オブジェクト指向と
ゲームプログラミング

○　C++編 － 第１０回　関数テンプレート　○

テンプレート

２つの数を受け取り、大きい方を返すmax関数を作成するとします。たいてい、以下のような関数になるでしょう。

```
int max(const int value1, const int value2)
{
    if(value1 > value2)
        return value1;
    return value2;
}
```

この関数は、整数に対しては問題なく使うことができますが、floatやdoubleなど、整数でない型には使うことができません。そこで、以下のような関数を作成することが考えられます。

```
float max(const float value1, const float value2)
{
    if(value1 > value2)
        return value1;
    return value2;
}

double max(const double value1, const double value2)
{
    if(value1 > value2)
        return value1;
    return value2;
}
```

このような解決方法でも、もちろん正しく動作しますが、型ごとに関数を作成するのはとても面倒です。そこで、テンプレートという仕組みが考案されました。テンプレートを用いると、さまざまな型に適用できる一般的な関数やクラスを記述できるようになります。

マクロとテンプレート

C言語にはマクロがあり、max関数をマクロで定義することにより型のない処理を作成することができます。マクロ版max関数は以下のようになるでしょう。

```
#define MAX(x, y) ((x) > (y) ? (x) : (y))
```

マクロの使用は、演算子の優先順位規則とマクロで定義された処理との矛盾によるバグの危険性が常につきまといます。また、引数を使ってマクロを呼び出すことによる副作用は、一般的に致命傷になりやすい言われています。マクロは処理できることが限られており、記述が難しくなるだけでなく読むのも困難になってしまいます。
型のないマクロはC++が提供する型チェックの保護を受けられません。マクロは型のない関数を作成しますが、テンプレートは後で型を指定することができる関数を作成します。テンプレートはC++の一部として機能し、マクロが持つ様々な問題を解決することができるのです。

関数テンプレート

前述の３つのmax関数を比較すると、これらの間にある唯一の違いは、型がそれぞれint, float, doubleになっているということです。この３つの関数を関数テンプレートで置き換えるとどうなるのでしょうか。テンプレートでは、int, float, doubleなどの型をTという仮の型に置き換えてしまうのです。

```
// max - 型Tの２つの数のうち大きい方を返す
template <class T>
T max(const T t1, const T t2)
{
    if(t1 > t2)
```

```
        return t1;
    return t2;
}
```

maxは関数ではなく、関数テンプレートです。キーワードtemplateの後に鍵カッコが続き、その中には１つ以上のキーワードclassがついた型名や定数、またはその両方を含むことができます。その下に続くカッコ()は、通常の関数の定義のものに類似しています。このカッコの中で定義されるおのおのの名前(t1とt2)は、テンプレートの定義のどこかで必ず使用しなければなりません。
この例の関数テンプレートmaxは､型がTである２つのオブジェクトt1とt2のうち大きい方を返します。Tは後で指定されるなんらかのクラス(型)です。

以上のように定義した関数テンプレートmaxは、プロトタイプ定義や関数呼び出しが記述されることにより実体化されます。

```
double max(const double d1, const double d2);   // double版max関数のプロトタイプ

int n = max(100, 200);                           // int版max関数の呼び出し
```

このような記述があると、コンパイラが仮の型Tをそれぞれdoubleやintに置き換え、置き換えた結果をコンパイルし、関数を実体化するのです。

関数テンプレートは、DirectXにも応用することができます。DirectXでは、生成したオブジェクトの解放処理を以下のように行います。

```
// DirectInputインタフェースの解放(pDInputはIDirectInput8*型)
if(pDInput != NULL) {
    pDInput->Release();
    pDInput = NULL;
}

// DirectDrawインタフェースの解放(pDDrawはIDirectDraw7*型)
if(pDDraw != NULL) {
    pDDraw->Release();
    pDDraw = NULL;
}
```

２つの解放処理の違いは、IDirectInput8*型とIDirectDraw7*型という型の違いだけなので、関数テンプレートに置き換えることができます。

```
template <class T>
void SafeRelease(T& x)
{
    if(x != NULL) {
        x->Release();
        x = NULL;
    }
}
```

関数テンプレートSafeReleaseにより、２つの解放処理は以下のように記述することができます。コンパイラにより、オブジェクトの型に合わせたSafeRelease関数が作成されるからです。

```
SafeRelease(pDInput);   // DirectInputオブジェクト解放
SafeRelease(pDDraw);    // DirectDrawオブジェクト解放
```

なお、関数テンプレートとinlineを組み合わせることもできます。こうすると、コンパイラにより関数が実体化された場合、インライン展開されるようになり、関数呼び出しのオーバーヘッドがなくなります。

```
template <class T>
inline void SafeRelease(T& x)
{
    if(x != NULL) {
        x->Release();
        x = NULL;
    }
}
```