第6章「ライブラリのアルゴリズムを使う」

主なトピック

- イテレータを引数にとりジェネリック（汎用）アリゴリズム

本日の内容

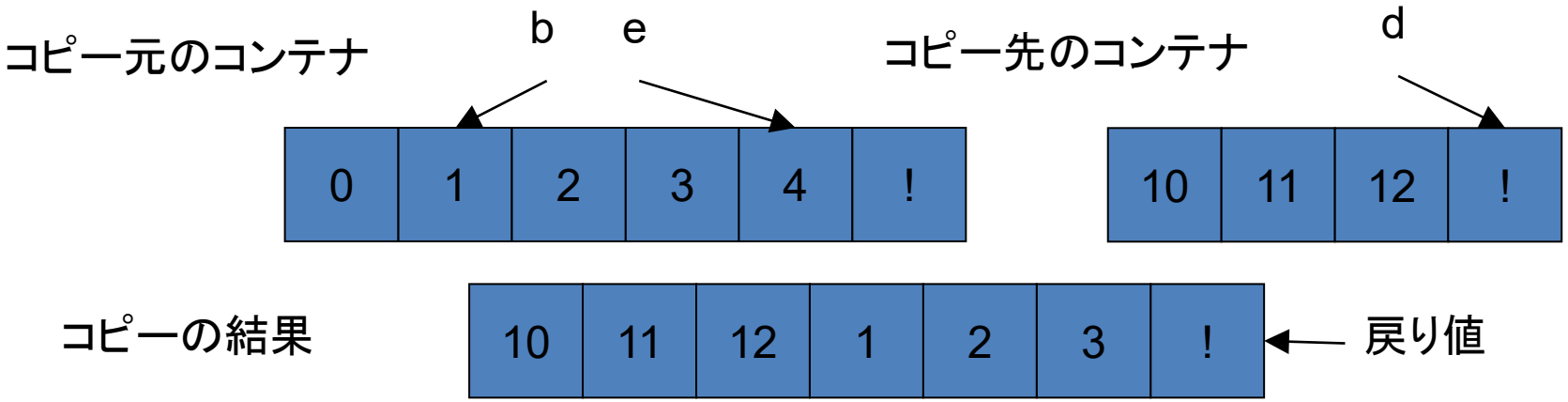
- ジェネリック（汎用）アルゴリズム
- イテレータの計算
- copy
- insert
- erase
- search, find, find_if
- remove, remove_if , remove_copy, remove_copy_if
- partition, stable_partition

ジェネリック（汎用）アルゴリズム

- 特定のコンテナに付随しているのではなく，引数の型からデータ形式を判断し，多くのコンテナに共通して作用できるアルゴリズム
- vector, list, string型などを対象に同じ関数で利用可能
- 通常，引数はイテレータ（反復子）で与えられる
- copy, searchなど
  - 異なる例: c.insert(), c.erase()，クラスのメンバ関数
- #include <algorithm>

copy

- copy(b, e, d)
  - 入力イテレータ b と e (eは含まない)の間の値を，出力イテレータ d で示される位置にコピーする
  - コピー先の最後の要素の一つを後を指すイテレータが返される



copyの例

```
vector<int> src, dst;
//  src[0] = 0; src[1] = 1; src[2] = 2; src[3] = 3; src[4] = 4;
//  dst[0] = 10; dst[1] = 11; dst[2] = 12;

//  srcの内容をすべてdstの最後にコピーする
copy( src.begin(), src.end(), back_inseter( dst ) );

//  srcのbからeの範囲をdstの最後にコピーする
vector<int>::iterator b, e;
//  bとeのイテレータを設定，例えば
b= ++(src.begin());  e = --( src.end() );
copy( b, e, back_inserter( dst ) );
```

イテレータアダプタ

- イテレータを返す関数
  - とくに以下のインサータがよく使われる
- #include <iterator>の中で定義
- back_inserter(c);
  - c というコンテナの最後に要素を追加するための出力イテレータを返す
  - push_back メンバ関数を持ってるコンテナにのみ利用可能(vector型, list型, string型など)

イテレータアダプタ

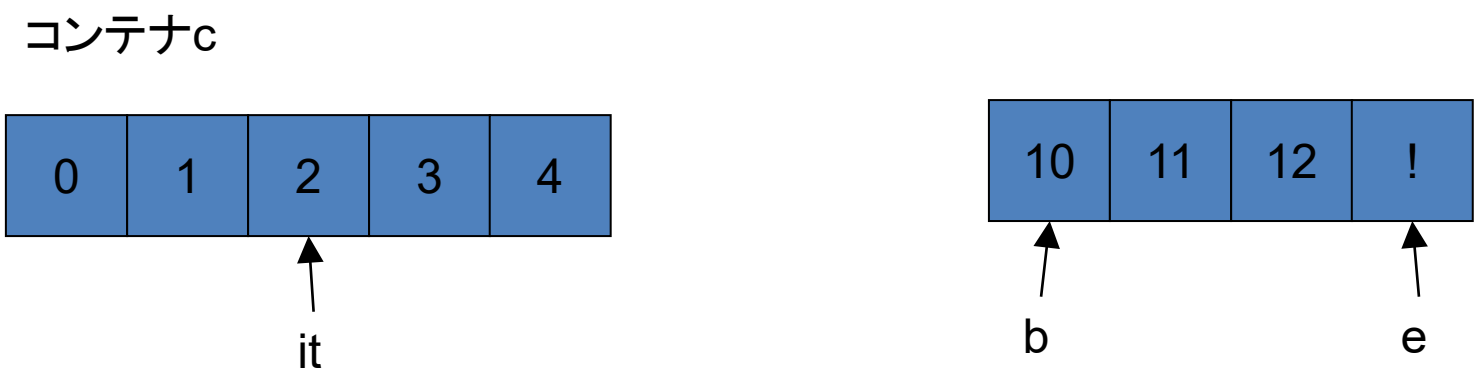
- front_inserter(c);
  - c というコンテナの先頭に要素を追加するための出力イテレータを返す
- inserter(c, it);
  - c というコンテナの it が指す要素の直前に値を挿入するための出力イテレータを返す

insert

- c.insert(it, t)
  - コンテナ c のイテレータ it が指す要素の直前に t を挿入する
  - 新しく挿入した t を指すイテレータを返す
- c.insert(it, b, e)
  - 同様にイテレータ b から e の間のシーケンスを挿入する，eそのものは挿入されない
  - void を戻す

insertのイメージ

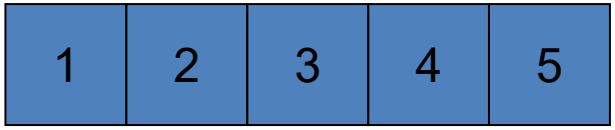
- c.insert(it, b, e)



insertの例

```
vector<int> src, dst;
//  src[0] = 0; src[1] = 1; src[2] = 2; src[3] = 3; src[4] = 4;
//  dst[0] = 10; dst[1] = 11; dst[2] = 12;

//  dst の先頭に src の全体を挿入
dst.insert( front_inserter( dst ), src.begin(), src.end() );

//  dst の2番目の要素の直前に src の全体を挿入
vector<int>::iterator it;
it = dst.begin(); ++it;
// dst.insert( it, src.begin(), src.end() );
dst.insert( inserter(dst, it), src.begin(), src.end() )
```

イテレータの計算



- vector<int> src;
- vector<int>::iterator iter;
- iter = src.begin();
  - iter+1; // srcの2番目の要素を指すイテレータ
  - cout << *(iter+1); //「2」が出力
- iter = src.end (); // 末尾の要素の一つ後を指す
  - iter-1; // srcの最後の要素を指すイテレータ
- iter + nやiter - nといった計算が可能
- （イテレータ種類に依存する→別の講義で説明する）

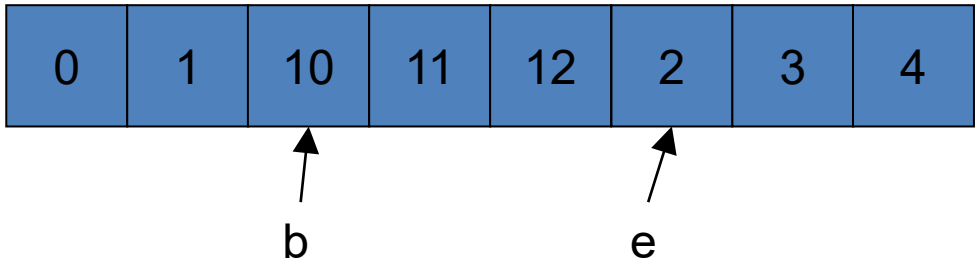
erase

- c.erase(it)
  - コンテナ c のイテレータ it が指す要素を削除する
  - 削除された要素の直後を指すイテレータを返す
- c.erase(b, e)
  - 同様にイテレータ b から e の間のシーケンスを削除する
  - eそのものは削除されない
  - 削除された要素の直後を指すイテレータを返す

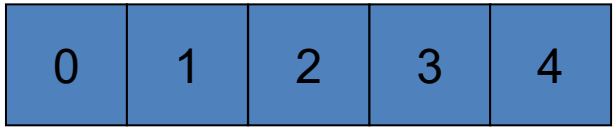
eraseのイメージ

- c.erase(b, e)

コンテナc



結果



eraseの例

```
vector<int> src;
vector<int>::iterator iter;

iter = src.begin();
while( iter != src.end() ){
   if( 何らかの条件 ){
        iter = src.erase(iter);
   }
   else {
        ++iter;
   }
}
```

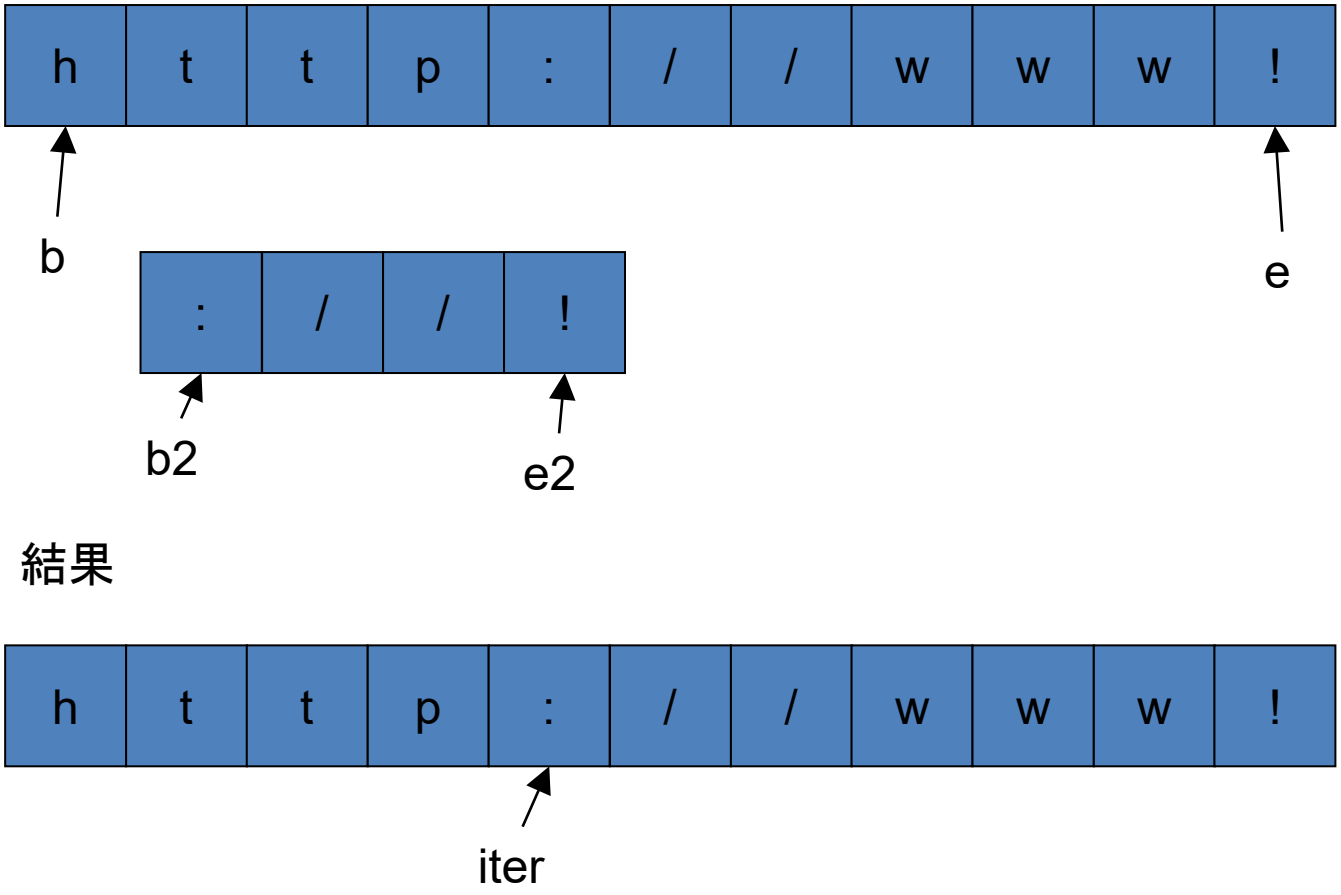
eraseの例

```
vector<int> src;
vector<int>::iterator b, e;

//  先頭から末尾の要素を削除
src.erase( src.begin(), src.end() );

b = src.begin() + 1;
e = src.begin() + 5;
//  2番目の要素から5番目の要素を削除
src.erase( b, e );
```

search

- search(b, e, b2, e2)
- search(b, e, b2, e2, p)
  - イテレータ b と e で指定される範囲の中から，イテレータ b2 から e2 のシーケンスが最初に現れる位置を示すイテレータを返す
  - マッチする文字列がなければ，e を返す
  - 上の例では==等しいかの判定をする
  - 下の例では判別関数pで等しいかを判定する

searchのイメージ



searchの例

```
string src = “http://www.u-aizu.ac.jp/”
string sep = “://”;
string::iterator iter;

//  srcからsepの文字列に位置を探す
iter = search( src.begin(), src.end(), sep.begin(), sep.end() );
//  sepが含まれているので，iterは「:」の位置を指すことになる
```

find, find_if

- find(b, e, t)
  - イテレータ b と e のシーケンスから t と最初に一致するイテレータを返す
- find_if(b, e, p)
  - 同様に判別関数pと最初に一致するイテレータを返す
- findは要素，searchはシーケンスを探す

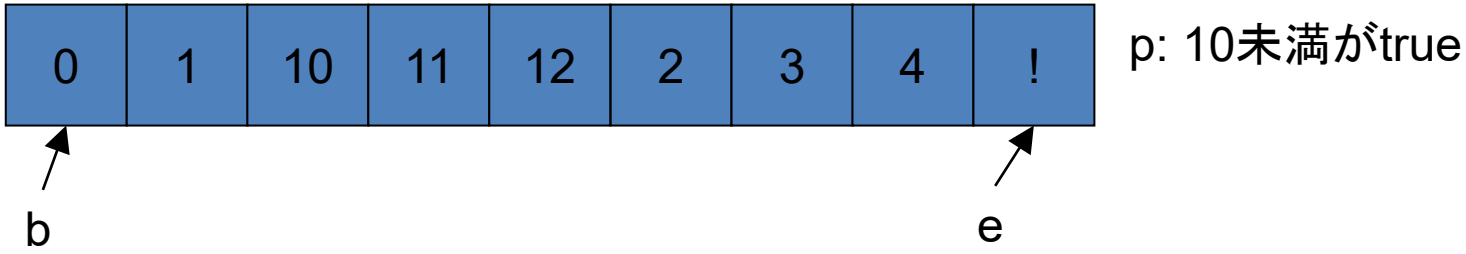
find_ifの例

```
string src = “http://www.u-aizu.ac.jp/”
string::iterator iter;

//  srcからアルファベット以外の文字列が最初に現れる位置を探す
iter = find_if( src.begin(), src.end(), compare );

bool compare(char c)
{
   return ( !isalpha(c) );
}
```

remove, remove_if

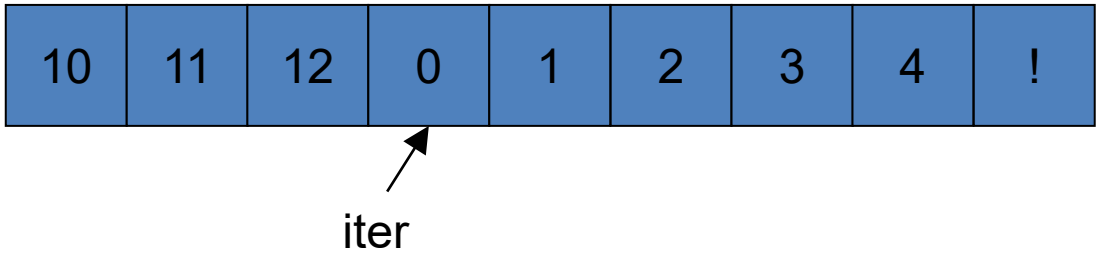
- remove(b, e, t)
- remove_if(b, e, p)
  - イテレータ b から e の間のシーケンスを並び替え，要素 t と一致しない要素，あるいは判別関数 p がfalseを返す要素を，シーケンスの前の方に集める
  - 戻り値は，集められた部分の一つ後ろを指すイテレータ（次のブロックの先頭）
  - removeという単語は，ここでは「削除」ではなく「除外」という意味

remove_ifのイメージ



p: 10未満がtrue

remove_ifの結果



remove_copy, remove_copy_if

- remove_copy(b, e, d, t)
- remove_copy_if(b, e, d, p)
  - イテレータ b から e の間のシーケンスの中から，要素 t と一致しない要素，あるいは判別関数 p がfalseを返す要素を，反復子 d が示す位置にコピーする
  - b と e の間のシーケンスは変化しない
  - 戻り値は，コピー先の最後の要素の一つ後を指すイテレータ

remove_copy_ifの例

```
//  students には既にデータが入っていると仮定する
//  不合格の生徒を fail にコピーする
//  students のデータはそのまま
vector<Student_info> students, fail;

//  students の先頭から末尾を対象に，pgradeがfalseを返す(f不合格)データを
failの末尾にコピーする
remove_copy_if( students.begin(), students.end(),
back_inserter(fail), pgrade);
```

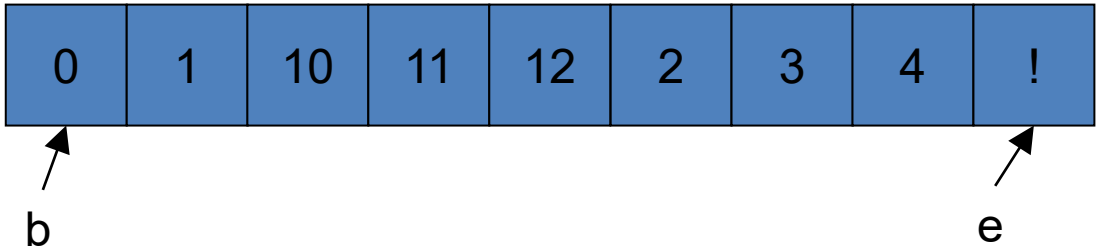
remove_copy_ifの例

```
bool pgrade(const Student_info& s){
   if(sが合格なら)        {
        return true;
   }
   else          { // 不合格なら
        return false;
   }
}
```

partition, stable_partition

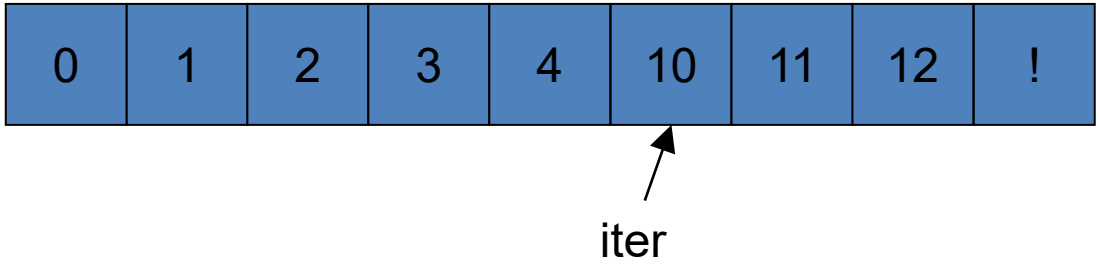
- partition(b, e, p)
- stable_partition(b, e, p)
  - イテレータ b から e の範囲を判別関数 p に基づいてグループ分けする.
  - p が true の要素が先頭にくる
  - 戻り値は p が false になる最初の要素
  - stable_partition は分類されたグループ内での順番が, もとからあった順番どおりになる

stable_partitionのイメージ



p: 10未満がtrue

stable_partitionの結果



partition関数だと0から4までの順序と,
10から12までの順序が保証されない

stable_partitionの例

```
//  students には既にデータが入っていると仮定する
//  students を合格と不合格に分類し，studentsを並び替える
vector<Student_info> students;
vector<Student_info>::iterator iter;

iter = stable_partition( students.begin(), students.end(),
   pgrade);
//  これで，studentsの前半には合格した生徒のデータが，後半には不合格の生徒のデータが集められる
//  iterには後半の先頭を指すイテレータが入る
```

演習第6回のヒント

- AlphaWords と OtherWords に分類
  - stable_partition
- AlphaWordsでは平均以上と以下に分類
  - stable_partition
- URLを「://」の前の後に分解
  - search
  - copy
  - イテレータ（反復子）の計算（n文字先）