C++14

constexprの制限緩和

高橋 晶(Akira Takahashi)
faithandbrave@longgate.co.jp

2013/10/26(土) C++14規格レビュー勉強会

はじめに

- この発表は、C++14のコア言語に導入される予定の「constexprの制限緩和(Relaxing constraints on constexpr)」に関するレビュー資料です。

- 提案文書：

  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3597.html
  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3598.html
  http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3652.html

概要

- C++は直交性を重視して設計されていて、直接関係ない機能同士を組み合わせて使える。しかし、constexprは他の機能(インスタンス、forループ、変数書き換え、例外等)とうまく組み合わせられない。
  これらの制限を回避するために表現力を犠牲にしなければならず、プログラマをイライラさせていた。

- constexpr関数、constexprメンバ関数、暗黙のconstといった制限を緩和する。

これまでconstexpr関数で可能だった操作

- ヌル文 { }
- static_assert宣言
- typedef宣言、エイリアス宣言(クラスやenumの定義は含まない)
- using宣言
- usingディレクティブ
- 唯一のreturn文

条件分岐にはif文の代わりに条件演算子かboolを返すconstexpr関数、
ループにはfor／while／do-whileの代わりに再帰が使われてきた。

constexpr関数の制限緩和の概要

- 変数宣言の許可
- if文とswitch文の許可
- 全てのループ文の許可
  - for、範囲for文、while、do-while
- 変数書き換えの許可
- 戻り値型(リテラル型)として、`void`を許可
- 追加として、`constexpr`非静的メンバ関数の暗黙の`const`修飾を削除

変数宣言の許可

```
constexpr int f()
{
    int result = 0; // OK
                    // 関数f()自体がconstexprであるため、
                    // 変数resultはconstexprである必要はない。
    return result;
}
```

ただし、

- static or thread_local記憶域の変数宣言は許可されない。
- 未初期化変数の宣言は許可されない。

if文とswitch文の許可

```
constexpr int abs(int x)
{
    if (x < 0) // OK
        return -x;
    return x;
}
```

```
enum class Weekday { Sun, Mon, Tue, };
constexpr Weekday intToWeekday(int n)
{
    switch (n) {
        case 0: return Weekday::Sun;
        case 1: return Weekday::Mon;
        case 2: return Weekday::Tue;
    }
    throw std::out_of_range("n is out of week");
}
```

ただし、goto文は許可されない

全てのループ文の許可

```
constexpr int f()
{
    int x = 0;

    // OK : for文
    for (int i = 0; i < 5; ++i) { x += i + 1; }

    // OK : 範囲for文
    int ar[] = {6, 7, 8};
    for (const int& i : ar) { x += i; }

    // OK : while文
    while (true) { x += 9; break; }

    // OK : do-while文
    do { x += 10; } while (false);

    return x;
}
```

変数書き換えの許可

```
constexpr int square(int x)
{
    x *= x; // OK : 変数は書き換えてもよい
    return x;
}
```

変数書き換えの許可

```
struct X {
    int x;
    constexpr X(int x)
        : x(x) {}

    constexpr int square()
    {
        x *= x; // OK : メンバ変数も書き換えられる
        return x;
    }
};

constexpr int square(int n)
{
    X x(n);
    return x.square();
}
```

戻り値型としてvoidを許可

constexprとして扱える型分類である「リテラル型(literal type)」に
voidが追加される。
これを使用して、参照パラメータを書き換えて返すスタイルが可能になる。

```
constexpr void square(int& x)
{
    x *= x;
}

constexpr int f(int x)
{
    square(x);
    return x;
}
```

constexpr非静的メンバ関数の暗黙のconst修飾を削除

C++11では、constexpr非静的メンバ関数は、暗黙でconstが付いていた。

```
struct X {
    constexpr int f();        // これは以下と同じ
//  constexpr int f() const;
};
```

C++14ではこの仕様が削除され、const or mutableを明示的に指定することになった。

※既存コードの互換性は壊れない。

C++14のconstexpr関数でできないこと

- 仮想関数定義
- インラインアセンブラ(asm)の定義
- goto文
- tryブロック
  - 例外は投げられるので、投げっぱなしのみ可能
- 非リテラル型の変数定義(vectorやstring等、フリーストアを必要とするもの)
- static or thread_local記憶域の変数定義
- その他、入出力(ファイル、ネットワーク)等。

所感

- constexpr関数は実行時でも呼べるという特性があります。そのため、実行時で可能な操作はできる限り、コンパイル時にもできるべきだと思います。
- 今回の提案が取り入れられることにより、コンパイル時のコードと実行時のコードに、より高い互換性がもたらされるでしょう。
- 今回のレビューには含めていませんが、標準ライブラリのconstexpr対応も着々と進められています。