vectorクラス

- コンテナクラスの一種

- コンテナとは？
  STL(Standard Template Library)で定義されている、C言語の配列のように複数の値を格納できるもの

- vector, list, map, array, bitset, stack, queue 等さまざまなコンテナクラスが存在する

vectorクラス

- コンテナクラス
  - vector：動的配列（自由にサイズを変更可能な配列）
  - array ：静的配列（一旦決めたサイズは変更不可）
  - list ：リスト構造を実現するクラス
  - map ：連想配列という特殊な配列クラス
  - bitset：2進数値を容易に扱うためのクラス
  - stack ：スタック（後入れ先出し）を実現するクラス
  - queue ：キュー（先入れ先出し）を実現するクラス

vectorクラス

- vectorを使う利点
  - (例)敵クラスのインスタンス生成

```
Enemy enemy1;
Enemy enemy2;
Enemy enemy3;
```

バラバラの変数で管理するより配列化したほうがよい

```
Enemy enemy[100];
```

しかし要素数(100体)までしか対応できない…

vectorクラス

- vectorを使う利点
  - (例)敵クラスのインスタンス生成

```
Enemy enemy[100];
```

```
Enemy enemy[10000];
```

とすると、最初に10000体分のメモリを確保しようとするため、敵が数体しか登場しないときはメモリの無駄遣いになる…

vectorクラス

- vectorを使う利点
  - **動的な配列**を実現できる
    配列の要素数最初に定めずに、都度追加できる！
  - クラスのメンバ関数として、要素数をカウントしたり、
    要素をすべてクリアしたりといった機能があり、
    C言語では容易でなかったことができる！

vectorクラス

- vectorを使う利点
  - (例)敵クラスのインスタンス生成

    vectorを使うことで、必要なときに必要なぶんだけ
    配列要素を確保することができる
    また不要になれば要素を削除することもできる！

    このような配列を**動的配列**もしくは**可変長配列**という

vectorクラス

- vectorクラスのメンバ関数
  - size()　　:配列の全要素数をカウント
  - push_back()　:配列末尾に要素を付け加える
  - emplace_back():配列末尾に要素を付け加える
  - pop_back()　:配列末尾のデータを消去する
  - erase()　　:指定された場所の要素を削除する
  - insert()　　:指定された場所へ要素を追加する
  - empty()　　:配列要素が空ならtrueを返す
  - clear()　　:配列要素をすべて削除

vectorクラス

- 教科書P227~228 **Sample604**

- C++作業フォルダ内に**Sample604**フォルダを作成
```
mkdir Sample604
cd Sample604
```

- main.cppを作成
```
copy nul main.cpp
```

vectorクラス

main.cpp (Sample604)

```
#include <vector>
#include <string>
int main(){
std::vector<int> v1;
std::vector<string> v2;
   v1.push_back(1);
   v1.push_back(2);
   v1.push_back(3);
   v2.push_back("ABC");
   v2.push_back("DEF");
   for (int i = 0; i < v1.size(); i++) {
       std::cout << ”v1[" << i << "]=" << v1[i] << std::endl;
   }
   for (int i = 0; i <  2.size(); i++) {
       std::cout << “v2[" << i << "]=" << v2[i] << std::endl;
   }
   return 0;
}
```

vectorクラス

- vectorの宣言方法

- #include <vector> が必須!

- std::vector<型名> 配列名
  std::vector<型名> 配列名 { 初期値 }

  ※using namespace stdを記述している場合
  「 std:: 」は省略可能

vectorクラス

main.cpp (Sample604)

```
#include <vector>
#include <string>
using namespace std;
int main(){
vector<int> v1;
vector<string> v2;
    v1.push_back(1);
    v1.push_back(2);
    v1.push_back(3);
    v2.push_back("ABC");
    v2.push_back("DEF");
    for (int i = 0; i < v1.size(); i++) {
        cout << ”v1[" << i << "]=" << v1[i] << endl;
    }
    for (int i = 0; i <  2.size(); i++) {
        cout << “v2[" << i << "]=" << v2[i] << endl;
    }
    return 0;
}
```

vectorクラス

- vectorの末尾にデータを追加

- std::vector<int> v1

```
v1.push_back(10)
```

or

```
v1.emplace_back(31)
```

引数に同じ基本型のデータを指定すると要素数が増えていく

vectorクラス(要素の追加と要素数)

main.cpp (Sample604)

```
    v1.push_back(3);
    v2.push_back("ABC");
    v2.push_back(“DEF”);
    v1.emplace_back(4); //末尾に要素追加
    cout << ”v1の要素数:” << v1.size() << endl;
    for (int i = 0; i < v1.size(); i++) {
        cout << ”v1[" << i << "]=" << v1[i] << endl;
    }
    v2.emplace_back(“G”); //末尾に要素追加
    cout << ”v2の要素数:” << v2.size() << endl;
    for (int i = 0; i <  2.size(); i++) {
        cout << “v2[" << i << "]=" << v2[i] << endl;
    }
    return 0;
}
```

vectorクラス(要素の削除)

main.cpp (Sample604)

```
    v1.push_back(3);
    v2.push_back("ABC");
    v2.push_back(“DEF”);
    v1.pop_back();  //末尾要素の削除
    v1.emplace_back(4);
    cout << ”v1の要素数:” << v1.size() << endl;
    for (int i = 0; i < v1.size(); i++) {
        std::cout << ”v1[" << i << "]=" << v1[i] << endl;
    }
    v2.pop_back();
    v2.emplace_back(“G”);
    cout << ”v2の要素数:” << v2.size() << endl;
    for (int i = 0; i <  2.size(); i++) {
        cout << “v2[" << i << "]=" << v2[i] << endl;
    }
    return 0;
}
```

vectorクラス(挿入と削除)

- vectorの末尾だけでなく、指定した場所に要素を追加するメンバ関数

  ・insert(場所, 挿入値)
  ・erase(場所)

  が存在するが、挿入する場所、削除する場所はどちらも配列の添え字の番号ではない…

イテレータ(iterator:反復子)

- vectorといったコンテナクラス内の要素の位置を示すポインタのようなもの

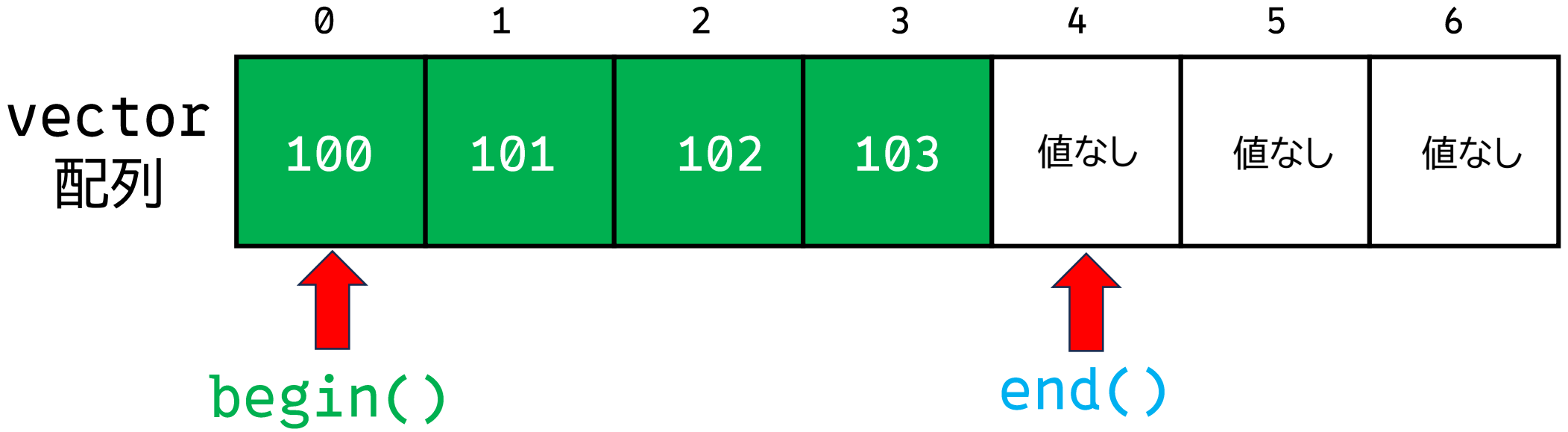
```
std::vector<型名>::iterator itr
```

```
itr = v1.begin();
```

とすると、itrは先頭要素のイテレータとなる

イテレータ(iterator:反復子)

- イテレータ ＝ インスタンス.begin()
  コンテナクラスの先頭要素の場所をイテレータに代入
- イテレータ ＝ インスタンス.end()
  コンテナクラスの最終要素のひとつ先の場所をイテレータに代入



イテレータ(iterator:反復子)

- イテレータを使うと、コンテナクラス内の指定場所への要素の追加や削除を行うことが可能

インスタンス.insert(イテレータ, 挿入値)
　イテレータが指し示す場所に値を挿入する

インスタンス.erase(イテレータ)
　イテレータの示す場所の要素を削除する

イテレータ(iterator:反復子)

main.cpp (Sample604)

```
    v1.push_back(3);
    v2.push_back("ABC");
    v2.push_back(“DEF”);
    v1.pop_back();
    v1.emplace_back(4);
    cout << ”v1の要素数:” << v1.size() << endl;

    vector<int>::iterator itrV1 = v1.begin();
    cout << ”イテレータが指す要素の値:” << *itrV1 << endl;
    v1.insert(itrV1 + 1, 11); //指定した場所（先頭から+1番目）へ要素（11）を追加
    v1.erase(itrV1 + 2);      //指定した場所（先頭から+2番目）の要素を削除

    for (int i = 0; i < v1.size(); i++) {
        cout << ”v1[" << i << "]=" << v1[i] << endl;
    }
```

イテレータ(iterator:反復子)

main.cpp (Sample604)

```
    v1.push_back(3);
    v2.push_back("ABC");
    v2.push_back("DEF");
    v1.pop_back();
```

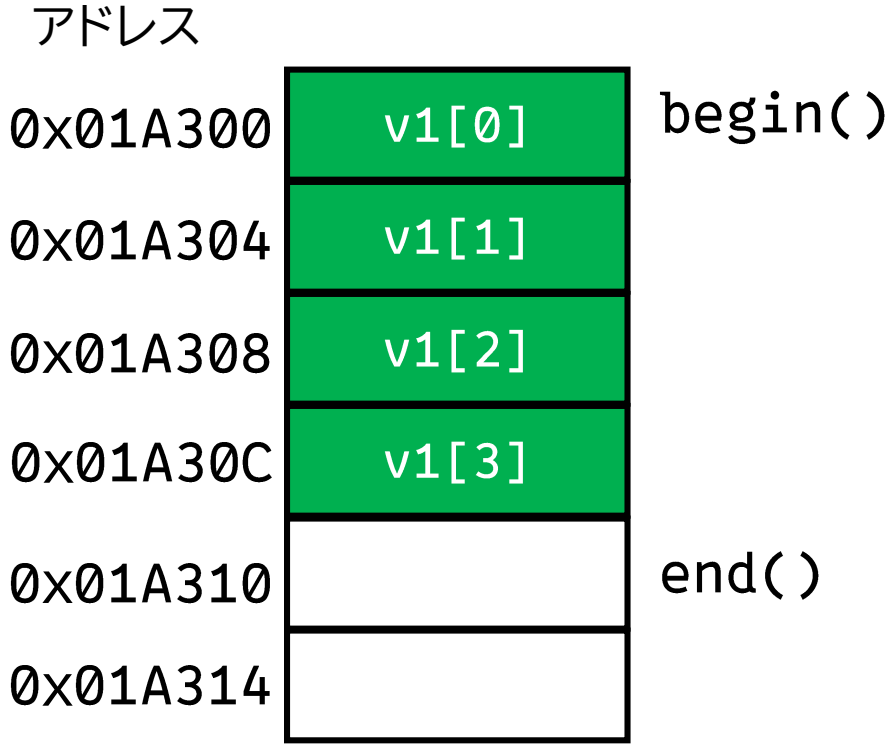
std::vector<int>::iterator

```
    co       "v1の要素数:" << v1.size() << endl;

    auto itrV1 = v1.begin();   //autoによる型推論で宣言を省略
    cout << "イテレータが指す要素の値:" << *itrV1 << endl;
    v1.insert(itrV1 + 1, 11); //指定した場所（先頭から+1番目）へ要素（11）を追加
    v1.erase(itrV1 + 2);      //指定した場所（先頭から+2番目）の要素を削除

    for (int i = 0; i < v1.size(); i++) {
        cout << "v1[" << i << "]=" << v1[i] << endl;
    }
```

イテレータ(iterator:反復子)

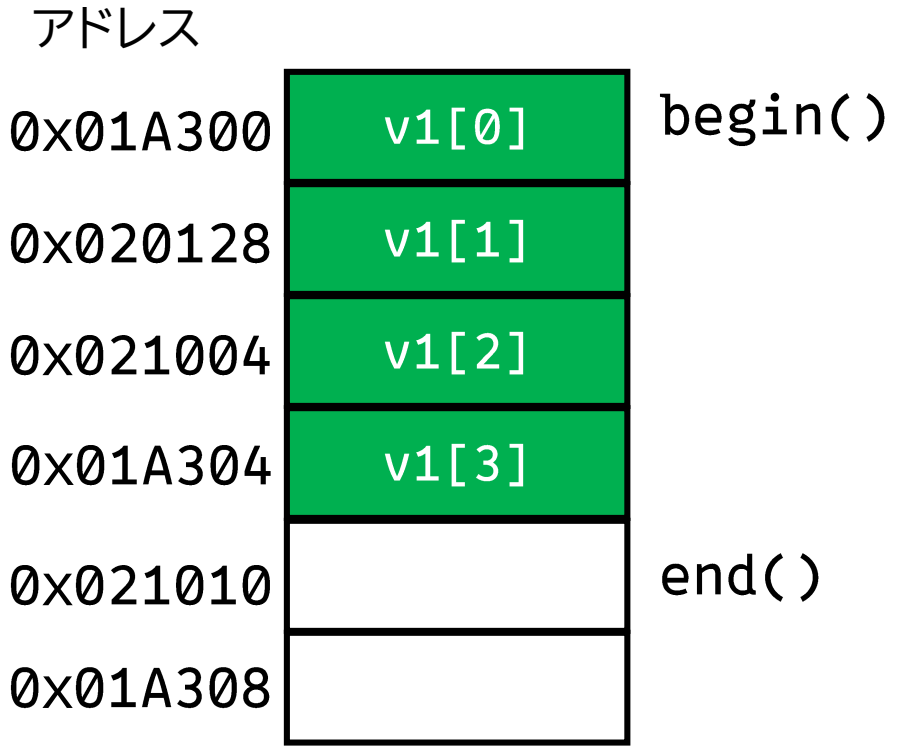
- イテレータの必要性

静的配列



連続したアドレスに配置

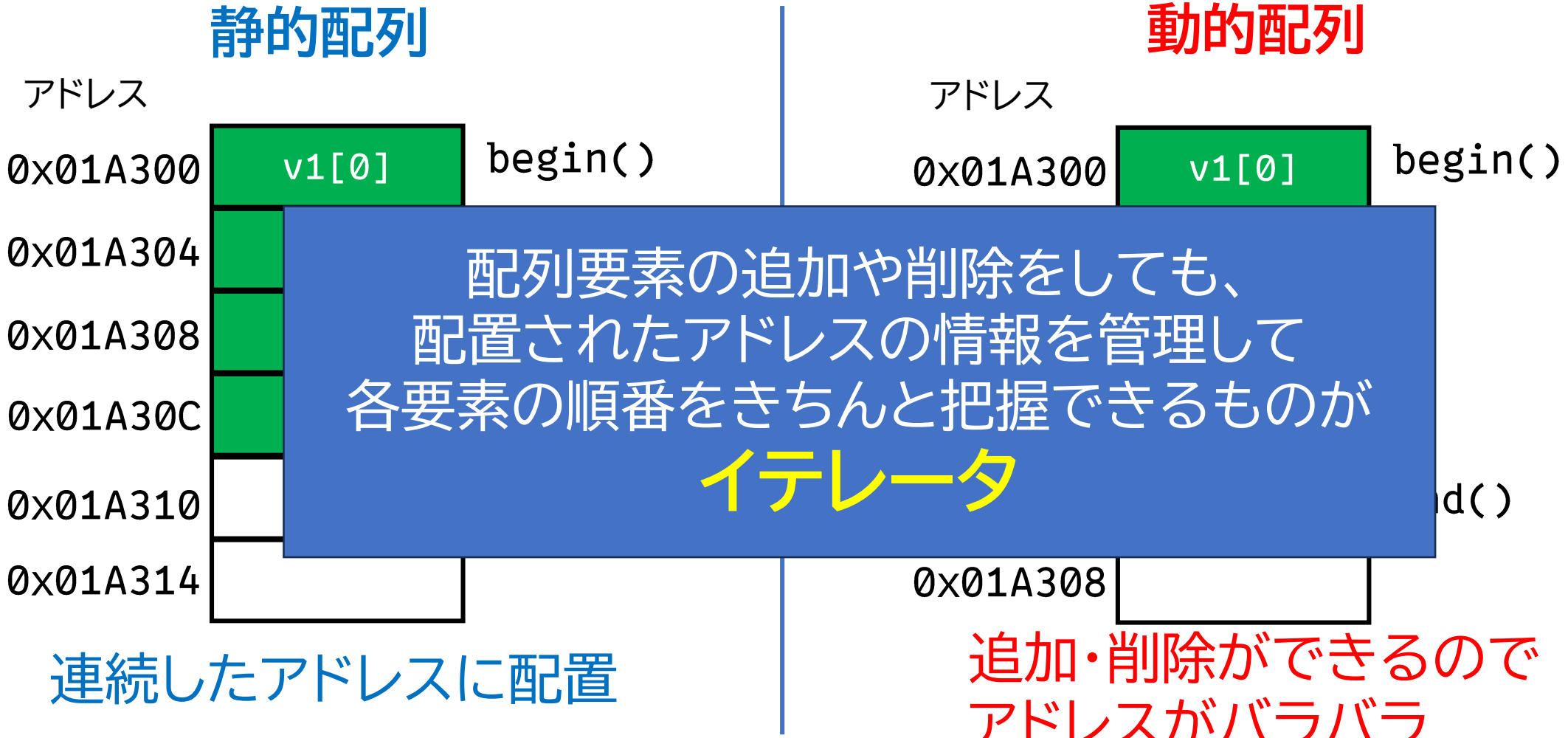
動的配列



追加・削除ができるため
アドレスが不連続

イテレータ(iterator:反復子)

- イテレータの必要性



イテレータ(iterator:反復子)

main.cpp (Sample604)

```
v2.push_back("DEF");
v1.pop_back();
v1.emplace_back(4);
cout << "v1の要素数:" << v1.size() << endl;

auto itrV1 = itrV1.begin();
cout << "イテレータが指す要素の値:" << *itrV1 << endl;
v1.insert(itrV1 + 1,11);
v1.erase(itrV1 + 2);
// for (int i = 0; i < v1.size(); i++) {
//      cout << "v1[" << i << "]=" << v1[i] << endl;
// }
for (auto itrV1 = v1.begin(); itrV1 != v1.end(); ++itrV1) {
    cout << *itrV1 << endl; //itrV1.end()は最終要素のひとつ後
}
```

イテレータを用いた要素の表示

イテレータ(iterator:反復子)

- イテレータを用いたループ処理

```
for (auto itrV1 = v1.begin()
          ; itrV1 != v1.end()
          ; ++itrV1)
```

v1の先頭要素からv1の最終要素まで
イテレータitrV1をひとつずつ進めながら、
ループ処理を行う

イテレータ(iterator:反復子)

- **範囲for文**
  イテレータを用いたfor文を、より簡略化したもの

```
for (auto itr : v1){    //範囲forの場合
  cout << itr << endl; //itrに”*”は不要
}
```

v1の先頭要素から最終要素まで、すべての要素ぶんループ
(※　ただし指定場所からの開始・終了は**不可**)

イテレータ(iterator:反復子)

main.cpp (Sample604)

```
    v2.push_back("DEF");
    v1.pop_back();
    v1.emplace_back(4);
    cout << "v1の要素数:" << v1.size() << endl;

    auto itrV1 = itrV1.begin();
    cout << "イテレータが指す要素の値:" << *itrV1 << endl;
    v1.insert(itrV1 + 1,11);
    v1.erase(itrV1 + 2);
    // for (int i = 0; i < v1.size(); i++) {
    //    cout << "v1[" << i << "]=" << v1[i] << endl;
    // }
    // for (auto itrV1 = v1.begin(); itrV1 != v1.end(); ++itrV1) {
    //    cout << *itrV1 << endl;
    // }
    for (auto itr :v1) {        //範囲for文にするとすべての要素を順次取り出す形となる
        cout << itr << endl;
    }
```

範囲forを用いた要素の表示

vectorクラス

- vectorまとめ
  - **動的配列**を実現するコンテナクラス
  - 最初に要素数を指定する必要がなく、適宜増減可能
  - C言語の配列と同様に**添え字番号**で要素にアクセス
  - 末尾に要素を追加、末尾の要素を削除可能
  - 末尾以外の場所にも追加・削除可能だが**イテレータ**による場所の指定が必要
  - 基本データ型以外のクラスインスタンスも格納可能