Ｃ＋＋ワンポイントレッスン

親クラスの非仮想デストラクタについて

ソフトウエア研究会

基底クラスのデストラクタを仮想(virtual)にする理由

- C++の入門書には、必ず「基底クラスのデストラクタはvirtualにしよう」と書いてある。
- 理由は、サブクラスのオブジェクトを基底クラスのポインタとして削除した場合、メモリリークが発生するためである。

```
// 非仮想な基底クラス
class Super {
public:
  Super(){}
  ~Super(){} // virtualでない
};
// サブクラス
class Sub : public Super {
public:
  Sub()
    : m_sub(new int[10])
  {}
  ~Sub() {
    delete [] m_sub;
  }
private:
  int* m_sub;
};
```

```
Super* super = new Sub; // 基底クラスのポインタとしてsubを生成
...
delete super; // Superのデストラクタは仮想ではないため、
              // Subのデストラクタは呼ばれない。(コンパイラは
              // superが実はSubのポインタだということを知るす
              // べがない)
Sub::m_subがメモリリークする!
```

```
// もしSuperのデストラクタが仮想の場合、
Super* super = new Sub; // 基底クラスのポインタとしてsubを生成
                        // superには仮想関数を処理するための
                        // v-tableという領域(の*)が付加される
...
delete super; // Superのデストラクタは仮想なので、v-tableを
              // 参照し、Subのデストラクタが呼ばれた後に
              // Superのデストラクタが呼ばれる。したがって、
              // メモリリークは発生しない。
```

仮想デストラクタがあると何が起こるか？

C++のメモリ管理と仮想関数の仕組み

(緑の文字はコンパイラによって自動的に追加されるコード)

```
//  クラス定義
class Super {
  struct _vtable_t {
    void (*destructor)();
  }
  static _vtable_t _vtable = { &~Super };
  _vtable_t* _p_vtable;
public:
  Super(){}
  virtual ~Super(){}
private:
  int m_super;
};
// サブクラス
class Sub : public Super {
static _vtable_t _vtable = { &~Sub };
public:
  Sub() : m_sub(new int[10]) {}
  ~Sub() { delete [] m_sub; }
private:
  int* m_sub;
};
```

```
// newとdelete時の動作

Super* super = new Sub;
super = (Super*)malloc(sizeof(Sub));
super->_p_vtable = &Sub::_vtable;
Sub::Sub(super);  // コンストラクタを呼び出す
.....
delete super;
super->_p_vtable->destructor(); // Sub::~Sub()が呼ばれる
free((void*)super); // メモリ管理システムにより、malloc
                    // したときのメモリブロックが解放される
```

仮想デストラクタがあると何が起こるか？

- メモリリークを回避するため、すべての基底クラスは仮想デストラクタを持つべきである。
- しかし、C++では一つでも仮想関数があると、一般的にオーバーヘッドが発生する。

1) クラスオブジェクトのサイズが増える。これは、仮想関数を処理するためにv-tableという領域が必要になり、ここにアクセスするための情報(通常はポインタ)がオブジェクトに付加されるため。

2) 適切な関数を呼び出すためにv-tableを参照するため、関数コールの処理時間が増加する。

オーバヘッドを回避するにはどうすればよいか？

- 非常に多くの小さなクラスオブジェクトがある場合、メモリサイズや関数コールのオーバーヘッドが無視できなくなる場合がある。
- しかし、仮想関数の無い基底クラスを定義すると、メモリリークの問題が発生する。

オーバヘッドを回避する方法

1) 仮想関数の無い基底クラスを用意し、すべての継承クラスからデストラクタを排除し、メンバ変数を全く持たないか、クラスオブジェクトではないプリミティブな変数のみで構成する。 → 現実的に制約が多すぎて、もはやC++のプログラムとは言えない。しかも、言語仕様としては安全は保障されていない。(一般的なメモリ管理機構ならほぼメモリリークは発生しない)

2) 仮想関数の無い基底クラスを用意し、基底クラスのポインタをdeleteできなくする。→ 基底クラスのデストラクタを protected: にする。これにより、基底クラスのポインタを削除できない不自由があるが、メモリリークの心配はなくなる。

3) 専用のアロケーターを使ってクラスオブジェクトを生成する。(boostのpoolなど) これにより、メモリ管理をクラス外で行えるため、メモリリークを回避できる。(ただし、継承クラスの実装には必ず用意されたアロケータを使うという制約が発生する)

結論

- 通常は、基底クラスのデストラクタは、仮想にする。
- 仮想関数を一つも持たない基底クラスで、速度やメモリサイズに対して要求が厳しい場合は、デストラクタをprotected:にし、基底クラスでのdeleteを抑制する。
- 基底クラスでのdeleteが不可欠な場合は、boost::poolの様な専用のアロケータを使用し、メモリ管理を別途行う。この場合はデストラクタはprivateかprotectedにする。
- 基底クラスのpublicな非仮想デストラクタは使用禁止！