Chapter-4 c++による大規模開発～C++標準テンプレートライブラリの利用(上級)～

この節では次のプログラムを使用します。次のURLからプログラムをダウンロードしておいてください。

4.1 std::array

C++11から固定長配列をセキュアに扱うことができるstd::arrayというテンプレートクラスが導入されました。このテンプレートクラスの導入により、旧来の固定長配列を使用する理由はほぼなくなりました。std::arrayクラスは旧来の固定長配列の大きな問題であった、メモリ破壊のしやすさを解消してくれます。次のコードを見てください。

```
//要素数10の配列を定義する。
int hoge[10];

int main()
{
    //メモリを壊している！
    hoge[10] = 10;
}
```

hogeの添え字に使える値は0～9の範囲なのに、10番目の要素にアクセスしてしまっています。旧来の配列はこのようにメモリを壊してもプログラムは動き続けてしまい、予期せぬ挙動を起こしてしまうことが多々ありました。このような小さなプログラムであれば、メモリ破壊に気づくことは容易ですが、大規模な開発になってくると、発見することが非常に難しい厄介なバグとなります。 C++11から導入されたstd::arrayクラスは、この問題を解決してくれます。次のコードはstd::arrayクラスを利用して固定長配列を使用しているコードです。

```
// std::arrayを利用するためにはarrayをインクルードする必要がある。
#include <array>
// テンプレート引数の第一引数は配列の要素の型、第二引数は要素数、
std::array<int, 10 > hoge;
int main()
{
    // ここでクラッシュしてくれる。
    hoge[10] = 10;
    return 0;
}
```

このプログラムも先ほどのものと同様に配列の範囲外にアクセスしています。しかし、大きな違いがあります。std::arrayクラスは範囲外アクセスを起こすと、プログラムをクラッシュしてくれるのです。プログラムがクラッシュすると聞くと、そっちの方が困る！と思ってしまうかもしれませんが、メモリを破壊してもそのまま動いてしまう方が、ほとんどの場合で厄介です。大規模な開発であれば、メモリを壊してしまったせいで、数時間立ってからプログラムがクラッシュしたり、全く関係のないプログラムで不具合が起きたりし

ます。メモリを壊されたら、即座にプログラムをクラッシュしてくれた方がよほどいいわけです。 また、std::arrayクラスはイテレータを返すbegin()関数やend()関数を利用することができます。また、要素数を返してくれるsize()関数など旧来の固定長配列にはなかった便利な機能が用意されています。

4.2 速度はどうなっているの？

さて、ゲームプログラマであれば最も気になるのは速度の問題です。範囲外アクセスを起こしたときにクラッシュするということは、std::arrayクラスの[]演算子は次のような実装になっているはずです。

```
template<class T>
class array{
    ・
   省略
    ・
public:
    //添え字演算子
    T operator[](int index)
    {
        if( index < size){
            //インデックスが要素数以下なら
            return p[index];
        }else{
            //範囲外アクセスなのでクラッシュさせる。
            std::abort();
        }
    }
}
```

これでは、配列の要素にアクセスするだけでif文のコストが発生してしまいます。ゲームはシビアなパフォーマンスが要求され、速いプログラムが正義です。配列にアクセスするたびにif文コストがかかるようでは、使い物にならないでしょう。確かにstd::arrayクラスは、デバッグモードではこのような実装になっています。しかしリリースビルでは、このif文は全て消え去ります。

```
template<class T>
class array{
    ・
   省略
    ・
public:
    //添え字演算子
    T& operator[](int index)
    {
    #ifdef _DEBUG
        if( index < size){
            //インデックスが要素数以下なら
            return p[index];
        }else{
            //範囲外アクセスなのでクラッシュさせる。
            std::abort();
```

```
        }
    #else
        //リリースビルドでは範囲チェックはなくなる。
        return p[index];
    #endif
    }
}
```

std::arrayはリリースビルドでは範囲チェックはなくなります。このため、std::arrayは製品版のビルドではセキュアではなくなり、旧来の固定長配列と同じになります。

4.3 【ハンズオン】MyArrayを実装してみる

では、std::arrayの理解を深めるために、std::arrayクラスを真似したMyArrayクラスを作成してみましょう。Sample_04_01を開いてください。

step-1 MyArrayクラスを作成する。

まずMyArrayクラスを作成してみましょう。main.cppに次のプログラムを入力してください。

```
// step-1 MyArrayクラスを作成する。
template <class T, int SIZE >
class MyArray {
    T value[SIZE];
public:
    T& operator[](int index)
    {
    //これは条件付きコンパイル機能。_DEBUGというシンボルが定義されている場合に#ifdef～#elseのコードがコンパイルされる。
#ifdef _DEBUG
        if (index < SIZE) {
            return value[index];
        }
        else {
            std::abort();
        }
#else
    //こっち側は_DEBUGが定義されていない場合にコンパイルされる。
        return value[index];
#endif
    }
};
```

step-2 MyArrayクラスを使ってみる。

続いて、MyArrayクラスを利用するコードを実装します。次のプログラムをmain.cppに入力してください。

```
// step-2 MyArrayクラスを使ってみる。
MyArray< int, 10 > hoge;
```

```
hoge[0] = 10;
std::cout << hoge[0];
// デバッグビルドではここでクラッシュする！
hoge[10] = 20;
```

入力出来たら、デバッグモードとリリースモードで動作を確認してください。デバッグモードではプログラムがクラッシュするはずです。

4.4 旧来の固定長配列を使う場面は？

ほとんどのケースで旧来の固定長配列を利用する場面はありません。デメリットしてしいてあげるなら、プログラムのタイプ量が増大してしまって、腱鞘炎になる確率が上がることでしょうか(しかし、これは意外と大きな理由になる)。もう一つは、std::arrayは比較的新しい機能だということです。大規模開発では、色々なスキルのプログラマが参加します。中にはstd::arrayを知らないというプログラマもいます。そのため、コードの可読性という点では下がってしまう琴になります。これはプロジェクトによっては使用を禁止にする理由になりえます。

4.5 実習課題

Question_04_01の固定長配列を用いている個所をstd::arrayを利用するように実装を改造しなさい。 std::arrayの使い方は次のURLで調べてください。

std::arrayのヘルプ

評価テスト

次の評価テストを行いなさい。
評価テストへジャンプ