РЕФЛЕКСИЯ, ПЛЕЙСХОЛДЕР, ПАТТЕРНЫ В C++ 26

Илья Казаков

Всем привет! Я Илья, разрабатываю систему хранения данных TATLIN.UNIFIED в YADRO и пишу на «плюсах». Постоянно слежу за обновлениями в языке и подмечаю, что может пригодиться в работе. Хочу поделиться с вами находками по одной из самых интригующих тем — стандарту C++ 26, разработка которого стартовала в феврале прошлого года.

Рефлексия

На мой взгляд, главное нововведение в C++ 26 — это статическая рефлексия. Сколько библиотек имитировали ее, сколько костылей написали инженеры — и вот, наконец-то, рефлексия появится в новом стандарте. Функция принесет с собой новые операторы:

- Оператор рефлексии "^",
- Оператор [::], или, как его называют коллеги, «оператор аккордеон».

Работа с операторами должна проходить в контексте constexpr, иначе фича не была бы «статической». С помощью оператора "^" мы получим новый тип std::meta::info, а чтобы вернуть «рефлексивный» тип обратно в реальный мир, будем пользоваться «аккордеоном».

Вот простой пример, как объявить переменную типа int, сначала получив информацию о типе int и вернув его обратно:

```
[:^int:] i = 0;
```

Появится нормальная библиотека сериализации и десериализации, для которой не придется объявлять поля структур макросами, наследоваться от очередного базового класса или придумывать еще миллион костылей. А раз появится SerDes, то и адекватный ORM подоспеет.

Больше материалов про C++ от меня:

- Что изучить про алгоритмы и структуры данных разработчикам на C++
- Решаем задачу асинхронного ввода-вывода с библиотекой Asio
- C++ 23 глазами практикующего системного программиста

Pattern matching

Pattern matching уже существует в языках Rust, Kotlin, Python и Haskell. Пора бы добавить его и в C++. Шансы увидеть эту фичу в C++ 26 очень высоки — добавится новый оператор match, с помощью которого мы и будем перечислять шаблоны для сопоставления:

```
expr match {
    pattern => expr;
    ...
}
```

Я привел не весь новый синтаксис, а только новый оператор. Необычно, что его сделали инфиксным. К примеру, если мы хотим использовать сопоставленное выражение, то придется использовать новое ключевое слово let:

```
(42) match {
    let x => std::cout << "Got: " << x << std::endl;
}
```

Плейсхолдер _

Вишенка на торте нового стандарта — нижнее подчеркивание теперь несет особый смысл, и эта фича точно будет в C++ 26.

Попытайтесь вспомнить, что вы делаете, когда хотите декомпозировать std::pair с помощью structured binding, но второе значение вам не нужно? Предположу, что вы, как и я, называете второй параметр нижним подчеркиванием, вот так:

```
auto [val, _] = std::pair{1, 2};
```

А что будет, если в этой же области видимости мы попробуем сделать так еще раз?

```
auto [val1, _] = std::pair{1,2};
auto [val2, _] = std::pair{3,4};
```

Если ваш компилятор еще ничего не знает про C++26, вас попросят так больше не делать. И, конечно, код не скомпилируется. К чему это все? А к тому, что теперь объявление переменной с именем _ называется name-independent declaration. И можно объявлять такую переменную столько, сколько захочется.