# オブジェクト指向と ゲームプログラミング

## ○　C++編 － 第１１回　クラステンプレート　○

### 構造体テンプレート

テンプレートは、構造体やクラスに対しても使用することができます。キャラクターのｘ，ｙ座標を整数で保持するPOINT構造体を考えてみましょう。たいてい、以下のようになるでしょう。

```
struct POINT {
    int   x;
    int   y;
};
```

しかし、あとになって座標はintではなくdoubleに変更するとしたら、intの部分をdoubleに書き換える…というのはあまりよい方法ではありません。また、intのPOINT構造体とdoubleのPOINT構造体を１つのプログラムで混合して使いたいということも考えられます。このようなときに使えるのがテンプレートです。上記の構造体をテンプレートにすると、以下のようになります。

```
template <class T>
struct POINT {
    T   x;
    T   y;
};
```

構造体テンプレートPOINTを使用するには、「構造体名<型名>」という形で宣言します。

```
POINT<int>      pt_i;   // TをintにしたPOINT構造体になる
POINT<double>   pt_d;   // TをdoubleにしたPOINT構造体になる
POINT<float>    pt_f;   // TをfloatにしたPOINT構造体になる
```

上記のように、POINT<int>と宣言するとint型のメンバx, yを持つ構造体になり、POINT<double>と宣言するとdouble型のメンバx, yを持つ構造体になります。型名には構造体名やクラス名を指定することもできます。

### クラステンプレート

テンプレートはクラスに対しても使用できます。以下のようなキャラクターの座標を管理するクラスを作成したとします。

```
class CPoint {
public:
    CPoint() : m_x(0), m_y(0) {}
    CPoint(const int x, const int y) : m_x(x), m_y(y) {}

    void set_x(const int x) { m_x = x; }
    void set_y(const int y) { m_y = y; }

    int get_x() const { return m_x; }
    int get_y() const { return m_y; }

private:
    int   m_x;
    int   m_y;
};
```

仕様が変更になり、座標はintではなくdoubleにするとしたら、intの部分をdoubleに書き換える…というのは非常に面倒です。このようなときに使えるのがクラステンプレートです。
上記のクラスをテンプレートにすると、次のようになります。

```
template <class T>
class CPoint {
public:
    CPoint() : m_x(0), m_y(0) {}
```

```
    CPoint(const T x, const T y) : m_x(x), m_y(y) {}

    void set_x(const T x) { m_x = x; }
    void set_y(const T y) { m_y = y; }

    T get_x() const { return m_x; }
    T get_y() const { return m_y; }

private:
    T   m_x;
    T   m_y;
};
```

クラステンプレートCPointを使用するには、「クラス名<型名>」という形で宣言します。

```
CPoint<int>      pt_i;  // Tをint   にしたCPointクラスになる
CPoint<double>   pt_d;  // Tをdoubleにしたクラスになる
CPoint<float>    pt_f;  // Tをfloat にしたCPointクラスになる
```

上記のように、CPoint<int>と宣言するとint型のメンバ変数m_x, m_yを持つクラスになり、CPoint<double>と宣言するとdouble型のメンバ変数m_x, m_yを持つクラスになります。型名には構造体名やクラス名を指定することもできます。

少し難しいのがクラステンプレートのメンバ関数をクラスの外で定義する場合です。クラステンプレートのメンバ関数は、自動的に関数テンプレートとして扱われます。そのため、最初にキーワードtemplateを宣言する必要があります。さらに、関数テンプレートの名前には必ずクラステンプレートの名前を付けなければなりません。
たとえば、クラステンプレートCPointのメンバ関数set_xをクラスの外で定義するには、以下のようになります。

```
template<class T>
void Point<T>::set_x(const T x)
{
    m_x = x;
}
```

クラステンプレートの名前は、CPointではなくCPoint<T>になります。ちなみにコンストラクタは、

```
template<class T> CPoint<T>::CPoint(const T x, const T y) …
```

のようになります。