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

Wykład 15 - LLM w wytwarzaniu oprogramowania. C++14,C++17,C++20,C++23.

Robert Nowak

25Z

Plan wykładu

- zintegrowanie narzędzia do tworzenia oprogramowania (IDE)
- standardy C++, Python, Rust.
- biblioteka standardowa C++, uzupełnienie;

Zintegrowane narzędzia do tworzenia oprogramowania (IDE)[2]

Nazwa	Licencja	Popularność[1]	Platforma
VS Code	Komer. + wolna	73.6%	wiele
Visual Studio	Komer.	29.3%	Windows, macOS
JetBrains	Komer. + Apache 2.0	94.0%	wiele
Vim/Neovim	Vim + Apache 2.0	34.1%	wiele
Sublime Text	Komer. + wolna	10.9%	wiele
Eclipse	EPL 2.0	9.4%	wiele
XCode	Komer. + wolna	9.3%	macOS
VSCodium	MIT	4.8%	wiele
Emacs	GPL v3.0	4.6%	wiele

[1] Ankieta, 2024, 58tys. uczestników

[2] Na podstawie raportu M. Borek, 2024

Narzędzia wspierające tworzenie kodu[3]

Nazwa	Open Source	Objaśnianie	CLI	Praca lokalna
GitHub/Copilot	-	tak	tak	-
Tabnine	-	tak	-	tak
Codeium	-	tak	-	-
Amazon Whisperer	-	tak	tak	-
Cody	tak	tak	tak	-
Continue	tak	tak	-	tak
CodeGeeX	tak	tak	-	-
FauxPilot	tak	-	-	tak
Tabby	tak	tak	-	tak

[3] Na podstawie raportu M. Borek, 2024

Integracja narzędzi z IDE[4]

Nazwa	VS Code	V S	Vim/ Neo	Jet Bra.	Ecli- pse	Sub- lime	X Code	Cod- ium	Em- acs
Copilot	tak	tak	tak	tak	-	-	-	tak	tak
Tabnine	tak	tak	tak	tak	tak	-	-	tak	tak
Codeium	tak	tak	tak	tak	tak	tak	tak	tak	tak
Am. W.	tak	tak	-	-	-	-	-	tak	-
Cody	tak	tak	-	tak	tak	-	-	tak	-
Continue	tak	tak	-	tak	-	-	-	tak	-
CodeGee	tak	tak	-	tak	-	-	-	tak	-
FauxPilot	-	tak	tak	-	-	-	-	tak	-
Tabby	tak	tak	tak	tak	-	-	-	tak	-

[4] Na podstawie raportu M. Borek, 2024

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
- ISO/IEC 14882:2024 C++23, c++2b (XII 2023) draft:N4950
- C++26 draft:N5032 (XII 2025)
  `git clone https://github.com/cplusplus/draft.git`

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 1.87 (od 15.05.2025)

- 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.

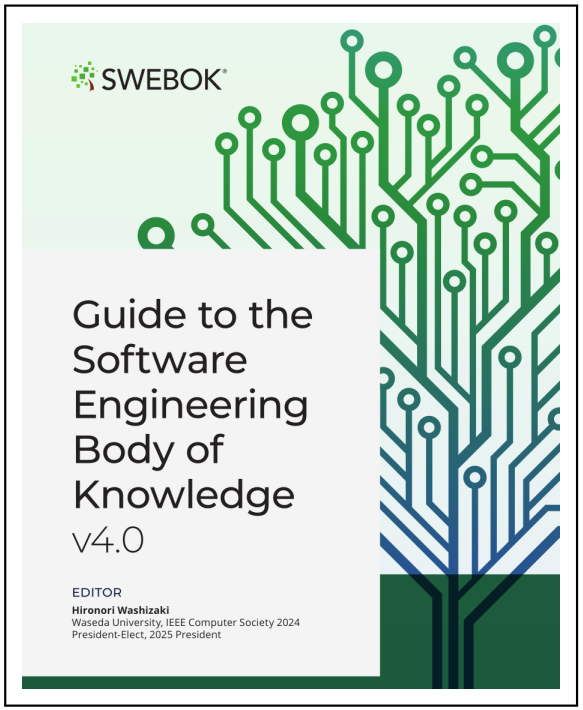
- każdy byt powinien mieć pojedynczą odpowiedzialność,
- dokumentacja projektowa powinna być generowana z kodu.

Hierarchia komentarzy:

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

SWEBOK

IEEE Computer Society,
Guide to the Software Engineering Body of Knowledge
`https://www.computer.org/education/bodies-of-knowledge/software-engineering/v4`



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

std::ref <functional>

`std::ref(X)` obiekt zaw. wskaźnik do X, możliwość kopiowania, itp.

`std::cref(X)` obiekt zaw. stały wskaźnik do X

```
int foo = 10;
auto bar = std::ref(foo);
++bar; //działa na obiecie, foo == 11

void thrd_fun(Data& data) { /* nieistotne */ }
Data data;
auto thr1 = std::thread( thread_fun, data ); //pracuje na kopii
auto thr2 = std::thread( thread_fun, ref(data) ); //pracuje na oryginale
```

rekordy, std::pair

```
template<class _T1, class _T2>
struct pair {
  typedef _T1 first_type;
  typedef _T2 second_type;
  _T1 first;//Pierwsza składowa
  _T2 second;//Druga składowa
  pair() : first(), second() { }
  pair(const _T1& __a, const _T2& __b)
  : first(__a), second(__b) { }
};
//Przykłady użycia
using StringCount = std::pair<std::string,int>;
StringCount f();//funkcja zwraca dwie wartości
//Tworzy parę o typach dedukowanych z argumentów
std::make_pair(24,true);
```

rekordy, std::tuple

```
//struktura zawierająca trzy obiekty (typów A,B,C)
using triple = std::pair<A,std::pair<B,C> >;
tuple<A,B,C> //Zawiera trzy obiekty (typów A,B,C)
```

- dostęp do składowej - get

```
int i = 3;
tuple<int,double,int&> t(2,1.5,i);
t.get<0>() = 4; //dostęp do elementu
double d = get<1>(t); //dostęp do elementu
```

- make_tuple - nie trzeba podawać typów obiektów
- operatory: ==, !=, <, >, <=, >=
- funkcja tie

```
int i; double d;
tie(i,d); //make_tuple(ref(i),ref(d)), zwraca tuple<int&,double&>
```

std::function (C++11)

Pozwala przechowywać funkcje i obiekty funkcyjne

- obiekt który akcje (np. wskaźnik do funkcji lub metody)
- możliwość kopiowania, przypisywania itp.

```
std::function<void (int, int)> pf; //funkcja dwuargumentowa
//boost::function2<void, int, int> pf; - to samo
void f(int x, int y) { cout << x + y << endl; }
pf = f;
pf(2,3);//wywołanie f dla x = 2, y = 3
struct F {
  void operator()(int x, int y);
};
pf = F();
pf(2,3);//wywołanie F.operator() dla x = 2, y = 3
pf = [](int x, int y){ cout << x + y << endl; };
pf(2,3);
```

std::function - przykłady

```
using PF = std::function<void (int)>;
PF pf; //Obiekt, który przechowuje komendę
pf(3); //Wyjątek: boost::bad_function_call
class Foo { public: void f(int x); /* ... */ };
Foo foo;
//Związanie obiektu dla którego będzie wołana metoda
pf = std::bind(Foo::f,ref(foo),_1);
pf(4); //Woła foo->f(4)

vector<PF> callbacks; //Możliwość przechowywania w kolekcjach
//Woła wszystkie metody w kolekcji (z argumentem = 7)
for_each(callbacks.begin(), callbacks.end(), [](PF& pf){ pf(7);});
```

Zastosowanie : wszędzie tam, gdzie wzorzec komendy

- separacja tworzenia akcji od jej wołania
- kolekcjonowanie poleceń

inne typy pomocnicze

- `std::variant` (C++11) – unia z wyróżnikiem bieżącego typu
- `std::optional` (C++11) – obiekt z sygnalizacją braku wartości

`std::any` (C++17)

kontener na wartość dowolnego typu (jak `void*`)

```
#include <any>
any a; //dla pustych obiektów wartość przechowywanego typu to void
assert(a.has_value() == false);
a = string("Hello"); //a przechowuje wartość string
assert(a.type() == typeid(string) );
string s = any_cast<string>(a); //dostęp do obiektu

 //wyjątek bad_any_cast, gdy niezgodne typy
template <typename Val> Val any_cast(const any& a);
 //nullptr gdy niezgodne typy
template <typename Val> const Val* any_cast(const any* a);
template <typename Val> Val* any_cast(any* a);
```

Tablice wielowymiarowe, Boost.Multi_Array

Wygodne do reprezentacji macierzy, tensorów, itp.

Typowe implementacje:

- tablice wielowymiarowe z C, np. `int tab[2][3];`
- wektory wektorów, np. `vector<vector<int> > tab;`
- `boost::multi_array <boost/multi_array.hpp>`

konstruktor	`//liczba elementów - kontener`
	`array< array_type::index, 2> shape = { 2, 3 };`
	`//wymiar - parametr szablonu`
	`multi_array<double, 2> matrix( shape );`

dostęp	`matrix[1][1] = 2.56; //operator indeksowania`
	`//kontener z indeksami`
	`array< array_type::index, 2> index = { 1, 1 };`
	`matrix( index ) = 2.56;`

Tablice wielowymiarowe `std::mdspan` (C++23)

- std::span (C++20) - widok na ciągłą sekwencją obiektów
- std::mdspan (C++23) – widok na skewencję jak na wielowymiarową tablicę

```
std::array a = {0,1,2,3,4,5,6,7,8,9,10,11};
mdspan(a.data(), 2, 6 ); //widok na tablice 2D
//[ [ 0, 1, 2, 3, 4, 5 ], [ 6, 7, 8, 9, 10, 11 ] ]
mdspan(a.data(), 3, 4 ); //widok na tablice 2D
//[ [ 0, 1, 2, 3], [ 4, 5, 6, 7], [ 8, 9, 10, 11] ]
mdspan(a.data(), 2, 2, 3 ); //widok na tablice 3D: 2 x 2 x 3
//[ [ [ 0, 1, 2], [3, 4, 5] ], [ 6, 7, 8], [9, 10, 11] ] ]
```

Pomocnicze: `std::rank` - zwraca liczbę wymiarów, `std::extent` - liczba elementów dla danego wymiaru

C++11, uzupełnienie

- **constexpr**, wyrażenia, funkcje i in. obliczane 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>;
  ```
- 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)

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

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++23 dostępny w następujących narzędziach:

kompilator	język	biblioteka standardowa
GNU gcc	12	15
Visual Studio	2022 (od ver. 19.44)	2022 (od ver. 19.44)
CLang	16	17

Podsumowanie

KISS (Keep it simple software)

BUZI (bez udziwnień zbędnych idioto)

Dziękuję