第8章「ジェネリック関数を書く」

主なトピック

- map型（あるいはKey-Value型，連想記憶）

ジェネリック関数

- 型に依存しない関数
- どんな型の変数が引数に与えられても使える関数
- 標準ライブラリのジェネリック関数の例
- search(b, e, b2, e2)
  - b から e で指定されるシーケンスの中から b2 から e2 のシーケンスを探す
  - b, e, b2, e2は，vector, list, stringなど，どんなコンテナでも利用可能

search関数の利用例

```
//  例1
string src = “http://www.u-aizu.ac.jp/”, sep = “://”;
search( src.begin(), src.end(), sep.begin(), sep.end() );

//  例2
vector<int> data, query;
//  data = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};, query = {4, 5,
   6};
search( data.begin(), data.end(), query.begin(),
   query.end() );
```

- 引数はすべてイテレータとして与えられている

ジェネリック関数とイテレータ

- イテレータの操作(++ など)は，元のデータの型に(char, int など)に依存しない
- コンテナそのものではなく，イテレータを引数にすることにより，データの型に依存しない処理が可能になる
- そのための手法がテンプレート関数

テンプレート関数の例

```
template < class T >
T median( vector<T> v)
{
   vector<T>::size_type size = v.size(), mid;
   if(size == 0)
       throw domain_error( “median of empty vector” );
   sort( v.begin(), v.end() );
   mid = size / 2;
   if( size % 2 == 0 )
       return ( v[ mid ] + v[ mid +1 ] ) / 2;
   else
       return v[ mid ];
}
```

テンプレート関数の例の解説

- 与えられた vector 型の変数のメディアンを計算する関数
- 引数は vector<T> 型，戻り値は T 型
  - 例えば，引数が vector<double> なら戻り値もdouble
  - 引数が vector<int> なら戻り値も int
- 実際の関数においては，Tが double だろうと int だろうと，処理に違いはない
- ただ，型が違うだけ

テンプレート関数の例の解説

- template < class T >
  - 以下の関数では，Tはクラスの型を表すと宣言
  - それ以降では，Tはクラスとして扱われ，コンパイルされる
  - T の実際の型は，median関数が呼び出されたときの引数で，自動的に判断

ジェネリック関数と型

```
//   2つの引数を取り，大きいほうの値を返す関数
//   この関数自体は，何の問題もない
template < class T >
T max( const T& a, const T& b )
{
   if( a > b )
       return a;
   else
       return b;
}
```

ジェネリック関数と型

```
//   しかし，max関数の呼び出し方によっては，コンパイルエラーになる
int main()
{
   int a = 10;
   double b = 20.0;
   cout << max( a, b );
}
//   コンパイラは，テンプレートのTが int なのか double なのか判定できないため
```

データ構造非依存性

- 標準ライブラリの例
- find(c.begin(), c.end(), val);
  - なぜ，このように設計（引数がイテレータ）されているのか？
- c.find(val)でも，いいのでは？
  - 各コンテナ(vector, list, string)ごとにfindを定義しなければならない
- find(c, val)でも，いいのでは？
  - これだと，コンテナ全体からの探索のみ

アルゴリズムとイテレータ

- ジェネリック関数を書くためには，コンテナとイテレータの種類を理解する必要がある
  - 入力イテレータ
  - 出力イテレータ
  - 前方向イテレータ
  - 双方向イテレータ
  - ランダムアクセスイテレータ

入力（インプット）イテレータ

- 以下の機能を満たすイテレータ
  - ++（前置と後置の両方）
  - ==（イテレータが等しいかの判定）
  - !=（イテレータが異なるかの判定）
  - *（イテレータが指す要素に読み込みアクセス）
  - ->（イテレータが指す要素のメンバに読み込みアクセス）

入力イテレータの例

- 順次読み込み専用アクセスの例, `find (1)`

```
template < class In, class X >
In find( In b, In e, const X& x )
{
   while( b != e && *b != x )      {
        ++ b;
   }
   return b;
}
```

- `b` と `e` が入力イテレータ
- `b` から `e` の間の最初に現れた要素`x`を指すイテレータを返す, なければ `e` を返すことになる

入力イテレータの例

- 順次読み込み専用アクセスの例, find (2)

```
template < class In, class X >
In find( In b, In e, const X& x )
{
   if (b == e || *b == x)   {
        return b;
   }
   b ++;
   return find( b, e, x );
}
```

- 再帰関数による表現

出力（アウトプット）イテレータ

- 以下の機能を満たすイテレータ
  - 読み込みと書き込みの違いを除いて，入力イテレータと同じ

出力イテレータの例

- 順次書き込み専用アクセスの例, copy

```
template < class In, class Out >
In copy( In b, In e, Out d )
{
   while( b != e )   {
        *d = *b;  d++;   b++;
   }
   return d;
}
```

- b から e の間の要素を d が指す位置にコピー
- d が出力イテレータ, b と e は入力イテレータ

前方向（フォワード）イテレータ

- 以下の機能を満たすイテレータ
  - 入力イテレータと出力イテレータを合わせたもの
  - 読み書きのアクセス

前方向イテレータの例

- 順次読み書きアクセスの例, replace

```
template < class For, class X >
void replace ( For b, For e, const X& x, const X& y )
{
   while( b != e )   {
        if( *b == x )
                *b = y;
        ++ b;
   }
}
```

- b から e の間の要素の x を y に置換
- b と e が前方向イテレータ

双方向（バイディレクショナル）イテレータ

- 以下の機能を満たすイテレータ
  - 前方向イテレータに -- を加えたもの

双方向イテレータの例

- **可逆アクセスの例**, reverse

```
template < class Bi > void reverse ( Bi b, Bi e )
{
   while( b != e )   {
        -- e;
        if( *b != *e )
               swap(*b++, *e);
   }
}
```

- b から e の間の要素の順序を入れ替える
- b と e が双方向イテレータ

ランダムアクセスイテレータ

- 以下の機能を満たすイテレータ
  - 双方向イテレータにイテレータの計算を加えたもの
  - p と q をイテレータ，n を整数として
  - p + n, p - n, n + p
  - p – q
  - p[n], *(p + n) と同じ
  - p < q, p > q, p <= q, p >= q

ランダムアクセスイテレータの例

- **ランダムアクセスの例**, `binary_search`

```
template < class Ran, class X >
bool binary_search ( Ran b, Ran e, const X& x )
{
   while( b < e )    {
        Ran m = b + (e - b) / 2;
        if( x < *m )  e = m;
        else if( *m < x )    b = m + 1;
        else           return true;
   }
   return false;
}
```

- `b` **と** `e` **がランダムアクセスイテレータ**

イテレータとコンテナ

- すべての標準ライブラリのコンテナで適用可能
  - 入力イテレータ
  - 出力イテレータ
  - 前方向イテレータ
  - 双方向イテレータ
- vector, stringでのみ適用可能
  - ランダムアクセスイテレータ
  - sort関数が使えるのもランダムアクセスイテレータのみ

演習8のポイント

- template < class For1, class For2 >
  For1 my_search(For1 b, For1 e, For2 b2, For2 e2);
- template < class In, class Out >
  Out my_copy(In b, In e, Out d);
- template < class Bi, class P >
  Bi my_partition(Bi b, Bi e, P p);

演習8のポイント

- partition関数では，イテレータが指す要素の値の入れ替えを行う
- イテレータから，それが指す値の型を知るために
- iterator_traits< Bi >::value_type x;
  - Bi が vector<int>::iterator なら int 型
  - Bi が vector<double>::iterator なら double 型
  - Bi が string::iterator なら char 型

演習8のポイント

- my_partition関数の中での叙述関数（判別関数）pの使い方
  - p(x); // x は通常の変数
  - p(*i); // i はイテレータ