Chapter 2 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(基本)～

2.1 C++標準テンプレートライブラリとは

C++標準テンプレートライブラリとは、C++のプログラミングを楽にするための強力なライブラリです。その中でもこのチャプターでは以下の3つについて取り上げます。

1. vector(可変長配列)
2. list(双方向リスト)
3. map(連想配列)

これらのクラスは、STLと呼ばれる標準テンプレートライブラリの一部であり、ゲーム開発などで非常に有用です。このクラスは基本的に同様の関数を備えているため、1つのクラスを理解すれば、他のクラスの使用方法も簡単に習得できます。

それでは、これらのクラスについて学んでいきましょう。

Chapter 2.2 可変長配列「vector」

2.2 可変長配列「vector」

vectorは要素数を動的に変化させることができる配列です。「動的」というのはプログラム実行中ということです。
これまで、皆さんが勉強した配列は、固定長配列です。固定長配列はプログラムの実行中にサイズを変えることはできません。
次のプログラムを読んでみてください。

```
int hoge[10];
 ・
 省略
 ・
for( int i = 0; i < 10; i++){
    std::cout << hoge[i];
}
```

このプログラムには配列hogeが登場していますが、この配列のサイズは10で固定です。プログラム実行中に、動的にサイズを変えることはできません。
可変長配列はこのサイズをプログラム実行中に変更することができるのです。
要素を追加するにはemplace_back()関数を利用します。

2.2.1 【ハンズオン】vectorを使ってみる。

では、習うより慣れろです。さっそくvectorを使ってみましょう。Sample_02_01を立ち上げてください。 立ち上がったら、main.cppを開いてください。

step-1 vectorをインクルード

vectorを利用するためには、まずインクルードを行う必要があります。main.cppにリスト2.1のプログラムを入力してください。
[リスト2.1]

```
//step-1 vectorをインクルード
#include <vector>
```

step-2 int型の要素を記録できる可変長配列を定義する。

続いて、int型の要素を記録できる可変長配列を定義します。main.cppにリスト2.2のプログラムを入力してください。
[リスト2.2]

```
//step-2 int型の要素を記録できる可変長配列を定義する。
std::vector<int> hoge;
```

step-3 hogeに要素を追加する

続いて、hogeに要素を追加します。要素の追加はemplace_back()関数を利用します。main.cppにリスト2.3のプログラムを入力してください。
[リスト2.3]

```
//step-3 hogeに要素を追加する
hoge.emplace_back(10); //配列の末尾に10を追加する。
hoge.emplace_back(20); //配列の末尾に20を追加する。
hoge.emplace_back(30); //配列の末尾に30を追加する。
```

step-4 hogeのサイズを表示する。

さて、step-3でhogeにint型のデータを3つ追加しました。ですので、可変長配列hogeのサイズは3になっているはずです。可変長配列のサイズはsize()関数で取得することができます。
では、本当にhogeのサイズが3になっているか確認してみましょう。main.cppにリスト2.4のプログラムを入力して下さい。 [リスト2.4]

```
//step-4 hogeのサイズを表示する。
std::cout << "hogeのサイズは" << hoge.size() << "です。\n";
```

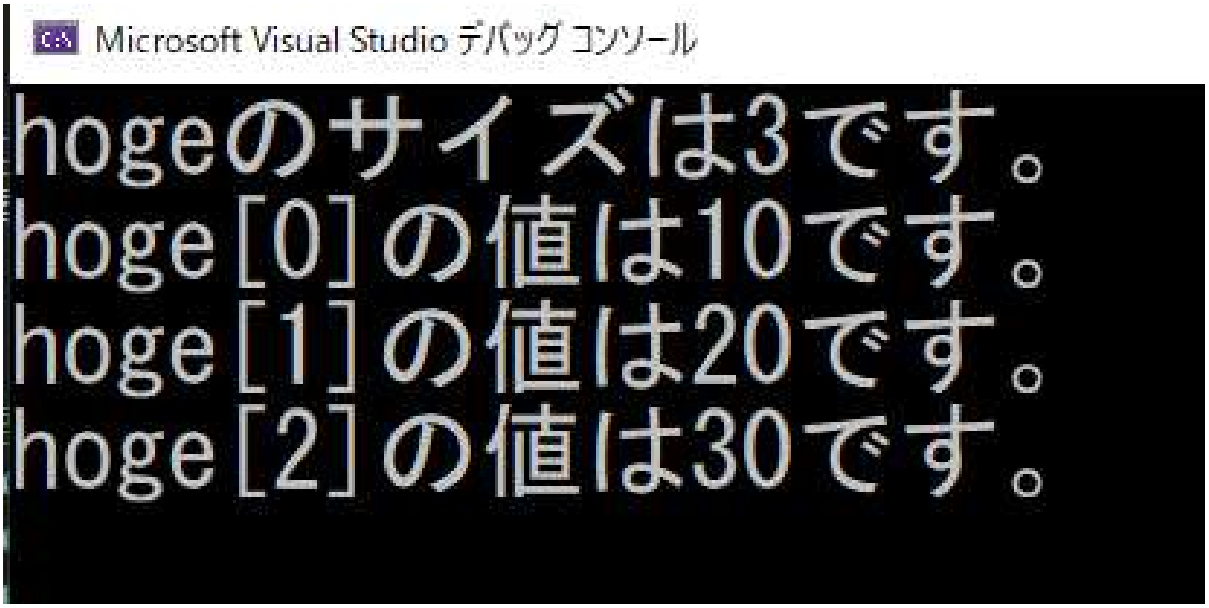
step-5 for文を使って、hogeの要素の値を表示する。

可変長配列は、固定長配列と同じように添え字演算子、[]を利用してアクセスすることができます。では、for文で回して、hogeの各要素を表示してみましょう。

main.cppにリスト2.5のプログラムを入力してください。
[リスト2.5]

```
//step-5 for文を使って、hogeの要素の値を表示する。
for (int i = 0; i < hoge.size(); i++) {
    std::cout
        << "hoge["
        << i
        << "]の値は"
        << hoge[i]
        << "です。\n";
}
```

では、ここまででー度実行してみて下さい。図2.2のように表示されていたら実装できています。
[図2.2]



step-6 さらにhogeに要素を追加する。

step-6では更にhogeに要素を追加してみましょう。main.cppにリスト2.6のプログラムを入力して下さい。
[リスト2.6]

```
//step-6 さらにhogeに要素を追加する。
hoge.emplace_back(40);
hoge.emplace_back(50);
```

step-7 再度hogeのサイズを表示する。

step-6でhogeに要素を２つ追加したので、hogeの要素数は5になっているはずです。では、再度hogeのサイズを表示するプログラムを追加しましょう。
main.cppにリスト2.7のプログラムを入力してください。
[リスト2.7]

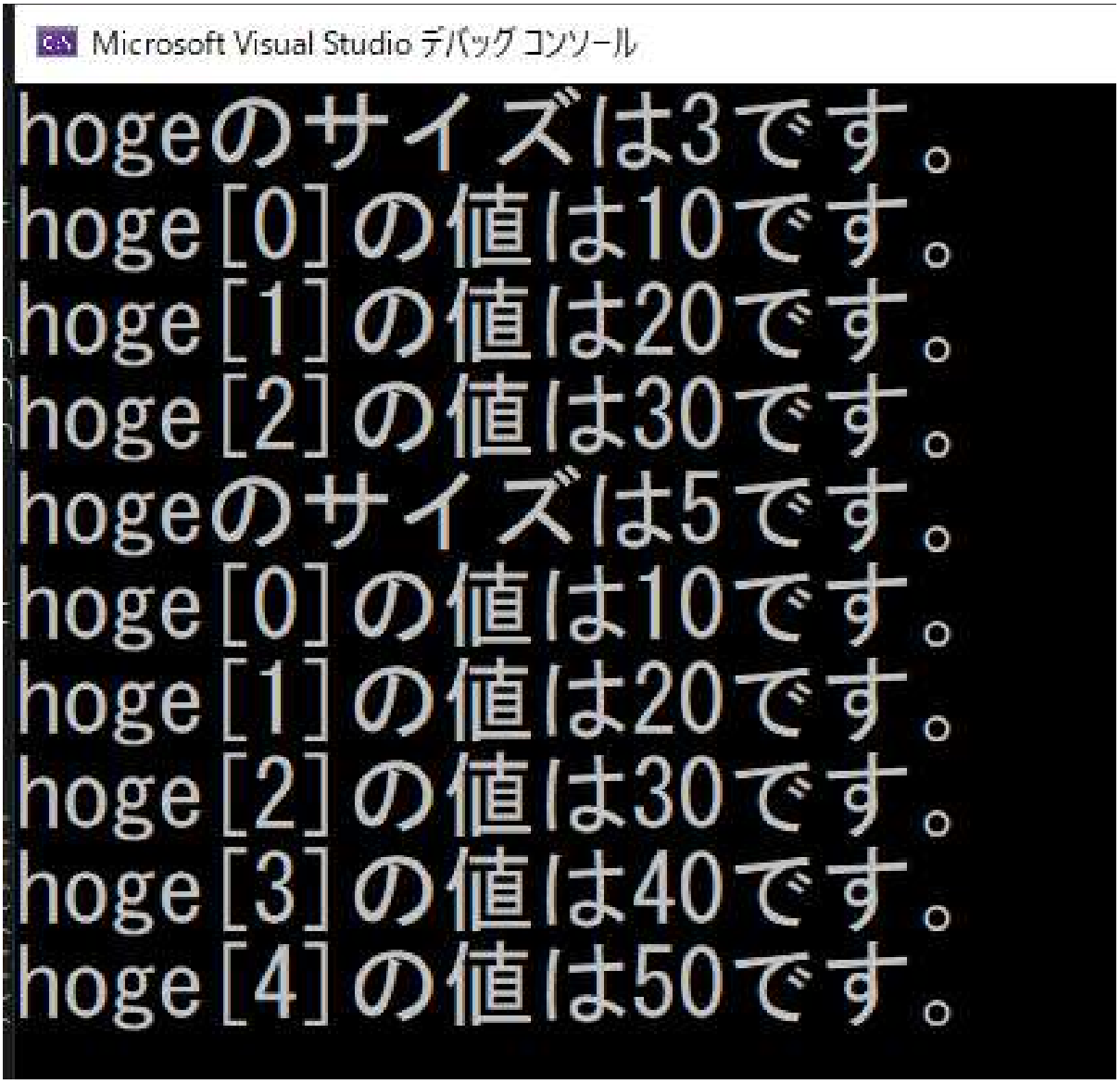
```
//step-7 再度hogeのサイズを表示する。
std::cout << "hogeのサイズは" << hoge.size() << "です。\n";
```

step-8 もう一度for文を使って、hogeの要素の値を表示する。

では、最後にもう一度for文を使ってhogeの要素の値を表示してみましょう。main.cppにリスト2.8のプログラムを入力してください。
[リスト2.8]

```
//step-8 もう一度for文を使って、hogeの要素の値を表示する。
for (int i = 0; i < hoge.size(); i++) {
    std::cout
        << "hoge["
        << i
        << "]の値は"
        << hoge[i]
        << "です。\n";
}
```

入力出来たら実行してみてください。図2.3のように表示されていたら実装できています。
[図2.3]



2.2.1 int型以外も記録できるの？

vectorクラスはテンプレートクラスとなっているため、int型以外の値も記録できます。float型を記録できる可変長配列を定義して、利用するコードを下記に示します。

```
std::vector<float> hoge;
hoge.emplace_back(10.0f);
hoge.emplace_back(20.0f);
hoge.emplace_back(30.0f);
```

vectorクラスはintやfloatのような組み込み型だけではなく、ユーザー定義のクラスでも利用できます。下記のコードはMatrixクラスのオブジェクトの値を記録しているコードです。

```
Matrix m0, m1, m2;
   ・
  省略
   ・
std::vector<Matrix> hoge;
```

```
  hoge.emplace_back( m0 );
  hoge.emplace_back( m1 );
  hoge.emplace_back( m2 );
```

2.2.2 評価テスト-3

次の評価テストを行いなさい。
評価テストへジャンプ

2.3 双方向リスト「list」

listは要素数を動的に変化させることができる双方向リストです。listは可変長配列のvectorと同様に、emplace_back()関数を利用することで、要素数を増やすことができます。次のプログラムを見てみてください。

```
std::list< int > hoge;
hoge.emplace_back( 10 );
hoge.emplace_back( 20 );
hoge.emplace_back( 30 );
```

vectorと同様にsize()関数を利用することで、要素数を取得することができます。

```
std::list< int > hoge;
hoge.emplace_back( 10 );
hoge.emplace_back( 20 );
hoge.emplace_back( 30 );

std::cout << "hogeのサイズは" << hoge.size() << "です。\n";
```

2.3.1 listとvectorの違い

さて、要素数を動的に変化させるという点では、listとvectorは全く同じもののように思えます。この節では、listとvectorの違いについていくつか見ていきましょう。

listは添え字演算子が使えない。

listはvectorと違って添え字演算子が使えません。つまり、次のようなプログラムがかけないということです。

```
std::list< int > hoge;
hoge.emplace_back( 10 );
hoge.emplace_back( 20 );
hoge.emplace_back( 30 );

// できないのでコンパイルエラーが起きる。
std::cout << hoge[2];
```

では、listを使っているときに要素のアクセスをどうするのか？というとイテレーター(反復子)を利用して、要素の先頭から順繰りアクセスしていきます。

```
std::list< int > hoge;
hoge.emplace_back( 10 );
hoge.emplace_back( 20 );
hoge.emplace_back( 30 );
```

```
//begin()関数を使って、先頭イテレータを取得する。
std::list< int >::iterator it = hoge.begin();
// 10と表示される。
std::cout << *it << "\n"
// 次の要素に進む。
it++;
// 20と表示される。
std::cout << *it << "\n"
```

イテレーターはlistが格納している要素を指し示す、ポインタのようなものです。ポインタと同じように*演算子を利用して、イテレーターが指し示している要素にアクセスすることができます。
また、イテレーターはlistだけではなくvectorや、この後勉強するmapなど、C++の標準ライブラリのコンテナクラスの全て利用できます。

> コンテナクラスとは、listやvectorのように、要素を記憶することができるC++標準ライブラリのクラスです。
> listやvector以外にも、array、deque、fowrard_list、map、multimap、unordered_map、set、multisetなどいくつか存在しています。
> どのコンテナクラスにもメリット、デメリットが存在しており、必要に応じて適切なコンテナクラスを選択することが重要になってきます。

イテレーターの典型的な利用方法として、for文での利用があげられます。

```
std::list< int > hoge;
hoge.emplace_back( 10 );
hoge.emplace_back( 20 );
hoge.emplace_back( 30 );

// for文を使って、リストの全要素にアクセスする。
for(
    std::list< int >::iterator it = hoge.begin();
    it != hoge.end(); // イテレーターが終端に到達するまで
    it++              // 次のイテレーターへ
){
    std::cout << *it << "\n";
}
```

listを使うべき場面は？

基本的にゲームにおいては、可変長配列vectorの方が多くの場面でlistより向いています。
しかし、頻繁に要素の追加と削除が発生する場合に、vectorクラスの代わりに利用することを検討すべきです。
この理由については、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(応用)～で解説します。

2.3.1 【ハンズオン】listを使ってみる。

では、習うより慣れろです。さっそくvectorを使ってみましょう。Sample_02_02を立ち上げてください。 立ち上がったら、main.cppを開いてください。

step-1 listをインクルード

listを利用するためには、まずインクルードを行う必要があります。main.cppにリスト2.1のプログラムを入力してください。
[リスト1]

```
//step-1 listをインクルード
#include <list>
```

step-2 int型の要素を記録できる双方向リストを定義する。

続いて、int型の要素を記録できる可変長配列を定義します。main.cppにリスト2のプログラムを入力してください。
[リスト2]

```
//step-2 int型の要素を記録できる双方向リストを定義する。
std::list<int> hoge;
```

step-3 hogeに要素を追加する

続いて、hogeに要素を追加します。要素の追加はemplace_back()関数を利用します。main.cppにリスト3のプログラムを入力してください。
[リスト3]

```
//step-3 hogeに要素を追加する
hoge.emplace_back(10); //配列の末尾に10を追加する。
hoge.emplace_back(20); //配列の末尾に20を追加する。
hoge.emplace_back(30); //配列の末尾に30を追加する。
```

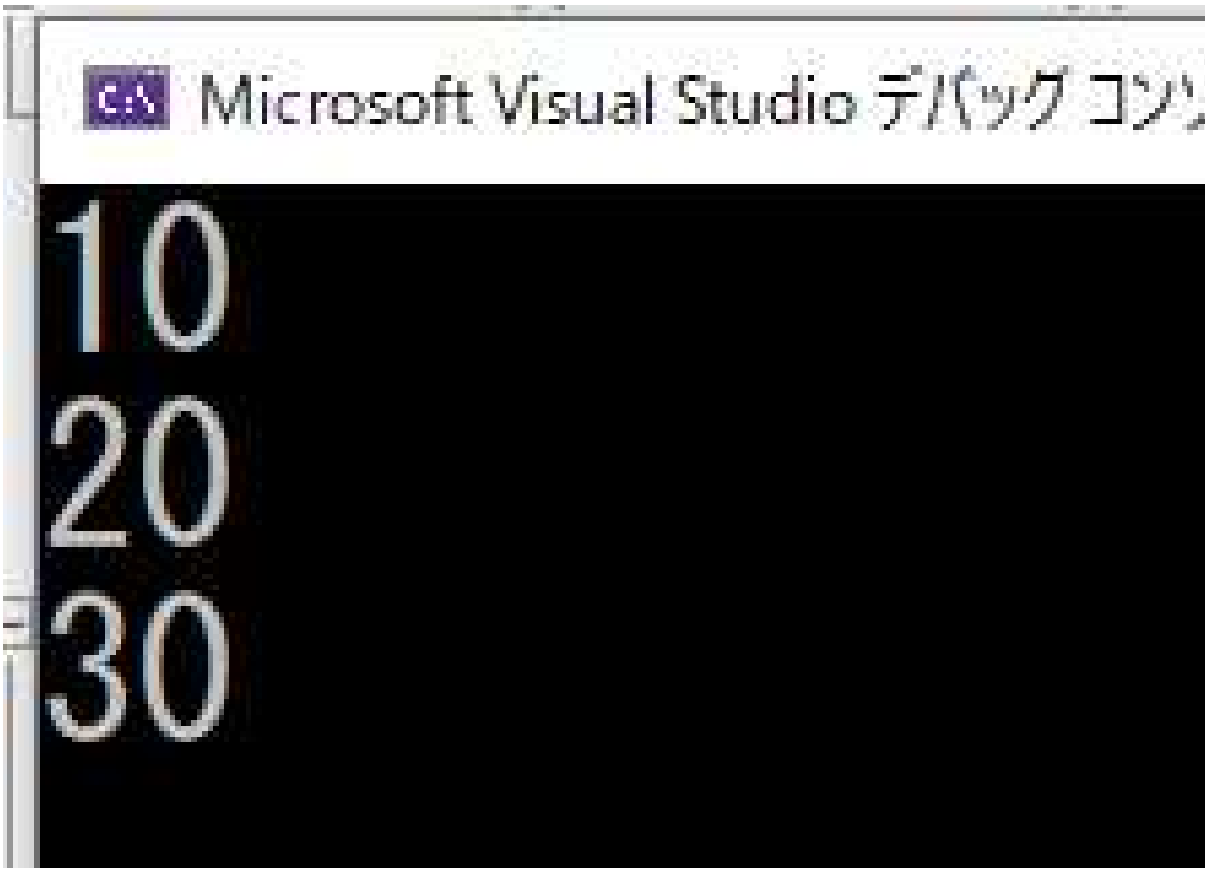
step-5 for文を使って、hogeの要素の値を表示する。

続いて、for文で回して、hogeの各要素を表示してみましょう。可変長配列と違い、添え字演算子が使えないため、イテレータを利用します。
main.cppにリスト4のプログラムを入力してください。
[リスト4]

```
// step-4 for文を使って、hogeの要素の値を表示する。
for (
    std::list<int>::iterator it = hoge.begin();
    it != hoge.end();
    it++
```

```
) {
    std::cout << *it << "\n";
}
```

では、ここまで実行してみて下さい。図2.4のように表示されていたら実装できています。
[図2.4]



2.3.2 評価テスト-4

次の評価テストを行いなさい。
評価テストへジャンプ