Przekazywanie wartości

dr inż. Paweł Kłeczek

2021

Przekazywanie wartości do i z funkcji – wytyczne

<table>
<tr><th></th><th>Cheap or impossible to copy (e.g., int, unique_ptr)</th><th>Cheap to move (e.g., vector<T>, string)
or Moderate cost to move (e.g., array<vector>, BigPOD)
or Don’t know (e.g., unfamiliar type, template)</th><th>Expensive to move (e.g., BigPOD[], array<BigPOD>)</th></tr>
<tr><td>Out</td><td colspan="2">return</td><td rowspan="2"></td></tr>
<tr><td>In/Out</td><td colspan="2">f(X&)</td></tr>
<tr><td>In</td><td rowspan="2">f(X)</td><td>f(const X&)</td><td rowspan="2"></td></tr>
<tr><td>In & retain copy</td><td>f(const X&) + f(X&&) & move</td></tr>
</table>

Uwagi do ramowych wytycznych

- **Parametry „do przeniesienia” przekazuj przez `X&&` (a następnie je przenoś).** [F.18]
  - **Wyjątek:** Wybrane typy o unikalnej własności (np. `std::unique_ptr`) przekazuj przez wartość.
    $\Rightarrow$ preferuj prostotę i czytelność (i bezpieczeństwo kodu) ponad wydajność
- **Zwracając wartość „*out*” preferuj użycie `return` ponad zwracanie jej przez referencję za pomocą parametrów** [F.20]
  - „wartość zwracana” $\neq$ „zwracanie przez wartość” – np. w przypadku metod-„getterów” możesz spokojnie zwracać typ `const T&`
  - Wartość zwracana z użyciem `return` dokumentuje się sama jako „wyłącznie wyjściowa”, a użycie parametru referencyjnego `X&` jest niejednoznaczne oraz podatne na niewłaściwe użycie danych. Duże obiekty są zoptymalizowane pod kątem niejawnych operacji przeniesienia – więc je również można zwracać z użyciem `return` bez straty dla wydajności.
  - Dla tzw. typów *non-value* (np. typów w hierarchii dziedziczenia) zwracaj obiekt z użyciem inteligentnych wskaźników (zwł. `std::unique_ptr`).

Uwagi do ramowych wytycznych

- **Chcąc zwrócić kilka wartości „out” preferuj zwracanie wartości struktury lub krotki.** [F.21]
  Wartość zwracana poprzez `return` dokumentuje się sama jako „wyłącznie wyjściowa”. Jeśli zwracane wartości są ze sobą powiązane znaczeniowo, zastosuj nazwany typ strukturalny. W przeciwnym razie, zastosuj anonimową krotkę (`std::tuple`) i rozpakowuj ją (`std::tie()`) w miejscu wywołania rozpatrywanej funkcji.
- **Preferuj `X*` ponad `X&`, jeśli „brak argumentu” jest poprawną możliwością.** [F.60]
  Wskaźnik (`X*`) posiada specjalny „niepoprawny” stan (`nullptr`), za to referencja (`X&`) – nie (nie ma „pustej referencji”).

  Czy w Twoim problemie występuje taki „niepoprawny” stan?
  - TAK $\Rightarrow$ zastosuj wskaźnik
  - NIE $\Rightarrow$ zastosuj referencję

    (bo ma prostszy zapis i potencjalnie lepsze optymalizacje kodu)

Uwagi do ramowych wytycznych

- **Zgodnie z konwencjami funkcje „zawłaszczające” argument powinny robić to zawsze, a nie w zależności od wybranej ścieżki sterowania.** Dla niektórych typów danych użycie operacji przeniesienia zmienia stan obiektu z którego zasoby zostały przeniesione – programista zwykle oczekuje takiego jego stanu po powrocie z funkcji.