if constexpr文は
テンプレート世界のラムダ式である

高橋 晶 (Akira Takahashi)
faithandbrave@gmail.com
Preferred Networks, Inc.
2024/04/19 (金) C++ MIX #10

自己紹介

- Preferred Networks社で、スーパーコンピュータMN-Coreのソフトウェアを作っています
  - エミュレータとかアセンブラとかの低レイヤーなものを作ってます
- 日本のC++標準化委員会に一時期参加していました
- C++の日本語リファレンスサイトcpprefjpを作っています
- 著書
  - 『C++テンプレートテクニック』
  - 『C++ポケットリファレンス』
  - 『プログラミングの魔導書』
- 今年はC++の入門サイトを作りたい

C++23対応のC++ポケリ完成しました

- 2023年にC++11版を出版して、改訂を11年続けています
- 「やりたいこと」から「どうやって」を調べる逆引きリファレンス
  - cpprefjpとは役割が異なる
- 章立てや説明の粒度は言語・著者ごとにちがう
- C++ポケリは、表紙のカジュアルさとちがって、書きすぎなくらい説明を書いている

情報発信をしよう

- 伝承されない技術は廃れていく
- 本を書く人が増えてほしい
- 本で一番たいへんなのは、書き上げること
  - 途中で投げ出さず、最後まで作り上げるのが一番だいじ
- 単発記事とちがって、本は
  - 一貫性ある方針のもと、
  - 体系的な解説を提供し、
  - レビューに多くの時間をかける (出版後に修正ができない)ことが大きな価値
- 出版社との橋渡しやレビューなど協力します

C++17で入ったif constexpr文のお話です

- if constexpr文は、コンパイル時条件による条件分岐です
  - 該当しなかった方の分岐コードは、バイナリに含まれません

```
if constexpr (std::is_same_v<T, int>) {
  // intに関する処理…
}
else {
  // それ以外の型の処理…
}
```

ところで、テンプレート使ってますか？

- 普段のアプリケーション開発でテンプレートは使っていますか？
- C++03の頃は、テンプレートを使うと「黒魔術だ」とか、「無駄に複雑なことをしてる」とか言われたりもしましたが
- C++11からはだいぶ使いやすくなりました
- 私はライブラリだけでなく、アプリケーション開発でもテンプレートは普段から使っています

テンプレートとは

- テンプレートは、パラメタライズド・タイプとか言われたりするもので、型をパラメータ化する仕組みです
- 型によらない処理を共通化できます
- C++23現在はいろんなものをテンプレートにできます
  - 関数テンプレート
  - クラステンプレート
  - 型の別名テンプレート (C++11)
  - 変数テンプレート (C++14)

ラムダ式とif constexprの特徴

- ラムダ式
  - その場 (関数ローカル) でちょっとした「関数」を定義できる
- if constexpr文
  - その場で「特定の型に対するちょっとした処理」を挟み込める

挟み込みたいちょっとした処理

- たとえば前処理

```
template <class T>
void f() {
  if constexpr (typeid(T) == typeid(int)) {
    // …前処理…
  }

  // …共通の処理…
}
```

- これをオーバーロードでやろうとすると
  - 関数内の変数を渡したりがめんどう
  - コードが散らばる

オーバーロードのめんどうさを考える

```
void f_impl(int x, double a, std::string b) {
  // …intに対する処理…
}

template <class T>
void f_impl(T, double, std::string) {
  // …それ以外の型ではなにもしない…
}
```

- 特定の型以外に対する空の関数を定義する必要がある
- 引数渡しのコスト (ムーブとかdeclvalとか) をまじめに考える必要がある

例：ほとんど同じ列挙子をもつ列挙型のハンドリング

```
enum class A { a, b };
enum class B { a, b, c };

template <class E>
string enum_to_string(E e) {
  if constexpr (requires{ E::c; }) {
    if (e == E::c) return "c";
  }
  switch (e) { …a, bの文字列化… }
}
```

- 文字列化だけでなく、列挙子を型に変換・対応させる場面でも使える

だいじなこと

- if constexprを挟み込むのは、「ちょっとした」処理であるということ
- ラムダ式と同じく、大きくなってきたら関数を分けることを考えよう

まとめ

- if constexpr文は、
  - 関数内で特定の型に対するちょっとした処理を挟み込めて便利
- 関数オーバーロードと違って、
  - 変数を持ち運ばないで済む
  - 対象外の型に対する空の関数を定義しないで済む

if constexprのその他の話

- C++規格的には「constexpr if statement」という名称で「if constexpr (条件式)」が入っている
  - 構文の都合
  - constexpr if (条件式) だとelseの方もconstexpr elseになってしまう
- ここでは機能名としてif constexprを言ってます
- D言語でstatic if文というのがあり、そちらはスコープを生成しないので便利
  - 型の定義を分岐したり、条件ごとに共通の変数を作れたり…
  - if constexpr (条件式) { int x = …; } else { T x = …; }
    f(x);