C++26 アップデート 2025-03

高橋 晶 (Akira Takahashi)
faithandbrave@gmail.com
Preferred Networks, Inc.
2025/04/25 (金) C++ MIX #14

C++26 2025-03

- 今回は、2025-03 mailingとしてC++26 Working Draftに入った変更を紹介していきます
- 今回でC++26仕様は一旦できたらしい
  - このあとは仕様バグの修正や、各国からのコメントへの対応などがある

constexpr unionによる遅延初期化

```
template <class T, int N>
class FixedVector {
  union U {
    constexpr U() { }
    constexpr ~U() { }
    T storage[N];
  };
  size_t size = 0;

  …
};
```

- 配列要素の初期化を遅延させるための手法としてunionが使われていたが、constexprをつけるとコンパイラによって動かなかったりした
- C++26では明確に許可される

タイムトラベル最適化を防止するstd::observable()

```
static void bad(const char *msg) {
  std::fputs(msg, stderr);
#ifdef DIE
  std::abort();
#endif
}

void inc(int *p) {
  if(!p) bad("Null! n");
  ++*p; // !p時にここで未定義動作
}
```

- DIEを定義しない場合、inc(nullptr)は未定義動作になることが保証されるため、**bad()をそもそも呼び出さない**タイムトラベル最適化が行われる
- C++26ではstd::observable()という関数でチェックポイントを設定すると、時間分割されてチェックポイントを跨いだタイムトラベル最適化が防止される
- C++26の標準出力関係にはチェックポイントが設定される

契約プログラミング

```
int f(const int x)
  pre (x != 1)
  post (r: r == x && r != 2) {
  contract_assert(x != 3);
  return x;
}

void clear()
  post (empty()) { … }
```

- 3つの契約機能
  - 関数の事前条件pre
  - 関数の事後条件post
  - 契約アサーションcontract_assert
- postは戻り値に変数名をつけて使用してもいいし、使用しなくてもいい
- pre/postは文脈依存キーワード(変数名とかに使える)
- コンパイラに契約モードの設定が追加される想定

memory_order::consume関係を非推奨化

```
std::atomic<T> x = …;

int a = x.load(memory_order::consume); // 非推奨
int b = a + 1;
int c = b + 1;
```

- consumeメモリオーダーは、並行プログラムにおいて、
  読み込んだ**値に依存した操作の実行順序**を保証するもの
- 実際はconsume相当の順序保証はどのメモリオーダーにも付くのと、
  コンパイラがconsume特化の実装をしなかったので非推奨化する
- 関連してstd::kill_dependency()と[[carries_dependency]]も非推奨化

コンセプトと変数テンプレートの テンプレートテンプレートパラメータ対応

```
template<
  template <typename T> concept C,
  template <typename T> auto V
>
struct S {
  concept D = C<int>;
  constexpr auto X = V<int>;
};

template <typename T>
concept Concept = true;

template <typename T>
constexpr auto Var = 42;

S<Concept, Var> s;
```

- コンセプトと変数テンプレートに、**テンプレートテンプレートパラメータ**が使えるようになる
- テンプレートテンプレートパラメータは、テンプレート引数をあとで指定する機能

トリビアルな再配置

```
template <class T>
class optional
    trivially_relocatable_if_eligible
    replaceable_if_eligible {
  union {
    T d_object;
  };
  bool d_engaged{false};
  …
};
```

- trivially relocatable（トリビアルな再配置）は、memcpyやビットコピーで再配置ができる性質
- 型にこの性質を与えることで、ムーブ元オブジェクトのデストラクタ呼び出しなどを省略できる
- そのために文脈依存キーワードが2つ追加される（それと対応した型特性も）

#embed : プリプロセス時ファイル読み込み

```
// 無限サイズのファイル"dev/urandom"
// から最大4バイトを読む
const unsigned char random[] = {
  #embed "/dev/urandom" limit(4)
};

constexpr int seed = 0;
for (int i = 0; i < 4; i++) {
  seed |= random[i] << (i * 8);
}
constexpr std::mt19937 gen{seed};
```

- #includeのように使える
  ファイル読み込み機能
- 上限のバイト数や、先頭列、末尾列、空であった場合のデータなどを指定できる

vector / string以外のコンテナをconstexpr対応

```
constexpr R f() {
  std::map<std::string, int> m = {
    …
  };

  if (m.contains("key")) {
    …
  }
  …
}
```

- C++20でstd::vectorとstd::stringがconstexprで使えるようになった
- C++26では、ほかの標準コンテナもすべてconstexpr対応する

hive : 要素のメモリ位置が安定するシーケンスコンテナ

```
std::hive<int> ls = {1, 2, 3};

// 要素を追加。
// {4, 5}の新たなメモリブロックが確保される
// {1, 2, 3}のメモリ位置は変わらない
ls.insert({4, 5});

// 要素3を削除。
// {4, 5}のメモリ位置は変わらない
ls.erase(std::next(ls.begin(), 2));
```

- 双方向Rangeのシーケンスコンテナstd::hiveが追加される
- 要素の追加・削除をしても、メモリ位置は変わらない
- ゲーム、HPC、物理シミュレーションなど幅広い分野で使われてきたデータ構造

indirectとpolymorphic：値の意味論をもつ動的確保オブジェクト

```
class ForwardListNode {
  int data_;
  indirect<ForwardListNode> next_;
};

class ForwardList {
  indirect<ForwardListNode> head_;
};
```

```
class Composite {
  indirect<A> a_;
  polymorphic<X> x_;
public:
  template <class Derived>
  Composite(const A& a, const Derived& x) :
    a_{a},
    x_{std::in_place_type<Derived>, x} {}
};
```

- indirectとpolymorphicは、動的確保したオブジェクトのポインタに、値の意味論をもたせるもの
- コピー時にポインタのコピーではなく、参照先の値をコピー（ディープコピー）する
- pImplイディオムもunique_ptrの変わりにindirectを使える

今回は以上です！

- C++26のアップデートや、
  それぞれの機能を掘り下げる発表は
  今後もやっていきます