交通・電力・環境システム研究室 プログラミング演習 2008 年度　第 5 回

デザインパターン，継承など

2008. 6. 27. 高木　亮

1　XML ファイルを読み込む方法 ／ 継承 ／ デザインパターン

1-1　前回(第 4 回)に引き続き，参考文献リストを取り扱うことを考える。

1-2　前回は，「参考文献リスト」，および「参考文献エントリ」を定義する XML タグを設計してみた。このような XML タグのそれぞれについて，対応するクラスを設計することができる。XML ファイルを読む際には，各タグのひとつひとつに対応するクラスのインスタンスを生成してゆけばよい（すべてのタグについてそうするのは必ずしも効率的ではないが，効率を別にすればそのようにクラスを設計することは可能である)。
これを，前回作成した XML ファイル references-list.xml の例に則して説明しよう。

1-3　このファイルの第 1 行目にある「XML 宣言」は無視してよい。すると，この XML ファイルの形式では，ひとつの「参考文献リスト」タグ（<references_list> タグ）が最上位にあるのみである。従って，この XML ファイルを読み込めば，これに対応するクラスのインスタンスがひとつ生成されるようにプログラムを書けばよさそうである。タグ名が<references_list>であるので，クラス名をとりあえず ReferencesList とすることにしよう。

1-4　前回紹介した手法で，TinyXML ライブラリによって，前回作成した references-list.xml ファイルをパース(解析)すると，大量のノード（DOM ノードなどと称する）が生成される。この XML ドキュメントそのものがひとつの DOM ノードだが，そのノードに「直属する」ノード（子ノード）が 3 つだけ存在する。それらは XML 宣言に対応するノード，コメントに対応するノード，そして<references_list>タグに対応するノードである。

1-5　コメント部分を省略すれば，この XML ファイルの構造はつぎのように簡略化して表現できる:

```
<?xml version="1.0" encoding="UTF-8"?>
<references_list>
  (なかみ)
</references_list>
```

このようなファイルを TinyXML ライブラリでパースすれば，ドキュメントノードの子ノードとして XML 宣言および<references_list>タグにそれぞれ対応

する DOM ノードがひとつずつできる。この DOM ノードは TinyXML ライブラリにおいては TiXmlNode クラスオブジェクトへのポインタ（`TiXmlNode *` 型）として管理され，アクセスすることができる。従って，`<references_list>` タグから ReferencesList クラスのオブジェクトを生成するためには，ReferencesList クラスが`<references_list>`タグに対応する DOM ノードを指す `const TiXmlNode *` 型変数ひとつを引数としてとるコンストラクタをもつようにクラスを設計しておけばよい。そのルールをクラスの設計が満たしている限り，プログラマは XML ファイルをパースしたのち`<references_list>`タグに対応する DOM ノードを見つけ，それをもとに ReferencesList オブジェクトを容易に生成できる。

そのようなプログラムのメインルーチンは，reflist2-main.cpp（添付）のように書ける[1]。

このソースコードにおいては，クラス ReferencesList について，コンストラクタに関する前提条件のほか，以下の 2 つの条件も満たされていることを仮定している：(1) ReferencesList クラスの宣言は reflist2-ReferencesList.h という名前のヘッダファイルにて行われていること，そして(2) ReferencesList クラスの内容を標準出力に書き出すための << 演算子も定義されていること。

このファイルは，`<references_list>`タグの「なかみ」（属性や子要素）がどのように設計されていようとも，当該 DOM ノードを与えられたとき ReferencesList クラスのコンストラクタが正しく動作してオブジェクトを生成してくれる限り変更の必要がないことに注意したい。

1-6 さて，これで ReferencesList クラスの生成を「指示」するルーチンはできたことになる。そこで，こんどは ReferencesList クラスを実際に設計してみよう。

ReferencesList クラスに対応する`<references_list>`タグは，その要素として任意の数の`<reference_entry>`タグを持つことができる。従って，`<reference_entry>`タグに対応するのが ReferenceEntry クラスであるとするなら，ReferencesList オブジェクトは ReferenceEntry クラスのオブジェクト（もしくはそれへのポインタ）を任意の数だけ保持することができなければならない。

最近の C++では，そのような目的に大変便利な STL ライブラリが標準で使えるようになっているから，その STL ライブラリのなかから適切な「コンテナ」を

[1] 今回のサンプルプログラムはすべて日本語をコメントに入れてあるが，日本語コメントをエディタで入力する際には，文字コードを「UTF-8」にすることを勧める。Meadow でこれを行うためには，入力中に C-x を押し，その後リターンキーを 1 回押し，さらに f キーを押す（もしくはマウスで Options メニュー→Mule (Multilingual Environment)→Set Coding Systems→For Saving This Buffer を選択）と，coding system として何を選ぶか尋ねられるので，utf-8 と入力してリターンキーを押せばよい。他のエディタについてはそれぞれのマニュアル等を参照のこと。

使えばよさそうである。今回は，vector テンプレートを使用し，ReferenceEntry へのポインタを任意の数だけ格納することとしよう。

先ほど，reflist2-main.cpp をコードするとき ReferencesList クラスについて仮定した前提と同様に，ReferenceEntry クラスについて 3 つの前提をおいたうえで，reflist2-ReferencesList.h および reflist2-ReferencesList-version-1.cpp をコードしてみよう（reflist2-ReferencesList-version-1.cpp のほうは，あとで 3 つの前提のうちひとつを多少変更するので書き換えることになるため，このようなファイル名にしてある）。3 つの前提とは：(1) ReferenceEntry クラスは，<reference_entry>タグに対応する DOM ノードを指す const TiXmlNode * 型変数ひとつを引数としてとるコンストラクタをもつこと，(2) ReferenceEntry クラスの宣言は reflist2-ReferenceEntry.h という名前のヘッダファイルにて行われていること，そして (3) ReferenceEntry クラスの内容を標準出力に書き出すための<<演算子が定義されていること，である。

完成した reflist2-ReferencesList.h および reflist2-ReferencesList-version-1.cpp のコードも添付されている。これらは，<reference_entry>タグの「なかみ」（属性や子要素）は何であれ動くようになっている。

まず，ReferencesList クラスは vector < ReferenceEntry * > クラスから導出されている（reflist2-ReferencesList.h ファイルの 21 行目を参照のこと）。この vector < ReferenceEntry * > クラスは ReferenceEntry クラスインスタンスへのポインタを複数格納可能なコンテナである。

ReferencesList クラスのコンストラクタの実装（reflist2-ReferencesList-version-1.cpp にある）では，<references_list>タグに対応する DOM ノード（変数名 n_in）の子ノードをひとつひとつチェックしていく（同ファイル 23～38 行目）。C++ の for ループになっており，n_in の最初の子ノードから順番に const TiXmlNode * 型の変数 n_ch に格納され，n_ch に対してチェックがなされる。まず，子ノード n_ch が ELEMENT（タグ要素）でないならば無視する（同 26～29 行目）。次いで，子ノードがタグ要素であっても，<reference_entry>タグ以外であった場合，XML ファイルの書式が間違っているということでエラーメッセージを出力し，プログラムを強制終了させる（同 30～35 行目）。以上のチェックをくぐり抜けたら，n_ch を用いて ReferenceEntry クラスのインスタンスを生成し（同 36 行目），ReferencesList クラス自身に格納する（同 37 行目。なお，ここで用いる push_back() 関数は vector テンプレートで定義されているもので，ひとつの要素を vector コンテナのリストの末尾に追加する機能を持っている）。

前提条件にある ReferencesList クラスオブジェクトの内容を標準出力に書き出すための演算子，operator<<の実装は，同ファイル 46～76 行目にある。

ReferencesList のコンテナ内に ReferenceEntry クラスインスタンスへのポインタがいくつ格納されているかをチェックし，格納されていれば，標準ライブラリの反復子を用いてひとつひとつを書き出していく（67～73 行目）。ここで変数 i が定義されているが，これは第何番目の文献エントリであるかを示すもので，書き出す際に行頭に入れる番号として使用している。

なお, ReferencesList クラスのオブジェクトを破棄する場合に使われるデストラクタが，同ファイルの末尾，84～92 行目に記述されている。

1-7 ここまでは比較的単純であったが, ここからは問題がいくつかある。前回の XML ファイルの設計によれば，`<reference_entry>` タグの子要素として `<book_entry>` タグもしくは `<journal_paper_entry>` タグのいずれかがひとつだけ存在し得る。そのような場合，`<reference_entry>` タグに対応するクラス ReferenceEntry のコンストラクタはどのように設計すればいいだろうか。ここで，`<book_entry>` タグおよび `<journal_paper_entry>` タグに対応するクラスは，それぞれ BookEntry および JournalPaperEntry であるとする。

これまでに定義されたクラス ReferencesList および ReferenceEntry と同様に，BookEntry および JournalPaperEntry の両クラスとも `const TiXmlNode *` 型変数ひとつを引数としてとるコンストラクタをもつものと仮定してみる。その場合，ReferenceEntry のコンストラクタとしては次のような動作をするものが考えられる: (1) ReferenceEntry のコンストラクタに与えられた DOM ノードの子ノードのうち `<book_entry>` タグまたは `<journal_paper_entry>` タグのどちらかに対応するものを探し出す（なかったらエラー），(2) その子ノードが `<book_entry>` タグであれば BookEntry クラスのコンストラクタを，`<journal_paper_entry>` タグであれば JournalPaperEntry クラスのコンストラクタを，それぞれ起動する，そして (3) できあがったインスタンスへのポインタを，ReferenceEntry のメンバ変数に格納する。

1-8 ここで，BookEntry クラスと JournalPaperEntry クラスは異なるクラスではあるが，似通っているものである，ということを指摘することができる。

BookEntry にも JournalPaperEntry にも「タイトル」「著者」「発行年月」という共通のデータ要素が含まれる。そして，これらクラスには，こうしたデータを用いて書誌情報を出力する機能が追加されるだろう。

そうであれば，ちょうど「人間」は「人間」である前に「動物」である，というのと同様, BookEntry も JournalPaperEntry も ReferenceEntry の一種である，というふうにできればいい。

C++で，それを実現する機構として「継承」がある。この機能を使ってみることにしよう。

まず, 継承の基本を理解するために作成した inheritance1.cpp, inheritance2.cpp,

inheritance3.cpp の 3 つのプログラムをみてほしい。これら 3 つのプログラムのどれにおいても Animal という基底クラスが定義され, Cat および Dog というクラスが Animal クラスの派生クラスとして定義されている。これらのクラスにはすべて cry() というメンバ関数があるが，これを実行したとき表示されるメッセージはクラスごとに異なるようになっている。3 つのファイル, inheritance1.cpp, inheritance2.cpp, inheritance3.cpp は, いずれも 1 ファイルのみでコンパイル・実行が可能である。それぞれを実行した結果を，inheritance1.out，inheritance2.out，inheritance3.out として示した。

まず, inheritance1.cpp の main() 関数をみてみると，62～64 行目で各クラスのインスタンスをひとつずつ生成している（ポインタ a, c, d)。これらのポインタを，`const Animal *` 型の別なポインタ（a_1，a_2，a_3）に代入しているのが，66～73 行目である。このとき重要なのが，Cat または Dog は Animal でもあるという性質である。

しかし，inheritance1.cpp では，Animal クラスから導出された Cat または Dog クラスは，Animal クラスにあるのと同名の cry() という関数を別に持っているだけであるから，インスタンスを生成したときには Cat もしくは Dog クラスとして生成されていても，a_2 または a_3 ポインタを通じて Animal クラスのインスタンスとしてアクセスされると，Animal クラスの cry() 関数が呼び出されてしまう（inheritance1.cpp の 96～99 行目，inheritance1.out の 15～18 行目)。逆に，導出クラスである Cat もしくは Dog クラスのポインタを通じて，この基底クラスの cry() 関数を呼び出すこともできる。このためにはスコープ解決演算子（有効範囲解決演算子とも）を利用して, 呼び出したいのが基底クラス Animal の cry() 関数であり，Cat もしくは Dog クラスのそれではないことを明示的に示してやる必要がある（inheritance1.cpp の 100～103 行目，inheritance1.out の 20～23 行目)。

これでは不便であるので，inheritance1.cpp をごくわずか変更したものが inheritance2.cpp である。変更点は 21 行目の 1 カ所だけで, Animal クラスのメンバ関数 cry() の宣言のさいに virtual というキーワードを追加している。このキーワード付きで宣言される関数を仮想関数と呼ぶ。このようにすると，inheritance2.cpp の 96～99 行目のように Animal クラスへのポインタ（a_2, a_3）経由で cry() 関数を呼び出しても，Cat もしくは Dog クラスのそれが実行されるようになる。

もし，Animal クラスのインスタンス化が不要であれば，inheritance3.cpp の 21 行目のように，基底クラスの仮想関数を “= 0” で初期化することもできる。この場合，この仮想関数は「純粋仮想関数」と呼ばれ，純粋仮想関数をひとつでも含むクラスは抽象クラスと呼ばれる。この場合，inheritance2.cpp の 62 行目のよ

うに Animal クラスはインスタンス化することができなくなる。また，inheritance2.cpp の 100～103 行目のようにスコープ解決演算子で基底クラスの cry() 関数を呼び出すといったこともできなくなる（そもそも実装されていないのだから，当然エラーとなる)。

1-9 これをふまえて，文献リストの設計に戻ろう。まず，BookEntry クラスも JournalPaperEntry クラスも ReferenceEntry クラスの派生クラスであると仮定する。そして，これまではひとつの<reference_entry>タグからひとつの ReferenceEntry クラスのインスタンスを生成することを前提に動いてきたが，今回はひとつの<reference_entry>タグから生成されるものは，なかにあるものが<book_entry>タグか<journal_paper_entry>タグかに応じて，BookEntry クラスもしくは JournalPaperEntry クラスのインスタンスをいずれかひとつだけ生成し，ReferenceEntry クラスへのポインタを格納するコンテナにそれを収納することを考えよう。ReferenceEntry クラスは，抽象クラスとして設計する。

従って，最初に考えていた reflist2-ReferencesList-version-1.cpp を少々変更する必要が生じる。

1-10 ここで，RTSS でも使用している Expandable.hh というファイルを使用することにしよう。このファイル（テンプレートクラスが定義されている）を使うと，次のようないいことがある。すなわち，現在は<reference_entry>タグの子要素としては<book_entry>タグ（書籍）か<journal_paper_entry>タグ（学会誌論文）のいずれかだけである。従って，クラスインスタンスを生成する部分では，例えば:

```
ReferenceEntry * x ;
if (タグが book_entry)
  x = new BookEntry (...) ;
else
  x = new JournalPaperEntry (...) ;
```

といったプログラムを書けばよさそうである。しかし，将来は別なタイプの文献エントリを作りたくなるかもしれない（例：インターネットのウェブページ)。その場合，新たな文献エントリに対応するクラスを作成するだけでなく，上記のようにインスタンスを生成しているプログラムの既存部分にも同時に変更を加えることが必要となる。これは多くの場合煩雑な作業を伴う。

Expandable.hh を利用すれば，あらたなエントリのクラスを追加したい場合，その追加クラスの分だけコードを追加すればよく，既存のプログラムは一切コードの追加や変更が必要なくなる。

1-11 Expandable.hh を使って，reflist2-ReferencesList-version-1.cpp を書き換え，

reflist2-ReferencesList.cpp としてみよう。変更するのは…-version-1.cpp の 36・37 行目である。この部分は，変更後の reflist2-ReferencesList.cpp ではだいぶ長いコードになる（36〜54 行目）。

36 行目で宣言・初期化される sw_end という変数は，ひとつの ReferenceEntry オブジェクトを作成すると true にセットされる（初期化時は false）。<reference_entry> タグ内に複数の <book_entry> もしくは <journal_paper_entry>などのタグが存在するのは XML ファイルの設計に反しているが，そのようなファイルを作成してプログラムに読ませた場合，ふたつめのタグを処理するより以前に sw_end が true にセットされることになる。このような場合には，プログラムはエラーメッセージを出力し異常終了するように設計されている（44〜49 行目）。

37〜54 行目の for ループでは，こうしたチェックを行った後，ReferenceEntry オブジェクト（正確に言えば，BookEntry または JournalPaperEntry のいずれか）をひとつだけ生成し，ReferencesList クラスのインスタンスである自分自身に格納する（51〜53 行目）。C++ の new 演算子のかわりに，ここでは Expandable.hh で定義されている create() という関数をポインタ生成に用いている（RefEntryExpander は Expandable <ReferenceEntry, XMLNodePointer> の typedef である）。

1-12 このような方法で生成できる ReferenceEntry クラスはどのように定義したらよいのだろうか。ヘッダファイル reflist2-ReferenceEntry.h およびソースファイル reflist2-ReferenceEntry.cpp をみてほしい。クラス ReferenceEntry（宣言はヘッダの 21〜60 行目）は抽象クラスであり，このクラスから派生する BookEntry および JournalPaperEntry の両クラスとも，ここに記されたメンバ関数群（printAuthors()，printOtherBibInfo()，printPublishedYearMonth()，printTitle()の 4 つ）をいわばオブジェクトと外界を結ぶインタフェースとして持つことになる。

なお，クラスオブジェクトの内容をストリームに書き出すための関数として純粋仮想関数である outputToStream() が用意されている（ヘッダファイル 44 行目およびソースファイル 13〜20 行目）が，これは ReferenceEntry クラスのポインタを経由してこれらオブジェクトにアクセスした際にもきちんと書き出しが行われるようにするためのテクニックである。挿入子の多重定義の方法と関係がある。「明解 C++」では 334 ページ付近に記述があるので参考にされたい。

1-13 こうして宣言された ReferenceEntry クラスからの派生クラスとして BookEntry および JournalPaperEntry の両クラスを宣言・実装すればよいことになる。BookEntry クラスの宣言と実装を見てみよう（ヘッダファイルは reflist2-BookEntry.h，ソースファイルは reflist2-BookEntry.cpp）。

ヘッダの 44～77 行目には，非公開メンバ変数がいくつかある。XML ファイルで指定されたデータは，これらの変数に格納されることになる。内部データが複雑なタグであれば，<author>タグに対応するクラス Author や，<year_month>タグに対応するクラス YearMonth のように 1 タグあたり 1 つのクラスを定義するのがよいだろうが，1 つの文字列変数だけで代表できそうな出版社名等までいちいちクラスを定義することもないと思われるので，ここでは<publisher>および<address>の両タグについては専用のクラスを用意しなかった。<author>タグについては複数記述されることがあり得ると考え，Author * 型の変数を複数格納できる vector コンテナを用いることとした（ヘッダ 59 行目）。

また，Expander.hh を利用する場合，必ず Creator クラスを同時に定義する必要がある。その部分がヘッダの 141～156 行目である。このうち 145 行目にある add_to_creators_list() という関数を実行している部分が Creator クラスの鍵であり，Creator クラスのオブジェクト生成用関数 create() に，ここで指定した文字列と初期化用オブジェクトへの参照を与えると，所望のオブジェクトを生成することができるようになる(従って，この add_to_creators_list() 関数に与える文字列は，各クラスについて独自のもの，つまり他クラスと一致しないものとする必要がある。また，Creator クラスはグローバル変数として[2]，最低ひとつのインスタンスをプログラムの本体部分が走り出す前に生成しておく必要がある。これを行うためのファイルとして，reflist2-BookEntryCreator.cpp というファイルを作っておく[3]。

では，ソースファイル reflist2-BookEntry.cpp をみてみよう。最初の長い関数（16～164 行目）がコンストラクタであり，XMLNodePointer 型（const TiXmlNode * 型の別名）の変数をひとつ初期化用に受け取り，これを用いてオブジェクトを作成する。コンストラクタ以外に，仮想メンバ関数の実装定義，挿入子やデストラクタの定義などが続いている。

コンストラクタ内では，タグが不正に複数存在するとか，必要なタグが存在しないとかのエラーをチェックしつつ，以下のような流れでデータを取り込んでいく。

(1) <title>タグの処理（35～67 行目）…タイトル文字列を取得し，title_str メンバ変数に格納する。タイトル文字列は name 属性に格納されているので，属性を読みとる作業を行っている（44～66 行目）。

(2) <author>タグの処理（68～72 行目）… <author>タグは専用のクラス

[2] グローバル変数でもよいが，実際には，reflist2-BookEntryCreator.cpp では「名前なし名前空間」を使って，ひとつの Creator クラスオブジェクトを生成している。ソースプログラムを実際に自分で読んでみよ。

[3] もし「ダイナミックリンクライブラリ」を作る場合，このような Creator クラスのインスタンスを含むファイルはライブラリに入れることができないので，ライブラリ生成時には注意する必要がある。

Author のコンストラクタに処理させている。生成されたインスタンスへのポインタは，authors 変数（vector コンテナ）に push_back() 関数で格納している。

(3) <publisher>タグの処理（73～105 行目）… <title>タグとほぼ同様である（全体が 33 行あるのまで同じだ！）。

(4) <address>タグの処理（106～138 行目）…これまた<title>タグとほぼ同様である（全体が 33 行あるのまで同じだ！）。

(5) <year_month>タグの処理（139～149 行目）… <author>タグと同様，専用のクラス YearMonth で処理する。ただし，<year_month>タグは 1 つだけしか書けないので，複数回書いた場合エラーメッセージを書き出して異常終了するコード（141～147 行目）が追加されている。

1-14 前項と同様に，reflist2-JournalPaperEntry.cpp のコンストラクタの構造を調べてみよ。特に，<volume_number>タグおよび<pages>タグについては，<title>タグなどと異なり数字を属性から取り出していることに注目されたい。

1-15 最後に，Author クラスおよび YearMonth クラスの宣言と実装を調べてみよ（ファイルは，ヘッダがそれぞれ reflist2-Author.h および reflist2-YearMonth.h，ソースがそれぞれ reflist2-Author.cpp および reflist2-YearMonth.cpp）。これまた，似たような構造のコンストラクタが発見できることであろう。

1-16 これをコンパイルし，実行してみよ。コンパイルには，Makefile-reflist2 が使える。できあがる実行ファイルは reflist2.exe である。XML ファイル references-list.xml を書き換えれば, reflist2.exe 実行時の出力結果も異なるはずであるから，いろいろ試してみてほしい。ただし，この段階では XML ファイル書き換えの際はローマ字のみを利用すること。

1-17 <reference_entry>タグの子タグとして使える新たな参考文献エントリタグを定義して，それに対応するクラスを実装してみよ。

なお，Expandable.hh で実装されているような手法は，いわゆる「デザインパターン」の本に出てくるものに似ている。高木がこの Expandable.hh を実装した時点では，デザインパターンの存在自体を知らなかったのだが，ここで実装されたものは「オブジェクト指向における再利用のためのデザインパターン 改訂版」(ソフトバンククリエイティブ(1999)＝研究室蔵書。デザインパターンとしてもっとも有名な，いわゆる Gang of Four (GoF) デザインパターン本である)にある Factory Method パターンに近いように思われる。こうした本を参照することで，より効率的なソフトウェアを書けるようになる。

当然のことだが，疑問・質問は随時高木まで（今年度は多忙なので，アポイントメントをとること！）。

以上

```
 1 // -*- C++ -*-
 2
 3 #include <iostream>
 4
 5 using std :: cout ;
 6 using std :: cerr ;
 7 using std :: endl ;
 8
 9 void
10 printUsage
11 ( const char * argzero )
12 {
13   cerr << "Usage:" << endl << " " << argzero << endl
14        << " No command line arguments allowed." << endl ;
15 }
16
17
18 class Animal
19 {
20 public :
21   void cry () const
22   { cout << "  I am an animal, I do not know how to cry." << endl ; }
23 } ;
24
25
26 class Cat
27   : public Animal
28 {
29 public :
30   void cry () const
31   { cout << "  I am a cat. Miaow!" << endl ; }
32
33   void feelCold () const
34   { cout << "  I am a Cat. I feel cold. Miaow!" << endl ; }
35 } ;
36
37
38 class Dog
39   : public Animal
40 {
41 public :
42   void cry () const
43   { cout << "  I am a dog. Bow Wow!" << endl ; }
44
45   void feelHot () const
46   { cout << "  I am a Dog. I feel hot. Bow!" << endl ; }
47 } ;
48
49
50 int
51 main
52 ( int argc ,
53   char * * argv )
54 {
55   if ( argc != 1 )
56   {
57     cerr << "Error: too many arguments." << endl ;
58     printUsage ( argv [ 0 ] ) ;
59     return 1 ;
60   }
61
62   Animal * a = new Animal ;
63   Cat * c = new Cat ;
64   Dog * d = new Dog ;
65
66   // An animal (neither cat nor dog) is an animal. Simple.
67   const Animal * a_1 = a ;
68
```

```
 69   // A cat is an animal, too...
 70   const Animal * a_2 = c ;
 71
 72   // A dog is an animal, too...
 73   const Animal * a_3 = d ;
 74
 75   // Each one has its own way to cry...
 76   cout << "Animal: a -> cry () :" << endl ;
 77   a -> cry () ;
 78   cout << "Cat:    c -> cry () :" << endl ;
 79   c -> cry () ;
 80   cout << "Dog:    d -> cry () :" << endl ;
 81   d -> cry () ;
 82
 83   // An animal (neither cat nor dog) cannot feel cold or hot
 84
 85   // A cat can feel cold, but cannot feel hot
 86   cout << endl << "Cat:    c -> feelCold () :" << endl ;
 87   c -> feelCold () ;
 88
 89   // A dog can feel hot, but cannot feel cold
 90   cout << "Dog:    d -> feelHot () :" << endl ;
 91   d -> feelHot () ;
 92
 93   // How about doing the same crying through the Animal * pointer?
 94   cout << endl << "Animal: a_1 -> cry () :" << endl ;
 95   a_1 -> cry () ;
 96   cout << "Animal: a_2 -> cry () : (This is actually a cat)" << endl ;
 97   a_2 -> cry () ;
 98   cout << "Animal: a_3 -> cry () : (This is actually a dog)" << endl ;
 99   a_3 -> cry () ;
100   cout << endl << "Cat:    c -> Animal :: cry () :" << endl ;
101   c -> Animal :: cry () ;
102   cout << "Dog:    d -> Animal :: cry () :" << endl ;
103   d -> Animal :: cry () ;
104 }
```

```
 1 Animal: a -> cry () :
 2   I am an animal, I do not know how to cry.
 3 Cat:    c -> cry () :
 4   I am a cat. Miaow!
 5 Dog:    d -> cry () :
 6   I am a dog. Bow Wow!
 7
 8 Cat:    c -> feelCold () :
 9   I am a Cat. I feel cold. Miaow!
10 Dog:    d -> feelHot () :
11   I am a Dog. I feel hot. Bow!
12
13 Animal: a_1 -> cry () :
14   I am an animal, I do not know how to cry.
15 Animal: a_2 -> cry () : (This is actually a cat)
16   I am an animal, I do not know how to cry.
17 Animal: a_3 -> cry () : (This is actually a dog)
18   I am an animal, I do not know how to cry.
19
20 Cat:    c -> Animal :: cry () :
21   I am an animal, I do not know how to cry.
22 Dog:    d -> Animal :: cry () :
23   I am an animal, I do not know how to cry.
```

```
 1 // -*- C++ -*-
 2
 3 #include <iostream>
 4
 5 using std :: cout ;
 6 using std :: cerr ;
 7 using std :: endl ;
 8
 9 void
10 printUsage
11 ( const char * argzero )
12 {
13   cerr << "Usage:" << endl << " " << argzero << endl
14        << " No command line arguments allowed." << endl ;
15 }
16
17
18 class Animal
19 {
20 public :
21   virtual void cry () const
22   { cout << "  I am an animal, I do not know how to cry." << endl ; }
23 } ;
24
25
26 class Cat
27   : public Animal
28 {
29 public :
30   void cry () const
31   { cout << "  I am a cat. Miaow!" << endl ; }
32
33   void feelCold () const
34   { cout << "  I am a Cat. I feel cold. Miaow!" << endl ; }
35 } ;
36
37
38 class Dog
39   : public Animal
40 {
41 public :
42   void cry () const
43   { cout << "  I am a dog. Bow Wow!" << endl ; }
44
45   void feelHot () const
46   { cout << "  I am a Dog. I feel hot. Bow!" << endl ; }
47 } ;
48
49
50 int
51 main
52 ( int argc ,
53   char * * argv )
54 {
55   if ( argc != 1 )
56   {
57     cerr << "Error: too many arguments." << endl ;
58     printUsage ( argv [ 0 ] ) ;
59     return 1 ;
60   }
61
62   Animal * a = new Animal ;
63   Cat * c = new Cat ;
64   Dog * d = new Dog ;
65
66   // An animal (neither cat nor dog) is an animal. Simple.
67   const Animal * a_1 = a ;
68
```

```
 69   // A cat is an animal, too...
 70   const Animal * a_2 = c ;
 71
 72   // A dog is an animal, too...
 73   const Animal * a_3 = d ;
 74
 75   // Each one has its own way to cry...
 76   cout << "Animal: a -> cry () :" << endl ;
 77   a -> cry () ;
 78   cout << "Cat:    c -> cry () :" << endl ;
 79   c -> cry () ;
 80   cout << "Dog:    d -> cry () :" << endl ;
 81   d -> cry () ;
 82
 83   // An animal (neither cat nor dog) cannot feel cold or hot
 84
 85   // A cat can feel cold, but cannot feel hot
 86   cout << endl << "Cat:    c -> feelCold () :" << endl ;
 87   c -> feelCold () ;
 88
 89   // A dog can feel hot, but cannot feel cold
 90   cout << "Dog:    d -> feelHot () :" << endl ;
 91   d -> feelHot () ;
 92
 93   // How about doing the same crying through the Animal * pointer?
 94   cout << endl << "Animal: a_1 -> cry () :" << endl ;
 95   a_1 -> cry () ;
 96   cout << "Animal: a_2 -> cry () : (This is actually a cat)" << endl ;
 97   a_2 -> cry () ;
 98   cout << "Animal: a_3 -> cry () : (This is actually a dog)" << endl ;
 99   a_3 -> cry () ;
100   cout << endl << "Cat:    c -> Animal :: cry () :" << endl ;
101   c -> Animal :: cry () ;
102   cout << "Dog:    d -> Animal :: cry () :" << endl ;
103   d -> Animal :: cry () ;
104 }
```

```
 1 Animal: a -> cry () :
 2   I am an animal, I do not know how to cry.
 3 Cat:    c -> cry () :
 4   I am a cat. Miaow!
 5 Dog:    d -> cry () :
 6   I am a dog. Bow Wow!
 7
 8 Cat:    c -> feelCold () :
 9   I am a Cat. I feel cold. Miaow!
10 Dog:    d -> feelHot () :
11   I am a Dog. I feel hot. Bow!
12
13 Animal: a_1 -> cry () :
14   I am an animal, I do not know how to cry.
15 Animal: a_2 -> cry () : (This is actually a cat)
16   I am a cat. Miaow!
17 Animal: a_3 -> cry () : (This is actually a dog)
18   I am a dog. Bow Wow!
19
20 Cat:    c -> Animal :: cry () :
21   I am an animal, I do not know how to cry.
22 Dog:    d -> Animal :: cry () :
23   I am an animal, I do not know how to cry.
```

```
 1 // -*- C++ -*-
 2
 3 #include <iostream>
 4
 5 using std :: cout ;
 6 using std :: cerr ;
 7 using std :: endl ;
 8
 9 void
10 printUsage
11 ( const char * argzero )
12 {
13   cerr << "Usage:" << endl << " " << argzero << endl
14        << " No command line arguments allowed." << endl ;
15 }
16
17
18 class Animal
19 {
20 public :
21   virtual void cry () const = 0 ;
22 } ;
23
24
25 class Cat
26   : public Animal
27 {
28 public :
29   void cry () const
30   { cout << "I am a cat. Miaow!" << endl ; }
31
32   void feelCold () const
33   { cout << "I am a Cat. I feel cold. Miaow!" << endl ; }
34 } ;
35
36
37 class Dog
38   : public Animal
39 {
40 public :
41   void cry () const
42   { cout << "I am a dog. Bow Wow!" << endl ; }
43
44   void feelHot () const
45   { cout << "I am a Dog. I feel hot. Bow!" << endl ; }
46 } ;
47
48
49 int
50 main
51 ( int argc ,
52   char * * argv )
53 {
54   if ( argc != 1 )
55   {
56     cerr << "Error: too many arguments." << endl ;
57     printUsage ( argv [ 0 ] ) ;
58     return 1 ;
59   }
60
61   // You cannot create an instance of an Animal...
62   Cat * c = new Cat ;
63   Dog * d = new Dog ;
64
65   // A cat is an animal, too...
66   const Animal * a_2 = c ;
67
68   // A dog is an animal, too...
```

```
69   const Animal * a_3 = d ;
70
71   // Each one has its own way to cry...
72   cout << "Cat:    c -> cry () :" << endl ;
73   c -> cry () ;
74   cout << "Dog:    d -> cry () :" << endl ;
75   d -> cry () ;
76
77   // A cat can feel cold, but cannot feel hot
78   cout << endl << "Cat:    c -> feelCold () :" << endl ;
79   c -> feelCold () ;
80
81   // A dog can feel hot, but cannot feel cold
82   cout << "Dog:    d -> feelHot () :" << endl ;
83   d -> feelHot () ;
84
85   // How about doing the same crying through the Animal * pointer?
86   cout << endl << "Animal: a_2 -> cry () : (This is actually a cat)" << endl ;
87   a_2 -> cry () ;
88   cout << "Animal: a_3 -> cry () : (This is actually a dog)" << endl ;
89   a_3 -> cry () ;
90 }
```

```
 1 Cat:    c -> cry () :
 2 I am a cat. Miaow!
 3 Dog:    d -> cry () :
 4 I am a dog. Bow Wow!
 5
 6 Cat:    c -> feelCold () :
 7 I am a Cat. I feel cold. Miaow!
 8 Dog:    d -> feelHot () :
 9 I am a Dog. I feel hot. Bow!
10
11 Animal: a_2 -> cry () : (This is actually a cat)
12 I am a cat. Miaow!
13 Animal: a_3 -> cry () : (This is actually a dog)
14 I am a dog. Bow Wow!
```

```
 1 // -*- C++ -*-
 2
 3 #include <iostream>
 4 #include "reflist2-Author.h"
 5
 6 using std :: ostream ;
 7 using std :: cout ;
 8 using std :: cerr ;
 9 using std :: endl ;
10
11
12
13 // ----
14 // Constructor of class Author.
15 // Author クラスのコンストラクタ.
16 // ----
17 Author :: Author
18 ( const TiXmlNode * n_in )
19   : title_str () ,
20     firstname_str () ,
21     familyname_str ()
22 {
23   // ----
24   // Search for children nodes. Actually there must be no ELEMENT child node.
25   // 子ノードを探索。実際には子ノード中に ELEMENT ノードはあってはならない。
26   // ----
27   for ( const TiXmlNode * n_ch = n_in -> FirstChild () ;
28         n_ch ; n_ch = n_ch -> NextSibling () )
29   {
30     if ( n_ch -> Type () != TiXmlNode :: ELEMENT )
31     {
32       continue ;
33     }
34     cerr << "Error: no child element is allowed in <author> tag." << endl ;
35     exit ( 1 ) ;
36   }
37
38   bool sw_t = false ;
39   bool sw_g = false ;
40   bool sw_f = false ;
41   for ( const TiXmlAttribute * t_a
42           = n_in -> ToElement () -> FirstAttribute () ;
43         t_a ; t_a = t_a -> Next () )
44   {
45     if ( string ( t_a -> Name () ) == "atitle" )
46     {
47       if ( sw_t )
48       {
49         cerr << "Error: duplicate ¥"atitle¥" attribute to tag <author>"
50              << "... internal error?" << endl ;
51         exit ( 1 ) ;
52       }
53       sw_t = true ;
54       title_str = t_a -> ValueStr () ;
55     }
56     else if ( string ( t_a -> Name () ) == "firstname" )
57     {
58       if ( sw_g )
59       {
60         cerr << "Error: duplicate ¥"firstname¥" attribute to tag <author>"
61              << "... internal error?" << endl ;
62         exit ( 1 ) ;
63       }
64       sw_g = true ;
65       firstname_str = t_a -> ValueStr () ;
66     }
67     else if ( string ( t_a -> Name () ) == "familyname" )
68     {
```

```
 69       if ( sw_f )
 70       {
 71         cerr << "Error: duplicate ¥"familyname¥" attribute to tag <author>"
 72              << "... internal error?" << endl ;
 73         exit ( 1 ) ;
 74       }
 75       sw_f = true ;
 76       familyname_str = t_a -> ValueStr () ;
 77     }
 78     else
 79     {
 80       cerr << "Error: attribute ¥"" << t_a -> Name ()
 81            << "¥" not allowed in tag <author>" << endl ;
 82       exit ( 1 ) ;
 83     }
 84   }
 85 }
 86
 87
 88
 89 // ----
 90 // Get full name.
 91 // フルネームを得る。
 92 // ----
 93 string
 94 Author :: getFullName
 95 ()
 96   const
 97 {
 98   return title_str + string ( " " ) + firstname_str + string ( " " )
 99     + familyname_str ;
100 }
101
102
103 // ----
104 // Writing out contents of class Author.
105 // Author クラスの内容の書き出し。
106 // ----
107 ostream &
108 operator<<
109 ( ostream & str_out ,
110   const Author & rl_out )
111 {
112   str_out << rl_out . getFullName () ;
113   return str_out ;
114 }
```

```
 1 // -*- C++ -*-
 2
 3 #ifndef REFLIST2_AUTHOR_H
 4 #define REFLIST2_AUTHOR_H
 5
 6 #include <iostream>
 7 #include <string>
 8 #include "tinyxml.h"
 9
10 using std :: ostream ;
11 using std :: string ;
12
13
14 // ----
15 // Declaration of class Author.
16 // Author クラスの宣言.
17 // ----
18 class Author
19 {
20
21   // ******** //
22   // Friends  //
23   // フレンド //
24   // ******** //
25
26   // ----
27   // Writing out to stdout.
28   // 標準出力への書き出し。
29   // ----
30   friend ostream & operator<< ( ostream & , const Author & ) ;
31
32   // ************************ //
33   // Private member variables //
34   // 非公開メンバ変数         //
35   // ************************ //
36   string title_str ;
37   string firstname_str ;
38   string familyname_str ;
39
40   // ************************ //
41   // Private member functions //
42   // 非公開メンバ関数         //
43   // ************************ //
44
45   // ----
46   // Prohibiting the default and the copy constructors.
47   // デフォルトコンストラクタとコピーコンストラクタの使用禁止
48   // ----
49   explicit Author () ;
50   explicit Author ( const Author & ) ;
51
52 public :
53
54   // ************************ //
55   // Public member functions //
56   // 公開メンバ関数          //
57   // ************************ //
58
59   // ----
60   // Constructor. Argument is const TiXmlNode * that contains <author>.
61   // コンストラクタ。引数は <author> を含む const TiXmlNode *.
62   // ----
63   explicit Author ( const TiXmlNode * ) ;
64
65   // ----
66   // Returns the full name as a string.
67   // フルネームを string 型として返す。
68   // ----
```

```
69   string getFullName () const ;
70
71 } ;
72
73
74
75 // ----
76 // Operator << for writing out the entire Author class object.
77 // Author クラスオブジェクト全体を書き出すための << 演算子。
78 // ----
79 ostream & operator<< ( ostream & , const Author & ) ;
80
81
82
83 #endif // REFLIST2_AUTHOR_H
```

```
 1 // -*- C++ -*-
 2
 3 #include <iostream>
 4 #include <cstdlib>
 5 #include <vector>
 6 #include <string>
 7 #include "reflist2-BookEntry.h"
 8
 9 using std :: ostream ;
10 using std :: vector ;
11 using std :: cerr ;
12 using std :: endl ;
13 using std :: string ;
14
15
16 // ----
17 // Constructor of class BookEntry.
18 // BookEntry クラスのコンストラクタ.
19 // ----
20 BookEntry :: BookEntry
21 ( const XMLNodePointer & n_in )
22   : authors ()
23 {
24   bool sw_title = false ;
25   bool sw_ym = false ;
26   bool sw_pb = false ;
27   bool sw_pa = false ;
28   for ( const TiXmlNode * n_ch = n_in -> FirstChild () ;
29         n_ch ; n_ch = n_ch -> NextSibling () )
30   {
31     if ( n_ch -> Type () != TiXmlNode :: ELEMENT )
32     {
33       continue ;
34     }
35     if ( string ( n_ch -> Value () ) == "title" )
36     {
37       if ( sw_title )
38       {
39         cerr << "Error: two or more <title> tags found in <book_entry>."
40              << endl ;
41         exit ( 1 ) ;
42       }
43       sw_title = true ;
44       bool sw_tf = false ;
45       for ( const TiXmlAttribute * t_a
46               = n_ch -> ToElement () -> FirstAttribute () ;
47             t_a ; t_a = t_a -> Next () )
48       {
49         if ( string ( t_a -> Name () ) == "name" )
50         {
51           if ( sw_tf )
52           {
53             cerr << "Error: duplicate ¥"name¥" attribute to tag <title>"
54                  << "... internal error?" << endl ;
55             exit ( 1 ) ;
56           }
57           sw_tf = true ;
58           title_str = t_a -> ValueStr () ;
59         }
60         else
61         {
62           cerr << "Error: attribute ¥"" << t_a -> Name ()
63                << "¥" not allowed in tag <title>" << endl ;
64           exit ( 1 ) ;
65         }
66       }
67     }
68     else if ( string ( n_ch -> Value () ) == "author" )
```

```
 69      {
 70        Author * a_x = new Author ( n_ch ) ;
 71        authors . push_back ( a_x ) ;
 72      }
 73      else if ( string ( n_ch -> Value () ) == "publisher" )
 74      {
 75        if ( sw_pb )
 76        {
 77          cerr << "Error: two or more <publisher> tags found in <book_entry>."
 78               << endl ;
 79          exit ( 1 ) ;
 80        }
 81        sw_pb = true ;
 82        bool sw_bf = false ;
 83        for ( const TiXmlAttribute * t_a
 84                = n_ch -> ToElement () -> FirstAttribute () ;
 85              t_a ; t_a = t_a -> Next () )
 86        {
 87          if ( string ( t_a -> Name () ) == "name" )
 88          {
 89            if ( sw_bf )
 90            {
 91              cerr << "Error: duplicate ¥"name¥" attribute to tag <publisher>"
 92                   << "... internal error?" << endl ;
 93              exit ( 1 ) ;
 94            }
 95            sw_bf = true ;
 96            publisher_str = t_a -> ValueStr () ;
 97          }
 98          else
 99          {
100            cerr << "Error: attribute ¥"" << t_a -> Name ()
101                 << "¥" not allowed in tag <publisher>" << endl ;
102            exit ( 1 ) ;
103          }
104        }
105      }
106      else if ( string ( n_ch -> Value () ) == "address" )
107      {
108        if ( sw_pa )
109        {
110          cerr << "Error: two or more <address> tags found in <book_entry>."
111               << endl ;
112          exit ( 1 ) ;
113        }
114        sw_pa = true ;
115        bool sw_af = false ;
116        for ( const TiXmlAttribute * t_a
117                = n_ch -> ToElement () -> FirstAttribute () ;
118              t_a ; t_a = t_a -> Next () )
119        {
120          if ( string ( t_a -> Name () ) == "name" )
121          {
122            if ( sw_af )
123            {
124              cerr << "Error: duplicate ¥"name¥" attribute to tag <address>"
125                   << "... internal error?" << endl ;
126              exit ( 1 ) ;
127            }
128            sw_af = true ;
129            paddr_str = t_a -> ValueStr () ;
130          }
131          else
132          {
133            cerr << "Error: attribute ¥"" << t_a -> Name ()
134                 << "¥" not allowed in tag <address>" << endl ;
135            exit ( 1 ) ;
136          }
```

```
137       }
138     }
139     else if ( string ( n_ch -> Value () ) == "year_month" )
140     {
141       if ( sw_ym )
142       {
143         cerr << "Error: two or more <year_month> tags found in <book_entry>."
144              << endl ;
145         exit ( 1 ) ;
146       }
147       sw_ym = true ;
148       ym = new YearMonth ( n_ch ) ;
149     }
150     else
151     {
152       cerr << "Error: tag <" << n_ch -> Value ()
153            << "> tag not allowedfound in <book_entry>." << endl ;
154       exit ( 1 ) ;
155     }
156   }
157   if ( ! ( sw_title && sw_ym && sw_pb && sw_pa )
158        || authors . size () == 0 )
159   {
160     cerr << "Error: incomplete information found in tag <book_entry>."
161          << endl ;
162     exit ( 1 ) ;
163   }
164 }
165
166
167
168 // ----
169 // Writing out authors.
170 // 著者一覧を書き出す。
171 // ----
172 void
173 BookEntry :: printAuthors
174 ( ostream & ostr )
175   const
176 {
177   if ( authors . size () == 1 )
178   {
179     ostr << * ( authors [ 0 ] ) ;
180   }
181   else
182   {
183     vector < Author * > :: const_iterator ia = authors . end () ;
184     -- ia ;
185     vector < Author * > :: const_iterator ib = authors . begin () ;
186     for ( vector < Author * > :: const_iterator ii
187             = authors . begin () ;
188           ii != authors . end () ; ++ ii )
189     {
190       if ( ii != ib )
191       {
192         if ( ii != ia )
193         {
194           ostr << ", " ;
195         }
196         else
197         {
198           ostr << " and " ;
199         }
200       }
201       ostr << ( * ( * ii ) ) ;
202     }
203   }
204 }
```

```
205
206
207
208 // ----
209 // Writing out title.
210 // タイトルを書き出す。
211 // ----
212 void
213 BookEntry :: printTitle
214 ( ostream & ostr )
215   const
216 {
217   ostr << title_str ;
218 }
219
220
221
222 // ----
223 // Writing out the date (month / year) of publication.
224 // 出版年月を出力する。
225 // ----
226 void
227 BookEntry :: printPublishedYearMonth
228 ( ostream & ostr )
229   const
230 {
231   ostr << ( * ym ) ;
232 }
233
234
235
236 // ----
237 // Writing other bibliography information.
238 // 他の書誌情報を出力する。
239 // ----
240 void
241 BookEntry :: printOtherBibInfo
242 ( ostream & ostr )
243   const
244 {
245   ostr << publisher_str << ", " << paddr_str ;
246 }
247
248
249
250 // ----
251 // Writing out contents of class BookEntry.
252 // BookEntry クラスの内容の書き出し。
253 // ----
254 ostream &
255 operator<<
256 ( ostream & ostr ,
257   const BookEntry & be_out )
258 {
259   be_out . printAuthors ( ostr ) ;
260   ostr << ": " ;
261   be_out . printTitle ( ostr ) ;
262   ostr << ", " ;
263   be_out . printOtherBibInfo ( ostr ) ;
264   ostr << ", " ;
265   be_out . printPublishedYearMonth ( ostr ) ;
266   ostr << endl ;
267   return ostr ;
268 }
269
270 ostream &
271 BookEntry :: outputToStream
272 ( ostream & ostr )
```

```
273   const
274 {
275   ostr << ( * this ) ;
276   return ostr ;
277 }
278
279
280
281 // ----
282 // Destructor of class BookEntry.
283 // BookEntry クラスのデストラクタ.
284 // ----
285 BookEntry :: ~BookEntry
286 ()
287 {
288   for ( vector < Author * > :: iterator ii = authors . begin () ;
289         ii != authors . end () ; ++ ii )
290   {
291     delete ( * ii ) ;
292   }
293   if ( ym )
294     delete ym ;
295 }
```

```
 1 // -*- C++ -*-
 2
 3 #include "reflist2-BookEntry.h"
 4
 5 // -----
 6 // Instantiate an object in an unnamed namespace.
 7 // 無名ネームスペースにおいて，オブジェクトをひとつインスタンス化する。
 8 // -----
 9 namespace
10 {
11   BookEntryCreator * x = new BookEntryCreator ;
12 }
```

```
 1 // -*- C++ -*-
 2
 3 #ifndef REFLIST2_BOOKENTRY_H
 4 #define REFLIST2_BOOKENTRY_H
 5
 6 #include <iostream>
 7 #include <string>
 8 #include <vector>
 9 #include "tinyxml.h"
10 #include "reflist2-ReferenceEntry.h"
11 #include "reflist2-Author.h"
12 #include "reflist2-YearMonth.h"
13
14 using std :: ostream ;
15 using std :: string ;
16 using std :: vector ;
17 using std :: cout ;
18 using std :: cerr ;
19 using std :: endl ;
20
21
22
23 // ----
24 // Declaration of class BookEntry.
25 // BookEntry クラスの宣言.
26 // ----
27
28 class BookEntry
29   : public ReferenceEntry ,
30     public RefEntryExpander
31 {
32
33   // ******** //
34   // Friends  //
35   // フレンド //
36   // ******** //
37
38   // ----
39   // Writing out to stdout.
40   // 標準出力への書き出し。
41   // ----
42   friend ostream & operator<< ( ostream & , const BookEntry & ) ;
43
44   // ************************ //
45   // Private member variables //
46   // 非公開メンバ変数         //
47   // ************************ //
48
49   // ----
50   // Title.
51   // タイトル。
52   // ----
53   string title_str ;
54
55   // ----
56   // Authors.
57   // 著者。
58   // ----
59   vector < Author * > authors ;
60
61   // ----
62   // Publisher.
63   // 出版社。
64   // ----
65   string publisher_str ;
66
67   // ----
68   // Address of the publisher.
```

```
69   // 出版社の住所。
70   // ----
71   string paddr_str ;
72
73   // ----
74   // Year and month of publication.
75   // 出版年月。
76   // ----
77   YearMonth * ym ;
78
79   // ********************** //
80   // Private member functions //
81   // 非公開メンバ関数         //
82   // ********************** //
83
84   // ----
85   // Prohibiting the default and the copy constructors.
86   // デフォルトコンストラクタとコピーコンストラクタの使用禁止
87   // ----
88   explicit BookEntry () ;
89   explicit BookEntry ( const BookEntry & ) ;
90
91   // ----
92   // Writing out to stdout. Pure virtual function.
93   // 標準出力への書き出し。純粋仮想関数。
94   // ----
95   ostream & outputToStream ( ostream & ) const ;
96
97 public :
98
99   // ********************** //
100  // Public member functions //
101  // 公開メンバ関数          //
102  // ********************** //
103
104  // ----
105  // Constructor. Argument is XMLNodePointer that contains <book_entry>.
106  // コンストラクタ。引数は <book_entry> を含む XMLNodePointer.
107  // ----
108  explicit BookEntry ( const XMLNodePointer & ) ;
109
110  // ----
111  // Virtual functions that must be implemented.
112  // 実装が必要な仮想関数。
113  // ----
114  void printAuthors ( ostream & ) const ;
115  void printTitle ( ostream & ) const ;
116  void printPublishedYearMonth ( ostream & ) const ;
117  void printOtherBibInfo ( ostream & ) const ;
118
119
120  // ----
121  // Destructor.
122  // デストラクタ。
123  // ----
124  ~BookEntry () ;
125
126 } ;
127
128
129 // ----
130 // Operator << for writing out the entire BookEntry class object.
131 // BookEntry クラスオブジェクト全体を書き出すための << 演算子。
132 // ----
133 ostream & operator<< ( ostream & , const BookEntry & ) ;
134
135
136
```

```
137 // -----
138 // Creator of class BookEntry
139 // BookEntry クラスの Creator.
140 // -----
141 class BookEntryCreator
142   : public RefEntryExpander :: ObjectCreator
143 {
144 public :
145   BookEntryCreator () { add_to_creators_list ( "book_entry" ) ; }
146   BookEntryCreator ( const char * x )
147     : RefEntryExpander :: ObjectCreator () { add_to_creators_list ( x ) ; }
148   BookEntry * create () const
149   {
150     cerr << "Error: BookEntry... default constructor prohibited"
151          << endl ;
152     exit ( 1 ) ;
153   }
154   BookEntry * create ( const XMLNodePointer & x ) const
155   { return new BookEntry ( x ) ; }
156 } ;
157
158
159
160 #endif // REFLIST2_BOOKENTRY_H
```

```
 1 // -*- C++ -*-
 2
 3 #include <iostream>
 4 #include <cstdlib>
 5 #include <vector>
 6 #include <string>
 7 #include <sstream>
 8 #include "reflist2-JournalPaperEntry.h"
 9
10 using std :: ostream ;
11 using std :: vector ;
12 using std :: cerr ;
13 using std :: endl ;
14 using std :: string ;
15 using std :: istringstream ;
16
17
18 // ----
19 // Constructor of class JournalPaperEntry.
20 // JournalPaperEntry クラスのコンストラクタ.
21 // ----
22 JournalPaperEntry :: JournalPaperEntry
23 ( const XMLNodePointer & n_in )
24   : authors ()
25 {
26   bool sw_title = false ;
27   bool sw_ym = false ;
28   bool sw_jn = false ;
29   bool sw_vn = false ;
30   bool sw_pg = false ;
31   for ( const TiXmlNode * n_ch = n_in -> FirstChild () ;
32         n_ch ; n_ch = n_ch -> NextSibling () )
33   {
34     if ( n_ch -> Type () != TiXmlNode :: ELEMENT )
35     {
36       continue ;
37     }
38     if ( string ( n_ch -> Value () ) == "title" )
39     {
40       if ( sw_title )
41       {
42         cerr << "Error: two or more <title> tags found in <journal_paper_entry>."
43              << endl ;
44         exit ( 1 ) ;
45       }
46       sw_title = true ;
47       bool sw_tf = false ;
48       for ( const TiXmlAttribute * t_a
49               = n_ch -> ToElement () -> FirstAttribute () ;
50             t_a ; t_a = t_a -> Next () )
51       {
52         if ( string ( t_a -> Name () ) == "name" )
53         {
54           if ( sw_tf )
55           {
56             cerr << "Error: duplicate ¥"name¥" attribute to tag <title>"
57                  << "... internal error?" << endl ;
58             exit ( 1 ) ;
59           }
60           sw_tf = true ;
61           title_str = t_a -> ValueStr () ;
62         }
63         else
64         {
65           cerr << "Error: attribute ¥"" << t_a -> Name ()
66                << "¥" not allowed in tag <title>" << endl ;
67           exit ( 1 ) ;
68         }
```

```
 69       }
 70     }
 71     else if ( string ( n_ch -> Value () ) == "author" )
 72     {
 73       Author * a_x = new Author ( n_ch ) ;
 74       authors . push_back ( a_x ) ;
 75     }
 76     else if ( string ( n_ch -> Value () ) == "journal" )
 77     {
 78       if ( sw_jn )
 79       {
 80         cerr << "Error: two or more <journal> tags found in <journal_paper_entry>."
 81              << endl ;
 82         exit ( 1 ) ;
 83       }
 84       sw_jn = true ;
 85       bool sw_bf = false ;
 86       for ( const TiXmlAttribute * t_a
 87               = n_ch -> ToElement () -> FirstAttribute () ;
 88             t_a ; t_a = t_a -> Next () )
 89       {
 90         if ( string ( t_a -> Name () ) == "name" )
 91         {
 92           if ( sw_bf )
 93           {
 94             cerr << "Error: duplicate ¥"name¥" attribute to tag <journal>"
 95                  << "... internal error?" << endl ;
 96             exit ( 1 ) ;
 97           }
 98           sw_bf = true ;
 99           journal_str = t_a -> ValueStr () ;
100         }
101         else
102         {
103           cerr << "Error: attribute ¥"" << t_a -> Name ()
104                << "¥" not allowed in tag <journal>" << endl ;
105           exit ( 1 ) ;
106         }
107       }
108     }
109     else if ( string ( n_ch -> Value () ) == "volume_number" )
110     {
111       if ( sw_vn )
112       {
113         cerr << "Error: two or more <volume_number> tags found in <journal_paper_entry>."
114              << endl ;
115         exit ( 1 ) ;
116       }
117       sw_vn = true ;
118       bool sw_vol = false ;
119       bool sw_num = false ;
120       for ( const TiXmlAttribute * t_a
121               = n_ch -> ToElement () -> FirstAttribute () ;
122             t_a ; t_a = t_a -> Next () )
123       {
124         if ( string ( t_a -> Name () ) == "volume" )
125         {
126           if ( sw_vol )
127           {
128             cerr << "Error: duplicate ¥"volume¥" attribute to tag "
129                  << "<volume_number>... internal error?" << endl ;
130             exit ( 1 ) ;
131           }
132           sw_vol = true ;
133           istringstream ix ( t_a -> ValueStr () ) ;
134           ix >> j_vol ;
135         }
136         else if ( string ( t_a -> Name () ) == "number" )
```

```
137         {
138           if ( sw_num )
139           {
140             cerr << "Error: duplicate ¥"number¥" attribute to tag "
141                  << "<volume_number>... internal error?" << endl ;
142             exit ( 1 ) ;
143           }
144           sw_num = true ;
145           istringstream iy ( t_a -> ValueStr () ) ;
146           iy >> j_num ;
147         }
148         else
149         {
150           cerr << "Error: attribute ¥"" << t_a -> Name ()
151                << "¥" not allowed in tag <volume_number>" << endl ;
152           exit ( 1 ) ;
153         }
154       }
155       if ( ! ( sw_vol && sw_num ) )
156       {
157         cerr << "Error: not all attributes are specified in tag "
158              << "<volume_number>" << endl ;
159         exit ( 1 ) ;
160       }
161     }
162     else if ( string ( n_ch -> Value () ) == "pages" )
163     {
164       if ( sw_pg )
165       {
166         cerr << "Error: two or more <pages> tags found in <journal_paper_entry>."
167              << endl ;
168         exit ( 1 ) ;
169       }
170       sw_pg = true ;
171       bool sw_ps = false ;
172       bool sw_pe = false ;
173       for ( const TiXmlAttribute * t_a
174               = n_ch -> ToElement () -> FirstAttribute () ;
175             t_a ; t_a = t_a -> Next () )
176       {
177         if ( string ( t_a -> Name () ) == "start" )
178         {
179           if ( sw_ps )
180           {
181             cerr << "Error: duplicate ¥"start¥" attribute to tag "
182                  << "<pages>... internal error?" << endl ;
183             exit ( 1 ) ;
184           }
185           sw_ps = true ;
186           istringstream is ( t_a -> ValueStr () ) ;
187           is >> page_start ;
188         }
189         else if ( string ( t_a -> Name () ) == "end" )
190         {
191           if ( sw_pe )
192           {
193             cerr << "Error: duplicate ¥"end¥" attribute to tag "
194                  << "<pages>... internal error?" << endl ;
195             exit ( 1 ) ;
196           }
197           sw_pe = true ;
198           istringstream ie ( t_a -> ValueStr () ) ;
199           ie >> page_end ;
200         }
201         else
202         {
203           cerr << "Error: attribute ¥"" << t_a -> Name ()
204                << "¥" not allowed in tag <pages>" << endl ;
```

```
205           exit ( 1 ) ;
206         }
207       }
208       if ( ! ( sw_ps && sw_pe ) )
209       {
210         cerr << "Error: not all attributes are specified in tag "
211              << "<pages>" << endl ;
212         exit ( 1 ) ;
213       }
214     }
215     else if ( string ( n_ch -> Value () ) == "year_month" )
216     {
217       if ( sw_ym )
218       {
219         cerr << "Error: two or more <year_month> tags found in <journal_paper_entry>."
220              << endl ;
221         exit ( 1 ) ;
222       }
223       sw_ym = true ;
224       ym = new YearMonth ( n_ch ) ;
225     }
226     else
227     {
228       cerr << "Error: tag <" << n_ch -> Value ()
229            << "> tag not allowedfound in <journal_paper_entry>." << endl ;
230       exit ( 1 ) ;
231     }
232   }
233   if ( ! ( sw_title && sw_ym && sw_jn && sw_vn && sw_pg )
234        || authors . size () == 0 )
235   {
236     cerr << "Error: incomplete information found in tag <journal_paper_entry>."
237          << endl ;
238     exit ( 1 ) ;
239   }
240 }
241
242
243
244 // ----
245 // Writing out authors.
246 // 著者一覧を書き出す。
247 // ----
248 void
249 JournalPaperEntry :: printAuthors
250 ( ostream & ostr )
251   const
252 {
253   if ( authors . size () == 1 )
254   {
255     ostr << * ( authors [ 0 ] ) ;
256   }
257   else
258   {
259     vector < Author * > :: const_iterator ia = authors . end () ;
260     -- ia ;
261     vector < Author * > :: const_iterator ib = authors . begin () ;
262     for ( vector < Author * > :: const_iterator ii
263             = authors . begin () ;
264           ii != authors . end () ; ++ ii )
265     {
266       if ( ii != ib )
267       {
268         if ( ii != ia )
269         {
270           ostr << ", " ;
271         }
272         else
```

```
273         {
274           ostr << " and " ;
275         }
276       }
277       ostr << ( * ( * ii ) ) ;
278     }
279   }
280 }
281
282
283
284 // ----
285 // Writing out title.
286 // タイトルを書き出す。
287 // ----
288 void
289 JournalPaperEntry :: printTitle
290 ( ostream & ostr )
291   const
292 {
293   ostr << title_str ;
294 }
295
296
297
298 // ----
299 // Writing out the date (month / year) of publication.
300 // 出版年月を出力する。
301 // ----
302 void
303 JournalPaperEntry :: printPublishedYearMonth
304 ( ostream & ostr )
305   const
306 {
307   ostr << ( * ym ) ;
308 }
309
310
311
312 // ----
313 // Writing other bibliography information.
314 // 他の書誌情報を出力する。
315 // ----
316 void
317 JournalPaperEntry :: printOtherBibInfo
318 ( ostream & ostr )
319   const
320 {
321   ostr << journal_str << ", Vol. " << j_vol << ", No. " << j_num << ", " ;
322   if ( page_start == page_end )
323   {
324     ostr << "p. " << page_start ;
325   }
326   else
327   {
328     ostr << "pp. " << page_start << " - " << page_end ;
329   }
330 }
331
332
333
334 // ----
335 // Writing out contents of class JournalPaperEntry.
336 // JournalPaperEntry クラスの内容の書き出し。
337 // ----
338 ostream &
339 operator<<
340 ( ostream & ostr ,
```

```
341   const JournalPaperEntry & be_out )
342 {
343   be_out . printAuthors ( ostr ) ;
344   ostr << ": " ;
345   be_out . printTitle ( ostr ) ;
346   ostr << ", " ;
347   be_out . printOtherBibInfo ( ostr ) ;
348   ostr << ", " ;
349   be_out . printPublishedYearMonth ( ostr ) ;
350   ostr << endl ;
351   return ostr ;
352 }
353
354
355 ostream &
356 JournalPaperEntry :: outputToStream
357 ( ostream & ostr )
358   const
359 {
360   ostr << ( * this ) ;
361   return ostr ;
362 }
363
364
365
366
367 // ----
368 // Destructor of class JournalPaperEntry.
369 // JournalPaperEntry クラスのデストラクタ.
370 // ----
371 JournalPaperEntry :: ~JournalPaperEntry
372 ()
373 {
374   for ( vector < Author * > :: iterator ii = authors . begin () ;
375         ii != authors . end () ; ++ ii )
376   {
377     delete ( * ii ) ;
378   }
379   if ( ym )
380     delete ym ;
381 }
```

```
 1 // -*- C++ -*-
 2
 3 #include "reflist2-JournalPaperEntry.h"
 4
 5 // -----
 6 // Instantiate an object in an unnamed namespace.
 7 // 無名ネームスペースにおいて，オブジェクトをひとつインスタンス化する。
 8 // -----
 9 namespace
10 {
11   JournalPaperEntryCreator * x = new JournalPaperEntryCreator ;
12 }
```

```
 1 // -*- C++ -*-
 2
 3 #ifndef REFLIST2_JOURNALPAPERENTRY_H
 4 #define REFLIST2_JOURNALPAPERENTRY_H
 5
 6 #include <iostream>
 7 #include <string>
 8 #include <vector>
 9 #include "tinyxml.h"
10 #include "reflist2-ReferenceEntry.h"
11 #include "reflist2-Author.h"
12 #include "reflist2-YearMonth.h"
13
14 using std :: ostream ;
15 using std :: string ;
16 using std :: vector ;
17 using std :: cout ;
18 using std :: cerr ;
19 using std :: endl ;
20
21
22 // ----
23 // Declaration of class JournalPaperEntry.
24 // JournalPaperEntry クラスの宣言.
25 // ----
26
27 class JournalPaperEntry
28   : public ReferenceEntry ,
29     public RefEntryExpander
30 {
31
32   // ******** //
33   // Friends  //
34   // フレンド //
35   // ******** //
36
37   // ----
38   // Writing out to stdout.
39   // 標準出力への書き出し。
40   // ----
41   friend ostream & operator<< ( ostream & , const JournalPaperEntry & ) ;
42
43   // ************************ //
44   // Private member variables //
45   // 非公開メンバ変数         //
46   // ************************ //
47
48   // ----
49   // Title.
50   // タイトル。
51   // ----
52   string title_str ;
53
54   // ----
55   // Authors.
56   // 著者。
57   // ----
58   vector < Author * > authors ;
59
60   // ----
61   // Journal name.
62   // 論文誌名。
63   // ----
64   string journal_str ;
65
66   // ----
67   // Volume and number.
68   // 巻号。
```

```
 69   // ----
 70   int j_vol ;
 71   int j_num ;
 72
 73   // ----
 74   // Start and end pages.
 75   // 開始・終了ページ。
 76   // ----
 77   int page_start ;
 78   int page_end ;
 79
 80   // ----
 81   // Year and month of publication.
 82   // 出版年月。
 83   // ----
 84   YearMonth * ym ;
 85
 86   // ********************** //
 87   // Private member functions //
 88   // 非公開メンバ関数         //
 89   // ********************** //
 90
 91   // ----
 92   // Prohibiting the default and the copy constructors.
 93   // デフォルトコンストラクタとコピーコンストラクタの使用禁止
 94   // ----
 95   explicit JournalPaperEntry () ;
 96   explicit JournalPaperEntry ( const JournalPaperEntry & ) ;
 97
 98   // ----
 99   // Writing out to stdout. Pure virtual function.
100   // 標準出力への書き出し。純粋仮想関数。
101   // ----
102   ostream & outputToStream ( ostream & ) const ;
103
104 public :
105
106   // ********************** //
107   // Public member functions //
108   // 公開メンバ関数          //
109   // ********************** //
110
111   // ----
112   // Constructor. Argument is XMLNodePointer that contains
113   // <journal_paper_entry>.
114   // コンストラクタ。引数は <journal_paper_entry> を含む XMLNodePointer.
115   // ----
116   explicit JournalPaperEntry ( const XMLNodePointer & ) ;
117
118   // ----
119   // Virtual functions that must be implemented.
120   // 実装が必要な仮想関数。
121   // ----
122   void printAuthors ( ostream & ) const ;
123   void printTitle ( ostream & ) const ;
124   void printPublishedYearMonth ( ostream & ) const ;
125   void printOtherBibInfo ( ostream & ) const ;
126
127
128   // ----
129   // Destructor.
130   // デストラクタ。
131   // ----
132   ~JournalPaperEntry () ;
133
134 } ;
135
136
```

```
137 // ----
138 // Operator << for writing out the entire JournalPaperEntry class object.
139 // JournalPaperEntry クラスオブジェクト全体を書き出すための << 演算子。
140 // ----
141 ostream & operator<< ( ostream & , const JournalPaperEntry & ) ;
142
143
144
145 // -----
146 // Creator of class JournalPaperEntry
147 // JournalPaperEntry クラスの Creator.
148 // -----
149 class JournalPaperEntryCreator
150   : public RefEntryExpander :: ObjectCreator
151 {
152 public :
153   JournalPaperEntryCreator () { add_to_creators_list ( "journal_paper_entry" ) ; }
154   JournalPaperEntryCreator ( const char * x )
155     : RefEntryExpander :: ObjectCreator () { add_to_creators_list ( x ) ; }
156   JournalPaperEntry * create () const
157   {
158     cerr << "Error: JournalPaperEntry... default constructor prohibited"
159          << endl ;
160     exit ( 1 ) ;
161   }
162   JournalPaperEntry * create ( const XMLNodePointer & x ) const
163   { return new JournalPaperEntry ( x ) ; }
164 } ;
165
166
167
168 #endif // REFLIST2_JOURNALPAPERENTRY_H
```

```
 1 // -*- C++ -*-
 2
 3 #include <iostream>
 4 #include <string>
 5 #include <cstdlib>
 6 #include "tinyxml.h"
 7 #include "reflist2-ReferencesList.h"
 8
 9 using std :: cout ;
10 using std :: cerr ;
11 using std :: endl ;
12 using std :: string ;
13
14
15 // The name of the XML file to be read
16 const char * XML_file_name = "references-list.xml" ;
17
18
19 void
20 printUsage
21 ( const char * argzero )
22 {
23   cerr << "Usage:" << endl << " " << argzero << endl
24        << " No command line arguments allowed." << endl ;
25 }
26
27
28 int
29 main
30 ( int argc ,
31   char * * argv )
32 {
33   if ( argc != 1 )
34   {
35     cerr << "Error: too many arguments." << endl ;
36     printUsage ( argv [ 0 ] ) ;
37     return 1 ;
38   }
39
40   TiXmlDocument doc ( XML_file_name ) ;
41   bool loadOkay = doc . LoadFile () ;
42   if ( ! loadOkay )
43   {
44     cerr << "Failed to load file ¥"" << XML_file_name << "¥"" << endl ;
45     exit ( 1 ) ;
46   }
47
48   // -----
49   // The file must have one "root" element of tag <references_list>.
50   // If multiple <references_list> tags exist, only the first one is read.
51   // このXMLファイルには <references_list> というタグがなければならない。
52   // <references_list> タグが複数あれば最初のだけを読む。
53   // -----
54   const TiXmlNode * doc_ch = doc . FirstChild () ;
55   while ( doc_ch && doc_ch -> Type () != TiXmlNode :: ELEMENT )
56   {
57     doc_ch = doc_ch -> NextSibling () ;
58   }
59
60   // -----
61   // If the file is correct, doc_ch must now be the first ELEMENT node.
62   // Now verify if this is of tag <references_list>.
63   // XMLファイルが正しければ, doc_ch は最初の ELEMENT (要素) ノードを指してい
64   // るはずである。これがほんとうに <references_list> タグのノードか確認する。
65   // -----
66   if ( ! doc_ch || doc_ch -> Type () != TiXmlNode :: ELEMENT
67        || string ( doc_ch -> Value () ) != "references_list" )
68   {
```

```
69      cerr << "Error: failed to find element ¥"references_list¥""
70           << " in file ¥"" << XML_file_name << "¥"." << endl ;
71      exit ( 1 ) ;
72    }
73
74    // -----
75    // Initialise a ReferencesList object using this verified doc_ch.
76    // 正しいことを確認した doc_ch で ReferencesList クラスを初期化する。
77    // -----
78    ReferencesList * x = new ReferencesList ( doc_ch ) ;
79    cout << "Created a ReferencesList object." << endl << ( * x ) << endl ;
80
81    exit ( 0 ) ;
82  }
```

```
 1 // -*- C++ -*-
 2
 3 #include <iostream>
 4 #include "reflist2-ReferenceEntry.h"
 5
 6 using std :: ostream ;
 7
 8
 9 // ----
10 // Writing out contents of class ReferenceEntry.
11 // ReferenceEntry クラスの内容の書き出し。
12 // ----
13 ostream &
14 operator<<
15 ( ostream & str_out ,
16   const ReferenceEntry & rl_out )
17 {
18   rl_out . outputToStream ( str_out ) ;
19   return str_out ;
20 }
```

```
 1 // -*- C++ -*-
 2
 3 #ifndef REFLIST2_REFERENCEENTRY_H
 4 #define REFLIST2_REFERENCEENTRY_H
 5
 6 #include <iostream>
 7 #include <vector>
 8 #include "tinyxml.h"
 9 #include "reflist2-ReferenceEntry.h"
10 #include "Expandable.hh"
11
12 using std :: ostream ;
13 using std :: vector ;
14
15
16 // ----
17 // Declaration of abstract class ReferenceEntry.
18 // 抽象クラス ReferenceEntry の宣言.
19 // ----
20
21 class ReferenceEntry
22 {
23
24   // ******** //
25   // Friends  //
26   // フレンド //
27   // ******** //
28
29   // ----
30   // Writing out to stdout.
31   // 標準出力への書き出し。
32   // ----
33   friend ostream & operator<< ( ostream & , const ReferenceEntry & ) ;
34
35   // ************************ //
36   // Private member functions //
37   // 非公開メンバ関数         //
38   // ************************ //
39
40   // ----
41   // Writing out to stdout. Pure virtual function.
42   // 標準出力への書き出し。純粋仮想関数。
43   // ----
44   virtual ostream & outputToStream ( ostream & ) const = 0 ;
45
46 public :
47
48   // ********************** //
49   // Public member functions //
50   // 公開メンバ関数          //
51   // ********************** //
52
53   virtual void printAuthors ( ostream & ) const = 0 ;
54   virtual void printTitle ( ostream & ) const = 0 ;
55   virtual void printPublishedYearMonth ( ostream & ) const = 0 ;
56   virtual void printOtherBibInfo ( ostream & ) const = 0 ;
57
58   virtual ~ReferenceEntry () {}
59
60 } ;
61
62
63
64 // ----
65 // Operator << for writing out the entire ReferenceEntry class object.
66 // ReferenceEntry クラスオブジェクト全体を書き出すための << 演算子。
67 // ----
68 ostream & operator<< ( ostream & , const ReferenceEntry & ) ;
```

```
69
70
71
72 // ----
73 // Define XMLNodePointer as typedef of const TiXmlNode *.
74 // const TiXmlNode * の typedef として XMLNodePointer を定義
75 // ----
76 typedef const TiXmlNode * XMLNodePointer ;
77
78
79
80 // ----
81 // Define RefEntryExpander as a typedef.
82 // RefEntryExpander を typedef で定義。
83 // ----
84 typedef Expandable < ReferenceEntry , XMLNodePointer > RefEntryExpander ;
85
86
87
88 #endif // REFLIST2_REFERENCEENTRY_H
```

```
 1 // -*- C++ -*-
 2
 3 #include <iostream>
 4 #include <cstdlib>
 5 #include <vector>
 6 #include <string>
 7 #include "reflist2-ReferencesList.h"
 8
 9 using std :: ostream ;
10 using std :: vector ;
11 using std :: cerr ;
12 using std :: endl ;
13 using std :: string ;
14
15
16 // ----
17 // Constructor of class ReferencesList.
18 // ReferencesList クラスのコンストラクタ.
19 // ----
20 ReferencesList :: ReferencesList
21 ( const TiXmlNode * n_in )
22 {
23   for ( const TiXmlNode * n_ch = n_in -> FirstChild () ;
24         n_ch ; n_ch = n_ch -> NextSibling () )
25   {
26     if ( n_ch -> Type () != TiXmlNode :: ELEMENT )
27     {
28       continue ;
29     }
30     if ( string ( n_ch -> Value () ) != "reference_entry" )
31     {
32       cerr << "Error: tags other than <reference_entry> not allowed as "
33            << "children of <references_list>" << endl ;
34       exit ( 1 ) ;
35     }
36     bool sw_end = false ;
37     for ( const TiXmlNode * n_gc = n_ch -> FirstChild () ;
38           n_gc ; n_gc = n_gc -> NextSibling () )
39     {
40       if ( n_ch -> Type () != TiXmlNode :: ELEMENT )
41       {
42         continue ;
43       }
44       if ( sw_end )
45       {
46         cerr << "Error: two or more tags not allowed within <reference_entry>"
47              << endl ;
48         exit ( 1 ) ;
49       }
50       sw_end = true ;
51       ReferenceEntry * x
52         = RefEntryExpander :: create ( string ( n_gc -> Value () ) , n_gc ) ;
53       push_back ( x ) ;
54     }
55   }
56 }
57
58
59 // ----
60 // Writing out contents of class ReferencesList.
61 // ReferencesList クラスの内容の書き出し。
62 // ----
63 ostream &
64 operator<<
65 ( ostream & str_out ,
66   const ReferencesList & rl_out )
67 {
68   ReferencesList :: size_type rl_size = rl_out . size () ;
```

```
 69   if ( rl_size == 0 )
 70   {
 71     str_out << "ReferencesList : No entry." << endl ;
 72   }
 73   else
 74   {
 75     str_out << "ReferencesList : " << rl_size ;
 76     if ( rl_size == 1 )
 77     {
 78       str_out << " entry." << endl ;
 79     }
 80     else
 81     {
 82       str_out << " entries." << endl ;
 83     }
 84     int i = 0 ;
 85     for ( ReferencesList :: const_iterator ii = rl_out . begin () ;
 86           ii != rl_out . end () ; ++ ii )
 87     {
 88       ++ i ;
 89       str_out << "[" << i << "] " << ( * ( * ii ) ) ;
 90     }
 91   }
 92   return str_out ;
 93 }
 94
 95
 96
 97 // ----
 98 // Destructor of class ReferencesList.
 99 // ReferencesList クラスのデストラクタ.
100 // ----
101 ReferencesList :: ~ReferencesList
102 ()
103 {
104   for ( ReferencesList :: iterator ii = begin () ;
105         ii != end () ; ++ ii )
106   {
107     delete ( * ii ) ;
108   }
109 }
```

```
 1 // -*- C++ -*-
 2
 3 #ifndef REFLIST2_REFERENCESLIST_H
 4 #define REFLIST2_REFERENCESLIST_H
 5
 6 #include <iostream>
 7 #include <vector>
 8 #include "tinyxml.h"
 9 #include "reflist2-ReferenceEntry.h"
10
11 using std :: ostream ;
12 using std :: vector ;
13
14
15 // ----
16 // Declaration of class ReferencesList.
17 // ReferencesList クラスの宣言.
18 // ----
19
20 class ReferencesList
21   : public vector < ReferenceEntry * >
22 {
23
24   // ******** //
25   // Friends  //
26   // フレンド //
27   // ******** //
28
29   // ----
30   // Writing out to stdout.
31   // 標準出力への書き出し。
32   // ----
33   friend ostream & operator<< ( ostream & , const ReferencesList & ) ;
34
35   // ************************ //
36   // Private member functions //
37   // 非公開メンバ関数         //
38   // ************************ //
39
40   // ----
41   // Prohibiting the default and the copy constructors.
42   // デフォルトコンストラクタとコピーコンストラクタの使用禁止
43   // ----
44   explicit ReferencesList () ;
45   explicit ReferencesList ( const ReferencesList & ) ;
46
47 public :
48
49   // ************************ //
50   // Public member functions //
51   // 公開メンバ関数          //
52   // ************************ //
53
54   // ----
55   // Constructor. Argument is DOM node for tag <references_list>.
56   // コンストラクタ。引数は <references_list> タグに対応する DOM ノード。
57   // ----
58   explicit ReferencesList ( const TiXmlNode * ) ;
59
60   // ----
61   // Destructor.
62   // デストラクタ。
63   // ----
64   ~ReferencesList () ;
65
66 } ;
67
68
```

```
69 // ----
70 // Operator << for writing out the entire ReferencesList class object.
71 // ReferencesList クラスオブジェクト全体を書き出すための << 演算子。
72 // ----
73 ostream & operator<< ( ostream & , const ReferencesList & ) ;
74
75
76 #endif // REFLIST2_REFERENCESLIST_H
```

```
 1 // -*- C++ -*-
 2
 3 #include <iostream>
 4 #include <cstdlib>
 5 #include <vector>
 6 #include <string>
 7 #include "reflist2-ReferencesList.h"
 8
 9 using std :: ostream ;
10 using std :: vector ;
11 using std :: cerr ;
12 using std :: endl ;
13 using std :: string ;
14
15
16 // ----
17 // Constructor of class ReferencesList.
18 // ReferencesList クラスのコンストラクタ.
19 // ----
20 ReferencesList :: ReferencesList
21 ( const TiXmlNode * n_in )
22 {
23   for ( const TiXmlNode * n_ch = n_in -> FirstChild () ;
24         n_ch ; n_ch = n_ch -> NextSibling () )
25   {
26     if ( n_ch -> Type () != TiXmlNode :: ELEMENT )
27     {
28       continue ;
29     }
30     if ( string ( n_ch -> Value () ) != "reference_entry" )
31     {
32       cerr << "Error: tags other than <reference_entry> not allowed as "
33            << "children of <references_list>" << endl ;
34       exit ( 1 ) ;
35     }
36     ReferenceEntry * x = new ReferenceEntry ( n_ch ) ;
37     push_back ( x ) ;
38   }
39 }
40
41
42 // ----
43 // Writing out contents of class ReferencesList.
44 // ReferencesList クラスの内容の書き出し。
45 // ----
46 ostream &
47 operator<<
48 ( ostream & str_out ,
49   const ReferencesList & rl_out )
50 {
51   ReferencesList :: size_type rl_size = rl_out . size () ;
52   if ( rl_size == 0 )
53   {
54     str_out << "ReferencesList : No entry." << endl ;
55   }
56   else
57   {
58     str_out << "ReferencesList : " << rl_size ;
59     if ( rl_size == 1 )
60     {
61       str_out << " entry." << endl ;
62     }
63     else
64     {
65       str_out << " entries." << endl ;
66     }
67     int i = 0 ;
68     for ( ReferencesList :: const_iterator ii = rl_out . begin () ;
```

```
69           ii != rl_out . end () ; ++ ii )
70     {
71       ++ i ;
72       str_out << "[" << i << "] " << ( * ( * ii ) ) ;
73     }
74   }
75   return str_out ;
76 }
77
78
79
80 // ----
81 // Destructor of class ReferencesList.
82 // ReferencesList クラスのデストラクタ.
83 // ----
84 ReferencesList :: ~ReferencesList
85 ()
86 {
87   for ( ReferencesList :: iterator ii = begin () ;
88         ii != end () ; ++ ii )
89   {
90     delete ( * ii ) ;
91   }
92 }
```

```
 1 // -*- C++ -*-
 2
 3 #include <iostream>
 4 #include <sstream>
 5 #include "reflist2-YearMonth.h"
 6
 7 using std :: ostream ;
 8 using std :: istringstream ;
 9 using std :: ostringstream ;
10 using std :: cout ;
11 using std :: cerr ;
12 using std :: endl ;
13
14
15
16 // ----
17 // Constructor of class YearMonth.
18 // YearMonth クラスのコンストラクタ.
19 // ----
20 YearMonth :: YearMonth
21 ( const TiXmlNode * n_in )
22   : year_i ( 0 ) ,
23     month_i ( 0 )
24 {
25   // ----
26   // Search for children nodes. Actually there must be no ELEMENT child node.
27   // 子ノードを探索。実際には子ノード中に ELEMENT ノードはあってはならない。
28   // ----
29   for ( const TiXmlNode * n_ch = n_in -> FirstChild () ;
30         n_ch ; n_ch = n_ch -> NextSibling () )
31   {
32     if ( n_ch -> Type () != TiXmlNode :: ELEMENT )
33     {
34       continue ;
35     }
36     cerr << "Error: no child element is allowed in <author> tag." << endl ;
37     exit ( 1 ) ;
38   }
39
40   bool sw_y = false ;
41   bool sw_m = false ;
42   for ( const TiXmlAttribute * t_a
43           = n_in -> ToElement () -> FirstAttribute () ;
44         t_a ; t_a = t_a -> Next () )
45   {
46     if ( string ( t_a -> Name () ) == "year" )
47     {
48       if ( sw_y )
49       {
50         cerr << "Error: duplicate ¥"year¥" attribute to tag <year_month>"
51              << "... internal error?" << endl ;
52         exit ( 1 ) ;
53       }
54       sw_y = true ;
55       istringstream iss_y ( t_a -> ValueStr () ) ;
56       iss_y >> year_i ;
57     }
58     else if ( string ( t_a -> Name () ) == "month" )
59     {
60       if ( sw_m )
61       {
62         cerr << "Error: duplicate ¥"month¥" attribute to tag <author>"
63              << "... internal error?" << endl ;
64         exit ( 1 ) ;
65       }
66       sw_m = true ;
67       istringstream iss_m ( t_a -> ValueStr () ) ;
68       iss_m >> month_i ;
```

```
 69       if ( month_i <= 0 || month_i > 12 )
 70       {
 71         cerr << "Error: month should be between 1 and 12" << endl ;
 72         exit ( 1 ) ;
 73       }
 74     }
 75     else
 76     {
 77       cerr << "Error: attribute ¥"" << t_a -> Name ()
 78            << "¥" not allowed in tag <year_month>" << endl ;
 79       exit ( 1 ) ;
 80     }
 81   }
 82   if ( ! ( sw_y && sw_m ) )
 83   {
 84     cerr << "Error: not all attributes are specified in tag "
 85          << "<volume_number>" << endl ;
 86     exit ( 1 ) ;
 87   }
 88 }
 89
 90
 91
 92 // ----
 93 // Get "Month Year" string, such as June 2008.
 94 // "June 2008" のような「月 年」文字列を得る。
 95 // ----
 96 string
 97 YearMonth :: getMonthYearString
 98 ()
 99   const
100 {
101   string x = "" ;
102   switch ( month_i )
103   {
104   case 1 :
105     x += "January " ; break ;
106   case 2 :
107     x += "February " ; break ;
108   case 3 :
109     x += "March " ; break ;
110   case 4 :
111     x += "April " ; break ;
112   case 5 :
113     x += "May " ; break ;
114   case 6 :
115     x += "June " ; break ;
116   case 7 :
117     x += "July " ; break ;
118   case 8 :
119     x += "August " ; break ;
120   case 9 :
121     x += "September " ; break ;
122   case 10 :
123     x += "October " ; break ;
124   case 11 :
125     x += "November " ; break ;
126   case 12 :
127     x += "December " ; break ;
128   default :
129     cerr << "Error: month not correct ... internal error" << endl ;
130     exit ( 1 ) ;
131   }
132   ostringstream y ;
133   y << year_i ;
134   return x + y . str () ;
135 }
136 // ----
```

```
137 // Writing out contents of class YearMonth.
138 // YearMonth クラスの内容の書き出し。
139 // ----
140 ostream &
141 operator<<
142 ( ostream & str_out ,
143   const YearMonth & rl_out )
144 {
145   str_out << rl_out . getMonthYearString () ;
146   return str_out ;
147 }
```

```
 1 // -*- C++ -*-
 2
 3 #ifndef REFLIST2_YEARMONTH_H
 4 #define REFLIST2_YEARMONTH_H
 5
 6 #include <iostream>
 7 #include <string>
 8 #include "tinyxml.h"
 9
10 using std :: ostream ;
11 using std :: string ;
12
13
14 // ----
15 // Declaration of class YearMonth.
16 // YearMonth クラスの宣言.
17 // ----
18 class YearMonth
19 {
20
21   // ******** //
22   // Friends  //
23   // フレンド //
24   // ******** //
25
26   // ----
27   // Writing out to stdout.
28   // 標準出力への書き出し。
29   // ----
30   friend ostream & operator<< ( ostream & , const YearMonth & ) ;
31
32   // ************************ //
33   // Private member variables //
34   // 非公開メンバ変数         //
35   // ************************ //
36   int year_i ;
37   int month_i ;
38
39   // ************************ //
40   // Private member functions //
41   // 非公開メンバ関数         //
42   // ************************ //
43
44   // ----
45   // Prohibiting the default and the copy constructors.
46   // デフォルトコンストラクタとコピーコンストラクタの使用禁止
47   // ----
48   explicit YearMonth () ;
49   explicit YearMonth ( const YearMonth & ) ;
50
51 public :
52
53   // ********************** //
54   // Public member functions //
55   // 公開メンバ関数          //
56   // ********************** //
57
58   // ----
59   // Constructor. Argument is const TiXmlNode * that contains <author>.
60   // コンストラクタ。引数は <author> を含む const TiXmlNode *.
61   // ----
62   explicit YearMonth ( const TiXmlNode * ) ;
63
64   // ----
65   // Get "Month Year" string, such as June 2008.
66   // "June 2008" のような「月 年」文字列を得る。
67   // ----
68   string getMonthYearString () const ;
```

```
69
70 } ;
71
72
73
74 // ----
75 // Operator << for writing out the entire YearMonth class object.
76 // YearMonth クラスオブジェクト全体を書き出すための << 演算子。
77 // ----
78 ostream & operator<< ( ostream & , const YearMonth & ) ;
79
80
81 #endif // REFLIST2_YEARMONTH_H
```

```
### -*- makefile -*-
### Makefile for reflist1

PROGRAM       = reflist2

NICE          =

CXX           = $(NICE) g++

CXXFLAGS      = -g -O2 -Wall
LDFLAGS       = -g -O2
CPPFLAGS      =

HDRS          = tinyxml.h Expandable.hh ¥
                reflist2-Author.h reflist2-BookEntry.h ¥
                reflist2-JournalPaperEntry.h reflist2-ReferenceEntry.h ¥
                reflist2-ReferencesList.h reflist2-YearMonth.h

LINKER        = $(NICE) g++

LIBS          =

SRCS          = reflist2-main.cpp ¥
                reflist2-Author.cpp reflist2-BookEntry.cpp ¥
                reflist2-BookEntryCreator.cpp reflist2-JournalPaperEntry.cpp ¥
                reflist2-JournalPaperEntryCreator.cpp ¥
                reflist2-ReferenceEntry.cpp reflist2-ReferencesList.cpp ¥
                reflist2-YearMonth.cpp ¥
                tinyxml.cpp tinyxmlerror.cpp tinyxmlparser.cpp

OBJS          = $(SRCS:%.cpp=%.o)

DEPS          = $(SRCS:%.cpp=%.dd)

MAKEFILES     = $(DEPS)

all:            $(PROGRAM)

include $(MAKEFILES)

%.dd:           %.cpp
                $(CXX) -M $(CPPFLAGS) $< | sed 's/$*.o/& $@/g' > $@

$(PROGRAM):     $(OBJS)
                $(LINKER) $(LDFLAGS) $(OBJS) $(LIBS) -o $(PROGRAM)

clean:;         rm -f $(OBJS) *~

program:        $(PROGRAM)

###
```

```
 1 // This may look like C code, but it is really -*- C++ -*-
 2 // RTSS --- Railway Total System Simulator
 3 // (c) TAKAGI Ryo (at Kogakuin University)
 4 // Expandable.hh --- template classes SimpleExpandable & Expandable
 5 // -----
 6 // ChangeLog:
 7 // 2007. 11. 19
 8 //  Merged into the source of RTSS.
 9 // 2004. 6. 5
10 //  Original date on the file --- OOMTS of Birmingham University
11 // -----
12
13 // -----
14 // Remark:
15 //  This file makes it easy to expand the program at a later date.
16 // Classes, structs and typedefs:
17 //  template <PP> class SimpleExpandable
18 //  template <PP, CA> class Expandable
19 // -----
20
21 #ifndef Expandable_HH
22 #define Expandable_HH
23
24 #include <cstdlib>
25 #include <string>
26 #include <vector>
27 #include <iostream>
28
29 using std :: vector ;
30
31 /*********************************************************************
32  * class SimpleExpandable
33  *********************************************************************/
34
35 template < typename PP >
36 class SimpleExpandable
37 {
38
39 public :
40
41   // Various classes needed
42   class ObjectCreator
43     {
44     public :
45       virtual PP * create () const = 0 ;
46       void add_to_creators_list ( const char * ) ;
47     } ;
48
49   struct ObjectCreatorEntry
50   {
51     ObjectCreatorEntry
52     ( const ObjectCreator * , const char * ) ;
53     const ObjectCreator * creator ;
54     std :: string object_type ;
55   } ;
56
57   class ObjectCreatorsList
58     : public std :: vector < ObjectCreatorEntry >
59     {
60     public:
61       ObjectCreatorsList ()
62         : std :: vector < ObjectCreatorEntry > () {}
63     } ;
64
65 private :
66
67   static ObjectCreatorsList * & object_creators () ;
68
```

```
 69 public :
 70
 71   // Constructor
 72   SimpleExpandable () {} ;
 73
 74   // Add another object creator
 75   static void add_object_creator
 76     ( const ObjectCreatorEntry & ) ;
 77
 78   // Create object of type PP, return pointer
 79   static PP * create ( const char * ) ;
 80   static PP * create ( const std :: string & ) ;
 81 } ;
 82
 83
 84 /*************************************************************************
 85  * class Expandable
 86  ************************************************************************/
 87
 88 template < typename PP , typename CA >
 89 class Expandable
 90 {
 91
 92 public :
 93
 94   // Various classes needed
 95   class ObjectCreator
 96     {
 97     public :
 98       virtual PP * create () const = 0 ;
 99       virtual PP * create ( const CA & ) const = 0 ;
100       void add_to_creators_list ( const char * ) ;
101     } ;
102
103   struct ObjectCreatorEntry
104   {
105     ObjectCreatorEntry
106     ( const ObjectCreator * , const char * ) ;
107     const ObjectCreator * creator ;
108     std :: string object_type ;
109   } ;
110
111   class ObjectCreatorsList
112     : public std :: vector < ObjectCreatorEntry >
113     {
114     public :
115       ObjectCreatorsList ()
116         : std :: vector < ObjectCreatorEntry > () {}
117     } ;
118
119 private :
120
121   static ObjectCreatorsList * & object_creators () ;
122
123 public :
124
125   // Constructor
126   Expandable () {} ;
127
128   // Add another object creator
129   static void add_object_creator
130     ( const ObjectCreatorEntry & ) ;
131
132   // Create object of type PP, return pointer
133   static PP * create ( const char * ) ;
134   static PP * create ( const std :: string & ) ;
135   static PP * create ( const char * , const CA & ) ;
136   static PP * create ( const std :: string & , const CA & ) ;
```

```
137 } ;
138
139
140 /*****************************************************************
141  *  Function definitions
142  *****************************************************************/
143
144 template < typename PP >
145 typename SimpleExpandable < PP > :: ObjectCreatorsList * &
146 SimpleExpandable < PP > :: object_creators ()
147 {
148   static typename SimpleExpandable < PP > :: ObjectCreatorsList *
149     x = new typename SimpleExpandable < PP > :: ObjectCreatorsList ;
150   return x ;
151 }
152
153
154
155 template < typename PP , typename CA >
156 typename Expandable < PP , CA > :: ObjectCreatorsList * &
157 Expandable < PP , CA > :: object_creators ()
158 {
159   static typename Expandable < PP , CA > :: ObjectCreatorsList *
160     x = new typename Expandable < PP , CA > :: ObjectCreatorsList ;
161   return x ;
162 }
163
164
165
166 template < typename PP >
167 SimpleExpandable < PP > :: ObjectCreatorEntry :: ObjectCreatorEntry
168 ( const typename SimpleExpandable < PP > :: ObjectCreator * creator_in ,
169   const char * object_type_in )
170 {
171   creator = creator_in ;
172   object_type = object_type_in ;
173 }
174
175
176
177 template < typename PP , typename CA >
178 Expandable < PP , CA > :: ObjectCreatorEntry :: ObjectCreatorEntry
179 ( const typename Expandable < PP , CA > :: ObjectCreator * creator_in ,
180   const char * object_type_in )
181 {
182   creator = creator_in ;
183   object_type = object_type_in ;
184 }
185
186
187
188 template < typename PP >
189 PP *
190 SimpleExpandable < PP > :: create
191 ( const std :: string & object_type_in )
192 {
193   ObjectCreatorsList * creators
194     = SimpleExpandable < PP > :: object_creators () ;
195   for ( int i = 0 ; i < creators -> size () ; i ++ ) {
196     std :: string & s_i = creators -> at ( i ) . object_type ;
197     if ( s_i == object_type_in
198          && s_i . length () == object_type_in . length () )
199       return creators -> at ( i ) . creator -> create ();
200   }
201   // ### We should consider throwing an error.
202   std :: cerr << "Error: Creator for ¥"" << object_type_in << "¥" not found."
203               << std :: endl ;
204   exit ( 1 ) ;
```

```
205   return 0 ;
206 }
207
208
209
210 template < typename PP , typename CA >
211 PP *
212 Expandable < PP , CA > :: create
213 ( const std :: string & object_type_in ,
214   const CA & ca_in )
215 {
216   ObjectCreatorsList * creators
217     = Expandable < PP , CA > :: object_creators () ;
218   for ( typename ObjectCreatorsList :: size_type i = 0 ;
219         i < creators -> size () ; ++ i ) {
220     std :: string & s_i = creators -> at ( i ) . object_type ;
221     if ( s_i == object_type_in
222          && s_i . length () == object_type_in . length () )
223       return creators -> at ( i ) . creator -> create ( ca_in ) ;
224   }
225   // ### We should consider throwing an error.
226   std :: cerr << "Error: Creator for ¥"" << object_type_in << "¥" not found."
227               << std :: endl ;
228   exit ( 1 ) ;
229   return 0 ;
230 }
231
232
233
234 template < typename PP , typename CA >
235 PP *
236 Expandable < PP , CA > :: create
237 ( const std :: string & object_type_in )
238 {
239   ObjectCreatorsList * creators
240     = Expandable < PP , CA > :: object_creators () ;
241   for ( int i = 0 ; i < creators -> size () ; i ++ ) {
242     std :: string & s_i = creators -> at ( i ) . object_type ;
243     if ( s_i == object_type_in
244          && s_i . length () == object_type_in . length () )
245       return creators -> at ( i ) . creator -> create ();
246   }
247   // ### We should consider throwing an error.
248   std :: cerr << "Error: Creator for ¥"" << object_type_in << "¥" not found."
249               << std :: endl ;
250   exit ( 1 ) ;
251   return 0 ;
252 }
253
254
255
256 template < typename PP >
257 PP *
258 SimpleExpandable < PP > :: create
259 ( const char * object_type_in )
260 {
261   return create ( std :: string ( object_type_in ) ) ;
262 }
263
264
265
266 template < typename PP , typename CA >
267 PP *
268 Expandable < PP , CA > :: create
269 ( const char * object_type_in ,
270   const CA & ca_in )
271 {
272   return create ( std :: string ( object_type_in ) , ca_in ) ;
```

```
273 }
274
275
276
277 template < typename PP , typename CA >
278 PP *
279 Expandable < PP , CA > :: create
280 ( const char * object_type_in )
281 {
282   return create ( std :: string ( object_type_in ) ) ;
283 }
284
285
286
287 template < typename PP >
288 void
289 SimpleExpandable < PP > :: add_object_creator
290 ( const typename SimpleExpandable < PP > :: ObjectCreatorEntry &
291   object_creator_in )
292 {
293   ObjectCreatorsList * creators
294     = SimpleExpandable < PP > :: object_creators () ;
295   creators -> push_back ( object_creator_in ) ;
296 }
297
298
299
300 template < typename PP , typename CA >
301 void
302 Expandable < PP , CA > :: add_object_creator
303 ( const typename Expandable < PP , CA > :: ObjectCreatorEntry &
304   object_creator_in )
305 {
306   ObjectCreatorsList * creators
307     = Expandable < PP , CA > :: object_creators () ;
308   creators -> push_back ( object_creator_in ) ;
309 }
310
311
312
313 template < typename PP >
314 void
315 SimpleExpandable < PP > :: ObjectCreator :: add_to_creators_list
316 ( const char * object_type_in )
317 {
318   SimpleExpandable < PP > :: add_object_creator
319     ( ObjectCreatorEntry ( this , object_type_in ) ) ;
320 }
321
322
323 template < typename PP , typename CA >
324 void
325 Expandable < PP , CA > :: ObjectCreator :: add_to_creators_list
326 ( const char * object_type_in )
327 {
328   Expandable < PP , CA > :: add_object_creator
329     ( ObjectCreatorEntry ( this , object_type_in ) ) ;
330 }
331
332 #endif  // Expandable_HH
```