Przydatne funkcjonalności języka C++

Arkadiusz Jędrzejewski
Gdańsk Embedded Meetup 9.05.2023

O mnie

Link do profilu na LinkedInie

Wstęp

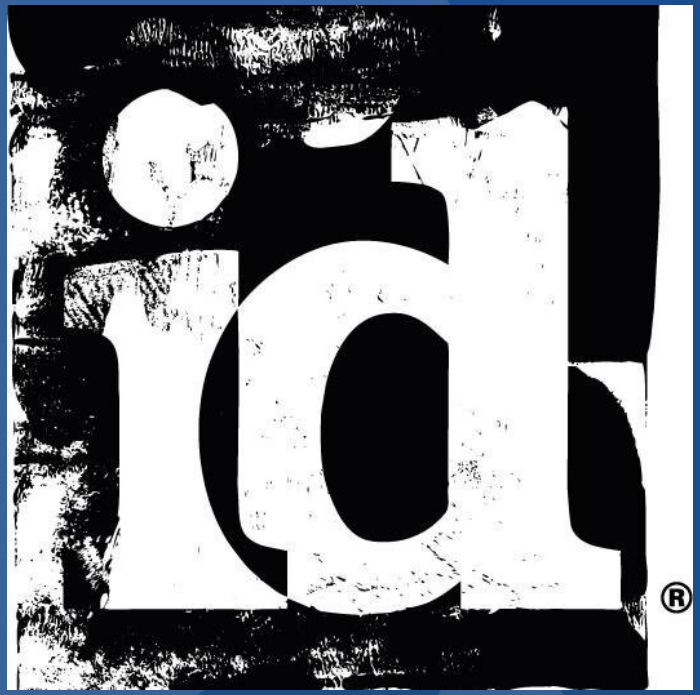

Część 1: Filozoficzne biadolenie
Część 2: Mięcho
Rozszerzenia języka C w kompilatorze GNU
Panicz Maciej Godek
godek.maciek@gmail.com
Gdańsk Embedded Meetup#5, 08.11.2022
Panicz Maciej Godek
Rozszerzenia języka C w kompilatorze GNU

Kuuk Thaayorre

Różnice w pamiętaniu zdarzeń

- Angielski:
  ”John broke the vase”
- Hiszpański
  ”Se rompió el florero”

A co z językami formalnymi?



Cechy języków funkcyjnych

- Czyste funkcje
- Brak mutowalności zmiennych

- ...i inne, ale my nie o tym

Korzyści

- Bezpieczeństwo wątków
- Lepsza reużywalność kodu
- Łatwiejsze testowanie
- Łatwiejsza analiza kodu

Zalety korzystania z innego języka

- Zmiana perspektywy
- Trafniejsze wyrażenie intencji
- Unikanie neologizmów

Dlaczego akurat C++!?

- Bo jest dostępny
- Bo jest dopasowany do domeny w jakiej się poruszacie

Zarządzanie pamięcią i zasobami

Ownership

- Każda zmienna ma właściciela
- Może być tylko jeden właściciel
- Jeśli właściciel wychodzi ze scope'a, to zmienna jest uwalniana

- Potrzebne są narzędzia: move, reference i RAII

RAII

- Resource Acquisition Is Initialization
- Wiąże czas życia zasobu z czasem życia obiektu
- C# ma IDisposable, Python ma context managery

unique_ptr

Przykład RAII

Przykład unique_ptr z własnym deleterem

```
using FilePtr = std::unique_ptr<FILE,
    decltype([](FILE* f) { std::fclose(f); })>;
```

Lambdy i biblioteka <algorithm>

Lambda expressions (od C++11)

- Nienazwane funktory zdolne do przechwytywania stanu zmiennych

<algorithm>

- min, max, for_each, copy, remove, replace, shuffle, operacje na zbiorach, stertach itd.
- Omówię tylko kwantyfikatory – all_of, any_of, none_of (od C++11)

Unikanie dynamicznego polimorfizmu

CRTP

- Curiously Recurring Template Pattern

std::variant (od C++17)

- Bezpieczniejsza alternatywa dla unii z C

Obsługa błędów

std::expected (od C++23)

nodiscard (od C++17)
likely, unlikely (od C++20)

```
error_handling_expected.cpp: In function ‘Result<long unsigned int> parse_and_count_entries(std::filesystem::__cxx11::path)’:
error_handling_expected.cpp:61:18: warning: ignoring return value of ‘Result<std::vector<PersonData> > parse_csv(std::filesystem::__cxx11::path)’, declared with attribute ‘nodiscard’ [-Wunused-result]
   61 |   parse_csv(path);
      |             ^
error_handling_expected.cpp:29:47: note: declared here
   29 | [[nodiscard]] Result<std::vector<PersonData>> parse_csv(std::filesystem::path path) {
      |                                               ^~~~~~~~~
```

Obsługa błędów w Ruście

```
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

std::optional (od C++17)

Wyjątki, std::expected i wydajność

- P2544R0 - C++ exceptions are becoming more and more problematic
- std::expected powoduje stały narzut na wydajność
- Wyjątki znacznie tracą na wydajności w środowiskach wielowątkowych

Pierwszy eksperyment

```cpp
struct invalid_value {};

void do_sqrt(std::span<double> values) {
   for (auto& v : values) {
      if (v < 0) throw invalid_value{};
      v = std::sqrt(v);
   }
}
```

Threads	1	2	4	8	12
0.0% failure	19ms	19ms	19ms	19ms	19ms
0.1% failure	19ms	19ms	19ms	19ms	20ms
1.0% failure	19ms	19ms	20ms	20ms	23ms
10% failure	23ms	34ms	59ms	168ms	247ms

Wyjątki

failure rate	0.0%	0.1%	1.0%	10%
sqrt	18ms	18ms	18ms	16ms

std::expected

Drugi eksperyment

```
struct invalid_value {};

unsigned do_fib(unsigned n, unsigned max_depth) {
   if (!max_depth) throw invalid_value();
   if (n <= 2) return 1;
   return do_fib(n - 2, max_depth - 1) + do_fib(n - 1, max_depth - 1);
}
```

Drugi eksperyment

Threads	1	2	4	8	12
0.0% failure	12ms	12ms	12ms	14ms	14ms
0.1% failure	14ms	14ms	14ms	14ms	15ms
1.0% failure	14ms	14ms	14ms	15ms	15ms
10% failure	18ms	20ms	27ms	64ms	101ms

Wyjątki

failure rate	0.0%	0.1%	1.0%	10%
sqrt	18ms	18ms	18ms	16ms
fib	63ms	63ms	63ms	63ms

std::expected

Narzędzia

Narzędzia

- Linter: clang-tidy, cppcheck
- Formatter: clang-format
- Menadżer pakietów: Conan, vcpkg

Fin

Źródła

- https://www.ted.com/talks/lera_boroditsky_how_language_shapes_the_way_we_think
- https://www.scientificamerican.com/article/how-language-shapes-thought/
- https://www.psychologytoday.com/us/blog/the-biolinguistic-turn/201702/how-the-language-we-speak-affects-the-way-we-think
- http://sevangelatos.com/john-carmack-on/
- https://abseil.io/about/philosophy

Źródła

- https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html
- https://en.cppreference.com/w/cpp/language/raii
- https://en.cppreference.com/w/cpp/memory/unique_ptr
- https://www.reddit.com/r/cpp/comments/x79zn1/using_stdunique_ptr_with_c_apis/
- https://en.cppreference.com/w/cpp/language/lambda

Źródła

- https://en.cppreference.com/w/cpp/algorithm/all_any_none_of
- https://johnnysswlab.com/the-true-price-of-virtual-functions-in-c/
- https://en.cppreference.com/w/cpp/utility/variant
- https://en.cppreference.com/w/cpp/utility/expected
- https://en.cppreference.com/w/cpp/language/attributes/nodiscard

Źródła

- https://en.cppreference.com/w/cpp/language/attributes/likely
- https://doc.rust-lang.org/book/ch09-02-recoverable-errors-with-result.html
- https://en.cppreference.com/w/cpp/utility/optional
- https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/p2544r0.html
- https://peps.python.org/pep-0020/

Źródła

- https://en.cppreference.com/w/cpp/language/Zero-overhead_principle
- https://clang.llvm.org/extra/clang-tidy/
- https://groups.google.com/g/comp.lang.lisp/c/7xCvdzijzgU/m/4xCFzLc3d5EJ