C++基礎　3

2006/05/10

オブジェクト指向

① クラスの継承

- クラスには、『継承』という機能があります。
- 継承とは、“既にあるクラスを元に、新しいクラスを作る”機能です。
- 継承元のクラスを『親クラス』又は『スーパークラス』あるいは『基底クラス』、継承先のクラスを『子クラス』又は『サブクラス』といいます。
- 子クラスは、親クラスのメンバ変数・メンバ関数の全てを受け継ぎます。

② 継承の書式

- クラスの継承は、次のように行います。

```
// 親クラス
class CParentClass
{
public:
    CParentClass();
    ~CParentClass();

    void    SetInt(int val);
    int     GetInt()const;
private:
    int     m_nInt;
};
// 子クラス
class CChildClass : public CParentClass
{
public:
    CChildClass();
    ~CChildClass();


};
```

CParentClass を、
public で継承、という意味

メンバ変数もメンバ関数も定義していないが、
CParentClass を継承しているため、
SetInt も GetInt も使用可能。
もちろん、m_nInt も持っている。

例：

```
#include <stdio.h>

// クラスCParentClassの宣言
class CParentClass
{
public:
    void    GetName()

private:
    int     m_nValA;
};

// クラスCParentClassの実装
int CParentClass::GetInt()const
{
    return m_nValA;
}

void CParentClass::SetInt(int val)
{
    m_nValA = val;
}

// クラスCChildClassの宣言(クラスCParentClassを継承)
class CChildClass : public CParentClass
{
};

// main(クラスCChildClassを使用)
int main()
{
    CChildClass cChild;
    cChild.SetInt(1);
    printf("%d", cChild.GetInt());

    return 0;
}
```

結果：

```
1
```

③ protected

上記クラスで、CChildClass に、継承したメンバ変数の値を表示する“DispInt”メンバ関数を追加したとします。

```
// クラスCChildClassの宣言(クラスCParentClassを継承)
class CChildClass : public CParentClass
{
public:
    void DispInt()const;
};

// クラスCChildClassの実装
void CChildClass::DispInt()const
{
    printf("%d", m_nValA);
}
```

しかし、これはコンパイルエラーになります。
m_nValA が **private** 節で宣言されているためです。**private** 節で宣言されている変数については、同クラス以外参照することが出来ないためです。
逆にもし **m_nValA** を **public** 節においてしまうと、どこからでも参照できるようになってしまうので、これは好ましくありません。

この状態を打破してくれるのが、**protected** 節です。
protected 節に書かれた変数・関数は、外からは触れられませんが、継承先のクラスからは参照可能なメンバとなります。

したがって、CParentClass の宣言を次のように修正すれば、子クラスからも **m_nValA** を参照できるようになります。

```
// クラスCParentClassの宣言
class CParentClass
{
public:
    int     GetInt()const;
    void    SetInt(int val);

protected:      <-- 変更。これで子クラスからも参照可能。
    int     m_nValA;
};
```

④ メンバ関数のオーバーライド

子クラスは、親クラスの持っているメンバ関数を上書き定義することが出来ます。これを**オーバーライド**といいます。

```
// クラスCChildClassの宣言(クラスCParentClassを継承)
class CChildClass : public CParentClass
{
public:
    int     GetInt()const;   // GetInt()を上書き(オーバーライド)
    void    DispInt()const;
};

// クラスCChildClassの実装
void CChildClass::DispInt()const
{
    printf("%d", m_nValA);
}

int CChildClass::GetInt()const
{
    return -9999;
}

// main(クラスCChildClassを使用)
int main()
{
    CChildClass cChild;
    cChild.SetInt(1);
    printf("%d¥n", cChild.GetInt());
    cChild.DispInt();

    return 0;
}
```

結果：

```
-9999
1
```

上書きされたメンバ関数が呼び出された。

m_nValA の値はセットされた『1』のまま。

ちなみに、親クラスの GetInt()が呼びたいときには、次のように明示的に書きます。

```
printf("%d¥n", cChild.CParentClass::GetInt());
```

結果：

```
1
1
```

⑤ クラスの型キャスト

親クラスのポインタには、子クラスのインスタンスのアドレスを格納することが出来ます。このとき、親クラスのポインタに対して GetInt(オーバーライドされた関数)を呼ぶと、親クラスの GetInt が呼ばれます。

```
// main(クラスCChildClassを使用)
int main()
{
    CChildClass cChild;
    cChild.SetInt(1);

    CParentClass*    pParent = &cChild; // 親クラスのポインタにキャスト

    printf("%d¥n", cChild.GetInt());
    printf("%d¥n", pParent->GetInt());

    return 0;
}
```

結果：

```
-9999
1
```

⑥ 仮想関数

⑤にて、親クラスのポインタにキャストされた場合、オーバーライド前のメンバ関数が呼ばれてしまうのが嫌な場合、親クラスのメンバ関数を仮想関数にします。
仮想関数にするには、関数に『virtual』という修飾子をつけます。この修飾子はクラス宣言部にのみ必要です。

```
// クラスCParentClassの宣言
class CParentClass
{
public:
    virtual int  GetInt()const;
    void         SetInt(int val);

protected:
    int      m_nValA;
};

// クラスCParentClassの実装
int CParentClass::GetInt()const
{
    return m_nValA;
}
```

virtual をつけると、
この関数は仮想関数になる。

実装には virtual はつけない。

結果：

```
-9999
-9999
```

ココまでのことを踏まえると、次のようなコードが書けることが分かるかと存じます。

```
class CChild{
public:
    virtual const char*  GetPosition()const           { return "子供"; }
    void                 SetName(const char* sName)   { strcpy(m_sName, sName); }
    const char*          GetName()const               { return m_sName; }
protected:
    char     m_sName[BUFSIZ];
};

class CTyounan : public CChild{
public:
    const char* GetPosition()const       { return "長男"; }
};

class CJinan : public CChild{
public:
    const char* GetPosition()const       { return "次男"; }
};

class CSannan : public CChild{
public:
    const char* GetPosition()const       { return "三男"; }
};
```

```
#define KYOUDAI  3

void main()
{
    CTyounan taro;
    CJinan jiro;
    CSannan sabu;

    taro.SetName("太郎");
    jiro.SetName("次郎");
    sabu.SetName("三郎");

    CChild* pKyoudai[KYOUDAI];
    pKyoudai[0] = &taro;
    pKyoudai[1] = &jiro;
    pKyoudai[2] = &sabu;

    printf("誰の身分が知りたい?¥n");
    for(int i = 0; i < KYOUDAI; i++){
        printf("%d:%s¥n", i, pKyoudai[i]->GetName());
    }
    printf("その他:表示しない¥n");

    int nSel(0);
    scanf("%d", &nSel);

    if(KYOUDAI < nSel){
        printf("ヤツの身分は%sです", pKyoudai[nSel]->GetPosition());
    }
}
```

結果：

```
誰の身分が知りたい?
0:太郎
1:次郎
2:三郎
その他:表示しない
2
ヤツの身分は三男です
```