nativeの基礎知識
「ポインタ」てなによ!?

わんくま同盟茶藝部顧問
Minrosoct MVP for Visual C++ 2004-

$\varepsilon\pi\iota\sigma\tau\eta\mu\eta$ episteme@wankuma.com

「ポインタ」てなによ!?

- わかんなくていいです。
- メンドクセーってことだけ知ってください。

C++におけるポインタ

- C#(てか.NET)には値型と参照型があります。
- 値型は struct / 参照型 は class

- naitiveの代表(?)C++には
  型としての参照はありません。
- あるのは値型のみ。class でさえも。

ポインタのつかいかた。

```
class Hoge {
public:
   void func();
   …
};

int main() {
   Hoge* p = new Hoge();
   p->func();
   delete p;
}
```

ポインタ: 血の掟

借りたものは返せ

.NET だと使わなくなったヤツはほっとけば勝手に回収してくれるんですけどねー

ポインタ: 掟を破ると

なにが起こるか わからない

現象は、たいてい原因とは別のところに現れます。
ものすごく見つけにくい厄介なバグです。

よくやる掟破り ： 返し忘れ

```
int main() {
   Hoge* p;
   p = new Hoge();
   … // いろいろあって
}
```

Java/.NET 屋さんは delete を忘れます。

よくやる掟破り ： 紛失

```
int main() {
   Hoge* p;
   p = new Hoge();
   … // いろいろあって
   p = new Hoge();
   …
   delete p;
}
```

最初に借りたHogeが迷子になってます。

よくやる掟破り ： 返し過ぎ

```
void f(Hoge* p) {
  // いろいろあって
  delete p;
}
int main() {
  Hoge* p;
  p = new Hoge();
  f(p);
  delete p;
}
```

mainの最後でもっぺんdeleteしちゃってます。

よくやる掟破り ： 未練たらたら

```
int main() {
   Hoge* p;
   p = new Hoge();
   … // いろいろあって
   delete p;
   … // いろいろあって
   p->func();
}
```

返したからには使っちゃダメですよー。

よくやる掟破り ： 強欲

```
int main() {
   Hoge* p;
   p = new Hoge[10](); // 10個借りる
   p[10].func();
   delete[] p; // まとめて返す
}
```

使えるのはp[0]～p[9]の10個!

めんどくせー最大の要因は

いちいち返さにゃならん。
必ず1回だけ。

0回でもダメ。2回でもダメ。
必ず1回だけ、
責任もって返さにゃならん。

あまりにめんどくせーのでJava/.NETではdelete不要にしちゃいました。

誰が返す!?

```
int main() {
   Hoge* p;
   p = new Hoge();
   Hoge* q = p; // ポインタのコピー
   p->func();
   // delete p しちゃダメよね。
   q->func();
   delete q;
}
```

最後に使った人が返します。

かしこいポインタ(のようなもの)

```
template<T> class shared_ptr {
  T* body;
  int* owner;
public:
  shared_ptr(T* p) { // コンストラクタ
    body = p;
    owner = new int;
    *owner = 1;
  }
  T* get() { return body; }
  …
}
```

ownerは使ってる人の数。

かしこいポインタ(のようなもの)

```
template<T> class shared_ptr {
  T* body;
  int* owner;
public:
  ~shared_ptr() { // デストラクタ
    if ( --*owner == 0 ) {
      delete body;
      delete owner;
    }
  }
  …
}
```

自分が最後の一人なら、返却します。

かしこいポインタ(のようなもの)

```
template<T> class shared_ptr {
   T* body;
   int* owner;
public:
   shared_ptr& operator=(const shared_ptr&)
   { // コピーされたら、元持ってたのを手放して
      // 新たに持ち換える
   }
   …
}
```

こんなのも実装せにゃなりません。

かしこいポインタ(のようなもの)

```
int main() {
  shared_ptr<Hoge> p(new Hoge());
  {
    shared_ptr<Hoge> q = p; // ここで利用者は二人。
    q.get()->func();
  } // qがいなくなって利用者は一人。
  p.get()->func();
  …
} // pがいなくなって利用者は0。
```

自分が最後の一人なら、返却します。

かしこいポインタ(のようなもの)

```
int main() {
  shared_ptr<Hoge> p(new Hoge());
  shared_ptr<Hoge> q = p; // ここで利用者は二人。
  p = shared_ptr<Hoge>(new Hoge()); // ポインタ持ち換え
  p->func();
  q->func();
  …
}
```

こんなことも想定しないとね。

僕のおはなしはここまで。

こんなの作れば、deleteしなくて済みます。
さほどにめんどくさくないですね。

こんなのを
「参照カウンタ」
っていいます。

つづきは~~Webで~~とっちゃんが！