6A-07

C++14 における名前付き引数の実装

鈴木　遼†　　長　幾朗†

早稲田大学†

1. はじめに

```
Date(11, 12); // 位置によって規定される引数
Date(day = 11, month = 12); // 名前付き引数
```

名前付き引数（named parameter）は関数呼び出しの記述において、引数をその順序ではなく名前によって対応させるプログラミング言語機能である。Ruby や Python, Swift など複数の主要な言語で採用されている。キーワード引数（keyword arguments）と呼ばれることもある。

名前付き引数の利点は以下のとおりである。

(1) 関数宣言を参照することなく実引数の意味を読み取れ、コードの可読性が向上する
(2) 実引数を特定の順序で記述する必要がなく、順序の誤りに起因するバグを回避できる
(3) 複数のデフォルト引数が存在する場合、引数の順序によらず、任意のデフォルト引数を用いることができる

C++ の現行規格 C++14[1]において、名前付き引数は言語機能レベルでサポートされていない。過去に導入に向けた提案がなされたが、解決困難な複数の問題を理由に却下され[2]、当面実装される見通しはない。

一方、C++ で名前付き引数に相当する機能をユーザが模擬するために、メソッドチェーンや演算子のオーバーロードを使った手法が考案されてきた。本論文では、既存のプロジェクトに組み込みやすく、柔軟性とパフォーマンスに優れた C++ 名前付き引数ライブラリの実装を提案し、その使い勝手を従来の模擬実装と比較する。

2. 既存手法

Named Parameter Idiom と呼ばれるテクニックは、複数のパラメータをメンバ変数にまとめたクラスを定義する。メソッドチェーンにより個々の値を設定させるインタフェースを提供し、関数はそのクラスを引数として受け取る。見た目は他の言語の名前付き引数とは異なる。C++14 からの constexpr の制限緩和により引数を定数式のまま関数に渡すことが可能になった。

Boost Parameter ライブラリ[3]は事前に名前ごとに型を割り当てる。実引数の記述時に代入演算子のオーバーロードにより、値を特殊なラップ型に変換し、関数内で対応する値を得る方法で実装される。Boost ライブラリへの依存が発生することと、独特な記述方法に対して多くの C++ 開発者が不慣れである欠点がある。徹底的な調査ではないが、著名なオープンソースの C++ ライブラリでの採用事例は見つからない。

3. 提案手法

提案手法は Boost の方針を 82 LoC の短いコードに簡略化しつつ、`operator()`によるオブジェクトの構築と constexpr に対応させた。完全な実装はオープンソースで公開されている(https://github.com/Reputeless/NamedParameter)。

ユーザコードでは、まず名前付き引数で使用する名前を、マクロを用いて以下のとおり定義する。名前空間内での定義も可能である。

```
NP_MAKE_NAMED_PARAMETER(day);
```

このコードはマクロ展開により、定数 `day` とエイリアステンプレート `day_` を生成する。

```
constexpr auto day =
np::NamedParameterHelper<class day_tag>{};

template <class Type>
using day_ = np::NamedParameterHelper
<class day_tag>::named_argument_type<Type>;
```

`NamedParameterHelper<day_tag>` 型のオブジェクト `day` はメンバ関数に `operator=` 及び `operator()` を持ち、`day=11`, `day(11)` のように与えられた値を `NamedParameter<day_tag, int>(11)` という値にラップして返す。このとき `day` はコンパイル時定数であり、見かけと違って `day` の状態は一切変化しない。

```
// result は NamedParameter<day_tad, int> 型
auto result = (day = 11);
```

名前付き引数が使われる関数側では、仮引数の型に `day_<Type>` を用いる。`Type` には実際に渡してほしい型を指定する。`day_<int>`は `NamedParameter<day_tag, int>` 型のエイリアスであり、関数呼び出し時に名前への代入式

Implementing Named Parameters in C++14
Ryo Suzuki, Waseda University
Ikuro Choh, Waseda University

で生成される NamedParameter 型に対応する。
仮引数の実際の値へは std::optional と同等のインタフェースを用いてアクセスする。

```
void Date(day_<int> d, month_<int> m) {
  std::cout << d.value() << '-' << *m;
}
```

4. 評価

それぞれの手法の使い勝手を 15 項目の観点から評価した（**表 1**）。提案手法に対しての評価を以下にまとめた。

デフォルト引数：代入式自体をデフォルトの値にすることで対応できる。
名前の衝突の防止：引数の名前を名前空間に定義することで衝突を回避できる。
仮引数 const 修飾：引数ごとに設定できる。
constexpr サポート：名前付き引数を使用する前にコンパイル時定数式であったコードは、そのままコンパイル時定数式として機能する。
出力パラメータ：ポインタと参照による出力パラメータに対応する。参照は実引数の記述時に std::ref() でラップして明示する必要がある。
任意の引数順序：Parameter pack を用いるか、n 個の引数に対する n! 個すべてのオーバーロードパターンについて定義を提供する必要がある。前者はデフォルト引数を使用できない。
エラーメッセージ：引数の不一致に対して、理解しやすいコンパイルエラーが出力される。
名前の省略：名前の省略は、名前なし引数のオーバーロードを定義しない限りできない。。
オブジェクトの構築：1 つの引数に複数のコンストラクタ引数を渡して値を構築できる。
入力支援との相性：関数呼び出しの記述時、IDE の支援により、必要な実引数の型が示される。
名前定義の簡単さ：1 行のマクロで定義できる。
関数定義の簡単さ：それぞれの仮引数を名前に応じた型に置き換えるだけである。
仮引数へのアクセスの記述性：C++標準の std::optional にならったインタフェースである。
実行時性能：gcc, clang において最適化が有効なコードでオーバーヘッドは生じなかった。
ヘッダファイル：82 LoC の軽量なシングルヘッダ実装である。

5. まとめと今後の課題

本論文では、C++14 プログラムで名前付き引数を模擬するライブラリを提案し、その特徴を説明した。従来手法と比較して簡潔な実装で、C++開発者にとって親しみやすい文法で導入できる。

表 1　各手法の使い勝手の比較（優◎～劣×）

	Idiom	Boost	提案手法
デフォルト引数	◎	◎	◎
名前の衝突の防止	◎	◎	◎
仮引数 const 修飾	△個別不可	○参照のみ	◎
constexpr サポート	◎	×	◎
出力パラメータ	◎	◎	◎
任意の引数順序	◎	◎	△全列挙
エラーメッセージ	◎易しい	×難解	○易しい
名前の省略	×	○先頭のみ	△全列挙
オブジェクトの構築	△要実装	×	◎
入力支援との相性	◎	×型が不明	○
名前定義の簡単さ	×大量のコード	◎	◎
仮引数記述の簡単さ	◎	×独特な構文	○
仮引数へのアクセスの記述性	○	◎	○
実行時性能	◎	○	◎
ヘッダファイル	◎不要	×複雑	○簡潔

既存のプロジェクトに組み込む際に必要な変更も少なく、エラーメッセージや IDE の入力支援を効果的に活用できる。引数としての主要な言語機能をサポートし、引数順序の任意性が重要でないケースにおいては最も有効な手法である。
今後さらに使用を簡潔にするために、参照の出力パラメータを使用する際の明示的な記述や、デフォルト引数の記述の冗長さ、任意の引数順序のサポートを改善したい。
C++14 の後継規格（通称 C++17）[3] が 2017 年に策定される見込みである。constexpr に対応した std::addressof[4]と、タプルの要素をコンストラクタ引数にしてオブジェクトを構築する std::make_from_tuple [5]が標準ライブラリに採用されることで、提案ライブラリの detail 名前空間に記述された 22 LoC にわたる全ての代替実装コードが削除でき、よりコンパクトなコードが達成される。

参考文献

[1] ISO/IEC 14882:2014 Information technology Programming languages C++, 2014.
[2] Ville Voutilainen, C++ Standard Evolution ClosedIssues List (Revision R12) http://cplusplus.github.io/EWG/ewg-closed.html#150 (2015)
[3] Boost Parameter Library http://www.boost.org/doc/libs/1_63_0/libs/parameter/doc/html/index.html (2016)
[4] ISO/IEC, Working Draft, Standard for ProgrammingLanguage C++, http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/n4606.pdf (2016)
[5] Pablo Halpern, make_from_tuple: apply forconstruction, http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0209r1.pdf (2016)
[6] Marshall Clow, C++ Standard Library Issues Resolved Directly In Jacksonville http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0304r0.html (2016)