オブジェクト指向と
ゲームプログラミング

○　C++編 － 第５回　参照型，this，静的メンバ　○

参照型

C++には、C言語に存在しない参照型(リファレンス)という型が存在します。C言語では、関数に構造体を渡す場合、ポインタを利用するのが一般的です。しかし、間接メンバ演算子"->"の概念が若干複雑であり、また未初期化ポインタや無効なポインタが渡されたときには不正な参照が発生してしまいます。こういった問題を解決するため、C++ではポインタに似て異なる「参照」という概念が導入されました。

```
int     a = 10;
int&    b = a;      // bはaの参照。変数名は別でも同じ領域を参照する変数である

b = 9999;           // bはaの参照なので、bを9999にするとaも9999になる
```

参照型は、"&"を伴って宣言された変数で、上記のように必ず宣言と同時に初期化が必要になります。参照型の変数は、同じ変数に別の名前を付けるような効果が得られます。名前は異なっても同じ変数を参照しているので、一方を変更するともう一方も変更されます。
参照型は、構造体やクラスを関数の引数にするときに使用します。これは、不正なポインタによるバグや、コピーを作成するオーバーヘッドを少なくするためです。なお、参照型で渡された引数を関数内で変更すると、当然呼び出し元の変数も書き換えられます。

thisポインタ

メンバ関数内で、自分自身を参照したいことがあります。たとえば、メンバ関数内でほかの関数を呼び出す際に、引数で自分自身のオブジェクトを渡すような場合です。同じように、メンバ関数が呼び出し元に対して、操作中のオブジェクト自身を返す必要が生じる場合があります。
そういった場合に、C++ではキーワードthisを使ってオブジェクト自身を参照することができます。thisを宣言したり、何か特別なことを行う必要はありません。thisはメンバ関数内なら常に使用可能であり、現在のオブジェクトに対するポインタを返します。
thisポインタは、オブジェクトを指すポインタが間接メンバ演算子"->"を使ってアクセスするのと同じように使うことができます。以下の文では、現在のオブジェクトのvisit関数を呼び出し、引数として自分自身を渡しています。

```
this->visit(this);      // 現在のオブジェクトのvisit関数の呼び出し
```

同様に、現在のオブジェクトのメンバ変数にアクセスするために、変数名の前にthisを付けることができます。

```
this->m_PosX = 89.3;    // 現在のオブジェクトのメンバ変数m_PosXに89.3を代入
```

ほとんどのメンバ関数では省略しているthisを、明示的に指定することができます。したがって、以下の２つの構文は同じ意味です。

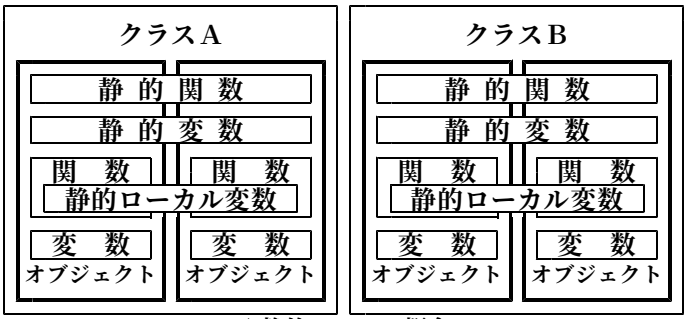
```
this->m_PosX = 89.3;    // 現在のオブジェクトのm_PosXに89.3を代入
m_PosX       = 89.3;    // メンバ変数m_PosXに89.3を代入(thisを省略した場合)
```

上記のようにthisを使うことが有益な状況としては、メンバ関数内のローカル変数とメンバ関数が同じ名前を持っている場合が考えられます。thisを使うことによって、これら２つの変数を区別することができます。なお、staticを使って宣言された静的なメンバ関数内では、thisを使うことはできません。

静的メンバ

同じクラスのオブジェクトが複数存在する際、オブジェクト間でデータを共有したい場合があります。このような場合には、キーワードstaticを使って変数(メンバ変数またはメンバ関数のローカル変数)を宣言します。
クラスのオブジェクトをいくつ生成しても、各オブジェクトの変数はほかのオブジェクトからは独立した値を持っています。しかし、staticを使って宣言された変数は、同じクラスのすべてのオブジェクト間でその変数を共有します。したがって、あるオブジェクトがstaticな変数の値を変更すると、ほかのオブジェクトのstaticな変数もその値に変更されます。
staticなメンバ変数は、オブジェクトを生成しなくても使用することができます。また、オブジェクトが生成されるごとに初期化しても意味がないので、コンストラクタで初期化しないようにします。

メンバ関数の宣言でもstaticを指定することができます。staticなメンバ関数は、オブジェクトを生成しなくても呼び出すことができます。ただし、staticなメンバ関数の中では、そのクラスのstaticでないメンバにアクセスすることができません。どうしてもアクセスしたい場合は、その関数にthisのようなオブジェクトへのポインタや参照を渡します。



▲静的メンバの概念

static指定されたメンバを静的メンバと呼びます。静的メンバは、インスタンスを生成しなくても、クラス名を使ってアクセスすることができます。なお、staticなメンバ変数は、クラス外部のどこか(通常は対応するcppファイル)で、一度だけ変数の定義を行っておく必要があります。

```
class Character {
public:
    Character() : m_PosX(0.0), m_PosX(0.0), m_AnimeNo(0)
    { IncCount(); }     // キャラクタが生成されるごとにカウンタを１増やす

    ~Character()
    { DecCount(); }     // キャラクタが破棄されるごとにカウンタを１減らす

    void Move(const double add_x, const double add_y);  // 移動
    void Animetion();                                   // アニメーション
    static int GetCount() { return m_Count; }           // キャラクタ数取得

private:
    double   m_PosX;                                    // ｘ座標
    double   m_PosY;                                    // ｙ座標
    int      m_AnimeNo;                                 // アニメーション番号

    static int   m_Count;                               // 総キャラクタ数

    static void  IncCount() { m_Count++; }              // キャラクタ数インクリメント
    static void  DecCount() { m_Count--; }              // キャラクタ数デクリメント
};

int   Character::m_Count = 0;                   // static変数の定義

int main()
{
    cout << Character::GetCount() << endl;      // オブジェクトを生成しなくても使用可

    cout << "キャラクタ生成" << endl;

    Character   hinomoto, minazuki;
    cout << hinomoto.GetCount() << endl;        // オブジェクトを使っても使用可

    return 0;
}
```

staticなメンバ変数は、オブジェクト固有でなくクラス固有であることから、クラス変数とも呼ばれます。また、同じようにstaticなメンバ関数をクラス関数と呼びます。これに対し、staticでないメンバ変数は、オブジェクト(インスタンス)ごとに独立していることから、インスタンス変数と呼びます。同じようにstaticでないメンバ関数をインスタンス関数と呼びます。

練習問題

1　次のプログラムを実行したとき、画面に表示される文字列を答えましょう。

```
#include <iostream>

using namespace std;

void func(int& i, int j);

int main()
{
    int   x = 8, y = 8;

    func(x, y);

    cout << "x:" << x << "  " << "y:" << y << endl;

    return 0;
}

void func(int& i, int j)
{
    i *= 2;
    j *= 3;
}
```

2　以下の文章の内容が正しい場合には○、間違っている場合には×を付けましょう。

(１)staticでないメンバ関数の中からは、staticなメンバにアクセスすることはできない。
(２)staticなメンバ関数の中からは、staticでないメンバにアクセスすることはできない。
(３)メンバ関数の中でstatic宣言された変数も、同じクラスのほかのオブジェクトと共有される。
(４)戻り値を返さないメンバ関数では、return文を使えない。

3　キーワードthisについて簡潔に説明しましょう。