テンプレートの利用とSTL

Game Programming B #05

向井 智彦

前回のおさらい

- 動的メモリ管理
  - new と delete，new[] と delete[]
- クラス継承とポインタ
  - 基底クラスの仮想関数を通じたポリモーフィズム
  - アップキャストとダウンキャスト
    - ダウンキャストの扱いには注意
- クラスインスタンスへのポインタの配列
  - 基本的には基底クラスへのポインタを格納
- 継承と合成：is-a or has-a

本日の内容

- テンプレート
  - 関数テンプレート、クラステンプレート
  - 作り方はさておき、使い方を中心に
- 標準テンプレートライブラリ
  - コンテナ
    - vector、map、list
  - イテレータと反復
    - vector::iterator、vector::begin(), vector::end()
  - アルゴリズム
    - sort、find、max_element, min_element

テンプレートとは？

- 変数型に依存しない関数やクラスの作成
  - C# やJavaではジェネリクス（generics）
- テンプレート関数の宣言
  - template<typename TYPE> TYPE Quad(TYPE a)
    { return a * 2; }
- 利用側
  - int a = Quad<int>(10); // int用のQuad
  - double b = Quad<double>(5.0); // double用のQuad

テンプレートクラス

```
template<typename TYPE> class Vector2 {
  Vector2(TYPE ix, TYPE iy);
  TYPE X();
  TYPE Y();
private:
  TYPE x;
  TYPE y;
};

Vector<int> intVector;
Vector<double> realVector;
```

標準テンプレートライブラリとは？

- STL: Standard Template Library
  - 一般的なアルゴリズムをテンプレートを用いて提供しているC++標準のライブラリ
  - アルゴリズムを利用するための枠組み
    - コンテナ
    - イテレータ
    - など
- その他の有名なテンプレートライブラリ
  - boost

コンテナ

- std::vector（ #include <vector> ）
  - 可変長ベクトル
- std::set（ #include <set> ）
  - 集合（重複を許さない）
- std::deque（#include <deque> ）
  - 双方向キュー
- std::map（ #include <map> ）
  - 連想配列 （辞書）

イテレータ

コンテナの種類に依存することなく、データへの反復処理を一貫して行うための仕組み

```
std::vector<int> v;
for (int i = 0; i < 10; ++i)
    v.push_back(rand());
     コンテナの型
for (std::vector<int>::iterator it = v.begin();
     it != v.end();
     ++it)
{
    std::cout << *it << std::endl;
}
```

※ポインタと同じ構文

イテレータ

コンテナの種類に依存することなく、データへの反復処理を一貫して行うための仕組み

```
std::map<char, int> m;
for (int i = 0; i < 10; ++i)
    m.insert(std::make_pair('a' + i, rand()));

for (std::map<char, int>::iterator it = m.begin();
     it != m.end();
     ++it)
{
    std::cout << it->second << std::endl;
}
```

イテレータと型推定 auto

初期化子(”=“の右辺)の型に沿って、変数型を自動的に決定する仕組み (※C++11以降)

```
std::vector<int> v;
for (int i = 0; i < 10; ++i)
    v.push_back(rand());

for (auto it = v.begin(); it != v.end(); ++it)
{
    std::cout << *it << std::endl;
}
```

イテレータと型推定 auto

「std::map<char, int>::iterator 」がイテレータ it の型として自動推定される

```
std::map<char, int> m;
for (int i = 0; i < 10; ++i)
    m.insert(std::make_pair('a' + i,rand()));

for (auto it = m.begin(); it != m.end(); ++it)
{
    std::cout << it->second << std::endl;
}
```

アルゴリズム

- #include <algorithm>

- find: 一致するデータをコンテナから検索
- sort: コンテナ内のデータをソート
- max_element: 最大要素へのイテレータ取得
- min_element: 最小要素へのイテレータ取得

演習課題1

- 10個の数字 {0, 2, 3, 1, -1, 5, 8, -6, 9, 7} の降順ソート結果を出力するプログラムを作成

  コンテナとしてvectorを利用しつつ…

  – std::sort の引数を工夫しつつ利用(no.1)

  – リバースイテレータ rbegin & rend を利用(no.2)

  および

  – コンテナとしてsetを利用する方式(no.3)

演習課題2

- 10個の数字 {0, 2, 3, 1, 4, 5, 8, 6, 9, 7} のうち偶数のみを出力するプログラムを std::find_if を用いて作成
  - コンテナは vector でも何でもOK

応用編

- 前回の課題：ShapeManagerで作成した図形データ配列を、可変長配列 vector に置き換え
  - Shape *shapes[6]; ---> vector<Shape*> shapes;
- 6つの図形データを、入力順ではなく、図形種別ごとにグループ化して表示するように変更
  - Shape::Type() を利用した find_if、もしくはsort