template (C++)

RAPT（山口祐介）

http://rapt21.com

http://blogs.wankuma.com/rapt/

- 対象レベル：1クマー
  > テーマに対しての基礎知識もない方向け
  > これから取り組んでみる方向け

- 処理系：Windows XPsp2 / VC++6sp6, VC++.NET2003sp1

- 肴：WTL

- template の概念は
  ビジネス文書のテンプレートと同じ

- 規定のフォーマットがあって、使用するときに
  主題となるものをあてはめることで完成

たとえば、ある 2 つの同じ型の変数同士の値を交換する関数 Swap を考えてみる

【 int 型の場合】

```
void Swap( int& a, int& b ) {
    int tmp = a;  a = b;  b = tmp;
}
```

```
int main() {
    int a = 1, b = 3;  Swap( a, b );
    cout << “a = ” << a << “, b = ” << b << endl;
}    // a = 3, b = 1
```

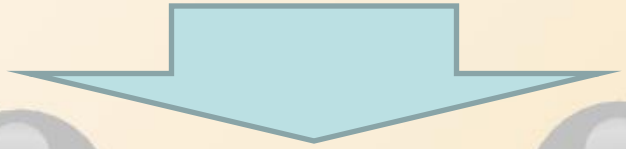
【 double 型の場合】

```
void Swap( double& a, double& b )
{
    double tmp = a;  a = b;  b = tmp;
}
```

【 string 型の場合】

```
void Swap( string& a, string& b )
{
    string tmp = a;  a = b;  b = tmp;
}
```

- 型が増えるごとに関数も増やさないと！
- アルゴリズムを変えると全部直さないと！



- 変化しているのは、int だったり、double だったり、string だったりと型のみ

- アルゴリズムは決まっている
- 使用する型はあとから決めればよい

- 関数名の前に template< typename T > を追加
- 今まで int などといった型名があった場所を T に置き換える

```
template< typename T >
void SwapT( T& a, T& b )
{
    T tmp = a;  a = b;  b = tmp;
}
```

～コラム1～：2クマー

- template 関数宣言での記述においては
  template< typename T > でも
  template< class T > でも
  どちらでもよい

○ template< typename T >
  void SwapT(T& a, T& b)

○ template< class T >
  void SwapT(T& a, T& b)

～コラム2～：2クマー

- 予約語 typename は、その識別子が型名であることを明示するキーワード

template< typename T >

- このテンプレート関数で後付けする部分として、識別子 T を宣言
- そして、この T は型名であることをコンパイラに教える

テンプレートクラスをみてみよう

```
template< typename T, size_t N >
class MyArray
{
protected:
    T    m_arr[ N ];

public:
    MyArray() { }
```

```
    const T& operator[]( size_t index ) const {
        return m_arr[ index ];
    }
    T& operator[]( size_t index ) {
        return m_arr[ index ];
    }
    size_t length() const {
        return N;
    }
};
```

```
int main() {
    MyArray< int, 10 > arr;
    arr[ 0 ] = 5;
    cout << arr[ 0 ] << endl;          // 5
    cout << arr.length() << endl;      // 10

    MyArray< string, 3 > arr2;
    arr2[ 0 ] = "test";
    cout << arr2[ 0 ] << endl;         // test
    cout << arr2.length() << endl;     // 3
}
```

- テンプレートクラスは後付け部分をクラス全体へと伸長しただけ
- また今回の例のように、テンプレート引数は、typename だけでなく、任意の定数やテンプレート関数などを複数個引数にとることができる

- 実際に template で実装されているソースを見た方が早い

- WTL ＜Windows Template Libraries＞を見てみる
  http://sourceforge.net/project/showfiles.php?group_id=109071

  http://rapt21.com/wtl.php ←**日本語化パッチ**

- HPEN をラップする CPenT クラス

```
template <bool t_bManaged>
class CPenT
{
public:
   HPEN    m_hPen;
   ~CPenT() {
       if(t_bManaged && m_hPen != NULL)
            DeleteObject();
   }
};
```

```
typedef CPenT<false>    CPenHandle;
typedef CPenT<true>     CPen;
```

- テンプレート引数である t_bManaged はそのハンドル HPEN を生存管理すべきかどうかを示す

- template 関数・クラスはコンパイル時に確定する必要があるため、template 引数に変数は指定できない

○
```
const size_t ten = 10;  // 定数なのでOK
MyArray< int, ten > arr;
```

×
```
size_t ten = 10;
MyArray< int, ten > arr;
```

template を使った時の長所

- 共通のアルゴリズムを何度も書かずにすむ

- 実行時高速
  - 大抵インライン展開される
  - 後付け部分についてはコンパイル時に確定するので実行時のオーバーヘッドが少ない

template を使った時の短所

- サイズが大きくなる
  コンパイル時、型ごとに関数・クラスの複製が作成されるため

- LIB や DLL などオブジェクトとして提供できない
  オブジェクトとしてはコンパイル時に確定するためヘッダ群を提供する必要がある

次なるステップの紹介

- template の特殊化
  ある特定の template 引数のときに処理を変更できる

- template クラスの継承
  →仮想関数のような仕組みをテンプレートで表現できる
  →WTL:　T* pT = static_cast<T*>(this);