python.org`, przykłady:

- PEP 0 - definicja PEP, lista PEP (jest ich ponad 8000)
- PEP 8 - styl kodowania
- PEP 20 - 'The Zen of Python' (przesłanie)

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
...
```

Konferencja PyCon: `http://pycon.org`

Standard Rust, rust-lang.org

obecnie stabilna wersja 1.78 (od 02.05.2024)

- codziennie nowa wersja,
- co 6 tygodni wersja beta, która następnie jest ogłaszana jako stabilna
- nie ma gwarancji zgodności

Zmiany w git:
`https://github.com/rust-lang/rust`

Uzgadnianie zmian: RFC (request for comments),
`https://rust-lang.github.io/rfcs`
np: `0002-rfc-process.md`, opisuje RFC dla Rust

Dokumentacja

Dokumentacja w kodzie zmniejsza ryzyko braku spójności

Komentarze - poprawiają czytelność kodu.

Komentarz mówi DLACZEGO, kod mówi JAK.

- każdy byt powinien mieć pojedynczą odpowiedzialność,
- dokumentacja projektowa powinna być generowana z kodu.

Hierarchia komentarzy:

1. odpowiedzialność bibliotek, pakietów, przestrzeni nazw, katalogów
2. odpowiedzialność modułów (plików), klas,
3. odpowiedzialność metod publicznych,
4. niebanalne algorytmy

C++11, uzupełnienie

- wyrażenia stałe, **constexpr**, możliwość używania w czasie kompilacji
- ```
  class Foo { //delegacja konstruktora
  public:
    Foo(const std::string& s) { /* ... */ }
    Foo(const char* c) : Foo(std::string(c)) {} //delegacja
    /* ... */

  //Wskazuje, aby nie tworzyć instancji w bieżącym module
  external template class std::vector<Foo>;
  ```
- nazwy dla typów za pomocą **using**

  ```
  using MyInt = int; //typedef int MyInt
  using MyF = void (*)(int,int); //typedef void (*MyF)(int,int);
  template<class T> using Vec = std::vector<T,std::allocator<T>>;
  Vec<int> v; //std::vector<int, std::allocator<int> > v;
  ```
- operator typu wyrażenia **decltype**

  ```
  using size_t = decltype(sizeof(0));
  using nullptr_t = decltype(nullptr);
  ```
  ```

Standard C++14, uzupełnienie

```
//dedukcja typu zwracanego, kompilator określa, że zwracamy double
auto getValue() {
  return 1.0;
}
```

przydatna do redukcji modyfikacji kodu:

```
struct Record {
  int id; //identyfikator
  std::string name;
};
//auto find_id(...)  pozwoli uniknąć modyfikacji kodu po zmianie typu
//identyfikatora w klasie Record
int find_id(const vector<Record>& records, const string& name) {
  auto it = find_if(records.begin(), records.end(),
                    [&](const Record& r){ return r.name==name; } );
  if(it == records.end())
    return -1;
  return it->id;
}
```

Standard C++14, uzupełnienie (2)

- **atrybut [[deprecated]]**

```
class [[deprecated]] Foo {};

//kompilacja kodu, który używa Foo dostarczy ostrzeżenie, np.
test.cpp:4:6: warning: ‘’Foo is deprecated ...
```

- literały binarne: `int val = 0b10101010;`
- separator przy definiowaniu ciągu cyfr (nie ma wpływu na wartość)

```
int mask = 0b1111’0000’1111’0000;
```

- zmienne szablonowe, np. **pi<T>**

```
cout << pi<int> << endl; //3
cout << pi<std::string> << endl; //napis pi
```

- generyczne funkcje anonimowe, (użycie `auto` w `lambda`) - podobne do szablonów

```
auto add = [](auto x, auto y) { return x + y ; }
```

Standard C++17, uzupełnienie

- std::string_view (referencja tylko do odczytu do części napisu)
- biblioteka do systemu plików (bazuje na Boost.Filesystem)
- usunięcie alternatywnych reprezentacji znaków specjalnych (trigraph), tzn ??= (reprezentuje #), ??/ (reprezentuje \) itp.
- inicjacja (opcjonalna) dla `if`, `switch` i `while`

```
if( int c = get(); c != SUCCESS ) { //'c' istnieje w bloku
  /* ...  */
}
```

- zmienne `inline`
- dodany typ `std::void_t`
- dodany typ `std::byte` - reprezentacja bajtów, niedozwolone operacje arytmetyczne, nie jest to typ znakowy
- literały szesnastkowe do reprezentacji zmiennopozycyjnej (IEEE 754)

```
double v1 = 1.2e3 //120.0
double v2 = 0x1.2p3; //1.125*2^3 = 9.0
```

Standard C++17, uzupełnienie (2)

- algorytmy STL mogą być wykonywane równolegle (współbieżnie lub wektorowo)

```
vector<double> v(1024);
fill( execution::unseq, v.begin(), v.end(), 1.0 );
for_each( execution::unseq, v.begin(), v.end(),
          [](double& d) { return sin(d) + 2.0*cos(d);});
```

  - `std::execution::sequenced_policy`
  - `std::execution::parallel_policy` - można używać wielu wątków
  - `std::execution::unsequenced_policy` - można stosować operacje wektorowe
  - `std::execution::parallel_unsequenced_policy`

Standard C++20, uzupełnienie

operator <=>, porównanie trójstronne (three-way comparison)

$$(a <=> b) \begin{cases} < 0 & \text{jeżeli a < b} \\ == 0 & \text{jeżeli a == b} \\ > 0 & \text{jeżeli a > b} \end{cases}$$

podobnie jak `strcmp` w 'C'

```
std::strong_ordering::equal
std::strong_ordering::less
std::strong_ordering::greater
std::strong_ordering::unordered
```

Dostarczone dla:

- typów liczbowych (całkowite, zmiennopozycyjne)
- wskaźników (arytmetyka adresowa)
- tablic (napisów)
- możliwość dostarczanie dla typów użytkownika

Standard C++20, uzupełnienie (2)

- nowa postać pętli `for` dla zakresu, `for (init; decl : expr)`

```
vector<Foo> vec:
for(int i = 0; Foo f : vec) {
  bar(f, i);
  ++i;
}
```

- napisy jako parametry szablonów (podobnie jak liczby całkowite, napis nie jest typem)

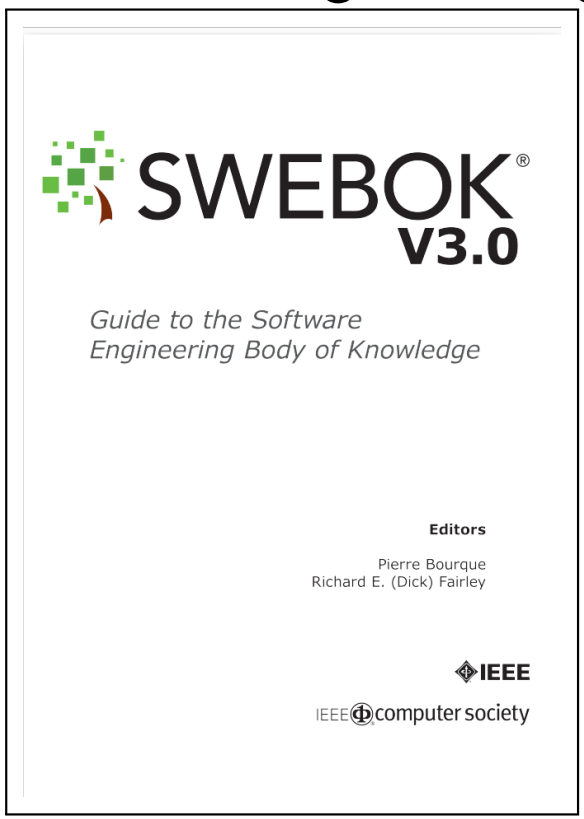
```
Foo<"foobar"> f;
```

C++20 dostępny w następujących narzędziach:

kompilator	język	biblioteka standardowa
GNU gcc	10	12
Visual Studio	2022, (oraz 2019 od ver. 16.9)	2022
CLang	11	13

SWEBOK

IEEE Computer Society,
Guide to the Software Engineering Body of Knowledge
`https://www.computer.org/education/bodies-of-knowledge/software-engineering/v3`



- zarządzanie wymaganiami,
- projektowanie i implementacja oprogramowania,
- testowanie,
- dostarczanie, konfiguracja, pielęgnacja,
- zarządzanie jakością,
- zarządzanie zespołem.

Podsumowanie

KISS (Keep it simple software)

BUZI (bez udziwnień zbędnych idioto)

Dziękuję