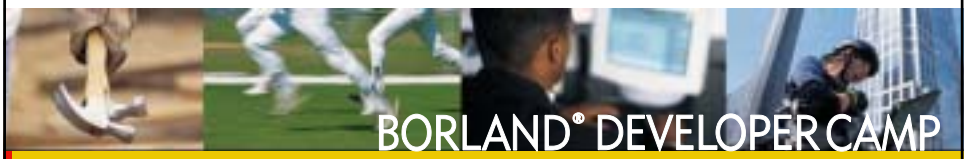


C++ TR1 / Boost C++ Library 概要

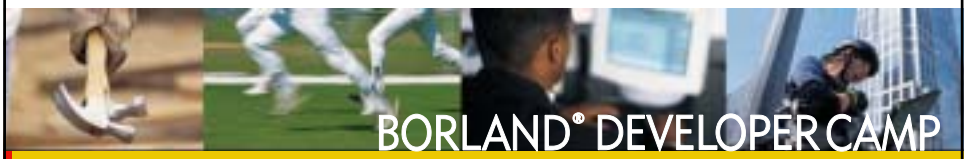
ボーランド株式会社
デベロッパーツールズ事業本部
岩本博幸

Borland

C++の歴史

- 1982 C with Class
- 1983 C++ (AT&T研究所内での最初のバージョン)
- 1985 C++ Release E (最初の外部へのリリース)
- 1985 The C++ Programming Language 出版
- 1990 Turbo C++ 誕生 (Turbo C++ for DOS 1.0)
- 1991 C++ 標準化活動の開始
- 1997 Borland C++Builder 1.0 Release
- 1998 ISO / ANSI による C++ 標準規格の承認 (ISO/IEC 14882)
- 2003 C++標準規格の改定版リリース (ISO/IEC 14882:2003 (E))
- 2003 次期C++標準規格に関する活動が開始
- 2005 TR1
- 200x? C++0x?

Borland



Boost C++ Library

Boost C++ Library (1)

- http://www.boost.org/
  - 有用で移植性があり多くの人にレビューされたライブラリを実装、公開している
  - C++の標準委員会のメンバーにより設立
    - 今も標準委員会のメンバー参加している
  - 次期C++標準化に適していることを確認する
    - 次期C++標準を担う機能のテストの場としても考えられている
      - 実際に 多くの Boost 由来のサブライブラリが次期 C++ 標準の Library technical Report に含まれている
        - 既にBoostから多くのライブラリがTR1に採用され承認済み
        - さらなるライブラリが TR2 で採用される見込み

Boost C++ Library (2)

- ライブラリがカバーする領域が広く、膨大
  - 現在 60 を越すサブライブラリの集まり
  - カバーする範囲も広い
    - 文字列処理、メモリ管理、数値計算、コンテナ、データ構造...
- 全てのライブラリの詳細は次の URL から参照できる Web ページで確認できる
  - http://www.boost.org/libs/libraries.htm

Boost C++ Libraries (3)

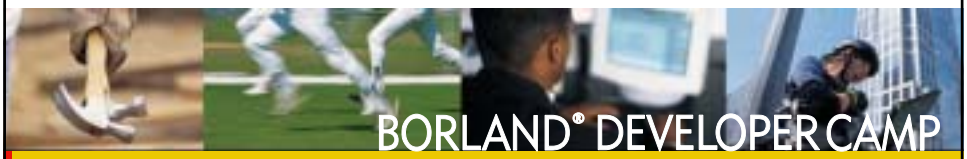
- Boost のすべての機能を全てのコンパイラで利用できる訳ではない
  - Boost は近代的な C++ の機能と標準ライブラリを有効に活用している
    - コンパイラの実装により一部の機能をサポートできない場合がある
      - コンパイラの種類 / バージョンにより変わる
      - Boost では特定のコンパイラの対応コードがマージされていることがあるので、標準の準拠度は計れないが Boost に対応するかどうかが問題である
    - Boost / コンパイラの実装による問題もある
- Boost はリリースの際、様々なプラットフォームで、退行テストが実施さ れている
  - 自分が利用しようとしているコンパイラのステータスを確認できる
    - http://boost.sourceforge.net/regression-logs/

Boost C++ Libraries (4)

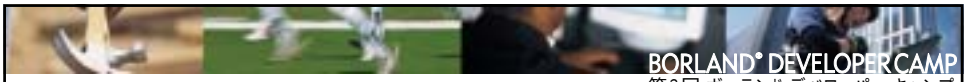
- ライセンス
  - 基本的に Boost Software License で提供される
    - http://www.boost.org/more/license_info.html
    - http://www.boost.org/LICENSE_1_0.txt

BCBBoost

- Boost が 1.33.1 でサポートする Borland のコンパイラは「Borland C++ 5.6.4 on Windows」
  - http://www.boost.org/ による
- Boost 1.33.1 を BCC32 (5.8.1/5.8.2) に対応させるアンオフィシャルパッチ
  - http://bcbboost.sourceforge.net/
    - 2006年11月現在の最新バージョンは 1_33_1-5_8_2-0.5
  - BCC32 5.8.1 を利用している開発環境
    - Borland Developer Studio 2006
  - BCC32 5.8.2 を利用している開発環境
    - Borland Developer Studio 2006 #update2
    - Turbo C++ 2006
- BCBBoost のテスト結果も見ることが出来る
  - http://bcbboost.sourceforge.net/test/1_33_1-5_8_2-0.5/cs-win32.html



C++0x / Library Technical Report（TR1）



C++0x

- 将来策定されるであろう次期標準C++の仮称
  - 200x年頃の策定を目指して活動中
- C++標準ライブラリの拡張が予定されている
  - http://www.open-std.org/jtc1/sc22/wg21/
  - 拡張の内容は Library Technical Report としてまとめられている
    - 最新バージョンのドラフト、その第一段階としての TR1
    - この後、TR2 を経て次期C++規格が策定される見通し

TR1

- C++ Library Extensions TR1
  - 次期標準C++ライブラリ拡張についてのドラフト文書
  - 次期標準C++に含まれる見込み
- Boost から強く影響を受けている
  - 多くのライブラリが Boost 由来
- ベンダーは TR1 を実装することが可能
  - 実際に 最近のGCC(libstdc++) 等では“実験的”として利用可能
  - DINKUMWARE 社も実装を提供している
  - 名前空間
    - std::tr1

TR1 で提案されたライブラリ概要 (1)

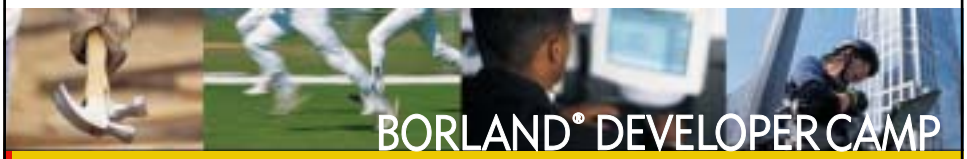
- 文字列とテキスト処理
  - 正規表現
    - regex
- コンテナ
  - array
    - 固定配列
    - std::vectorと違い大きさをコンパイル時に決定し動的メモリを利用しない
  - ハッシュベースのコンテナ
    - unordered_set / unordered_map / unordered_multiset / unordered_multimap

TR1 で提案されたライブラリ概要 (2)

- ユーティリティ
  - スマートポインタ
    - 参照先のオブジェクトを自動 delete するスマートポインタ
    - STL の auto_ptr では対応できない機能を持つスマートポインタの実装
      - 外部参照カウント方式の shared_ptr 等
  - tuple
    - 決まった個数の要素からなるコレクション
    - std::pairが2つの要素であったのに対し2つ以上のオブジェクトを扱える
- 数値計算
  - 乱数
    - 擬似乱数生成をサポートするライブラリ
    - C標準ライブラリの乱数機能より質の良い乱数生成をサポート
  - 数学関数
    - 各種数学関数を提供

TR1 で提案されたライブラリ概要 (3)

- 関数オブジェクト
  - bind
  - function
  - mem_fn
  - ref
  - reference_wrapper
- テンプレートメタプログラミング
  - traits
    - 型に関する情報をコンパイル時に知るための仕組み
- C との互換性向上
  - C99 に規定されたライブラリの機能を移植
  - ただし、 TR1 には C99の全てが含まれているわけではない



Turbo C++ / BDS 2006 から Boost を利用する

Borland Developer Studio 2006 での注意事項

- 以降、Turbo C++ 2006 でのインストール / 設定方法を解説、BDS 2006 でも基本的に同様
  - BDS2006 での注意事項
    - 最新のパッチを適用する
      - アップデート2 を適用することで、C++コンパイラが最新 (5.8.2)になる
      - BDS 2006 アップデート2
        http://support.borland.com/entry.jspa?externalID=4637&categoryID=385
      - BDS 2006 アップデート 2 適用環境と Turbo 用の Hotfix-Rollup
        http://support.borland.com/entry.jspa?externalID=5116&categoryID=385

Turbo C++ 2006 と Boost の入手

- Turbo C++ の入手
  - Explorer の場合
    - http://turboexplorer.com/jp/downloads/
- Boost の入手
  - www.boost.org のダウンロードリンクから参照できるページ
    Souceforge からダウンロードする
      - boost のセクションから
        - boost_1_33_1.exe をダウンロード
      - boost-jam のセクションから
        - boost-jam-3.1.13-1-ntx86.zip をダウンロード
        - boost-jam – Boost で利用されているビルドシステム

Boost の Turbo C++ 2006 へのインストール (1)

- Boost を展開
  - boost_1_33_1.exe を実行し、展開
    - Boost は 一部のライブラリ (regex, thread, signals, test…etc) 等を利用しない場合、ヘッダを #include することで利用可能となる
      - BDSの環境設定で include パスを指定する
    - ビルドが必要なライブラリを利用したい場合
      - bjam を利用してライブラリをビルドする
        - boost.Jam – Boost で提供されるビルドユーティリティ

Boost の Turbo C++ 2006 へのインストール (2)

- Boost のビルド
  - bjam を展開
  - BcbBoost を展開
    - Boost を展開したディレクトリへディレクトリマッピングを合わせた形でコピー
    - Boost は Boost.Jam では用意されていない BCC5.8.1 / 5.8.2 用の Jam ファイルを提供しており、これにより bjam から簡単にビルドできるようになる
      - Boost が提供している toolsetname-tools.jam
        - borland-5_5_1-tools.jam / borland-5_6_4-tools.jam
      - BcbBoost で追加される toolsetname-tools.jam
        - borland-5_8_1-tools.jam / borland-5_8_2-tools.jam

BORLAND® DEVELOPER CAMP
第3回 ボーランド デベロッパー キャンプ

Boost の Turbo C++ 2006 へのインストール (2)

- コマンドプロンプトを立ち上げ、Boost を展開したディレクトリに移動し bjam を用いてビルドを始める
  - 例

```
bjam -sTOOLS=borland-5_8_2 install
```

  - この場合、c:¥boost に出力される

```
bjam -sTOOLS=borland-5_8_2 "prefix=C:¥Program Files¥Borland¥BDS¥4.0" install
```

  - この場合、prefix が設定されているので$(BDS)¥include と $(BDS)¥lib に出力される

ここでは Boost.Python には言及しません。
これを利用する場合、他の関連ドキュメントを参考にしてください。

※ビルドには非常に時間がかかりますので、注意してください。

Boost の Turbo C++ 2006 へのインストール (3)

- 統合開発環境への環境変数の設定
  - プロジェクトのデフォルトオプションで設定するか、新たなビルド設定を作成する
  - プロジェクトのデフォルトオプションを利用する場合
    - プロジェクトを全て閉じた状態でメニューの「プロジェクト|デフォルトオプション|C++Builder」で「プロジェクトオプション・ダイアログ」を開く
      - 「C++コンパイラ (BCC32) |パスと定義」の「インクルードファイルの検索パス」に次のディレクトリへのパスを追加
        - [Boostが展開されたディレクトリ]¥include¥boost-1_33_1
      - 「リンカ (ilink32) |パスと定義」の「ライブラリパス」に次のディレクトリへのパスを追加します。
        - [Boostが展開されたディレクトリ]¥lib

Boost の Turbo C++ 2006 へのインストール (4)

- 新たなビルド設定を作成する場合
  - プロジェクトオプション・ダイアログを開き、「ビルド設定」の項の「設定」ボタンを押す
  - 「ビルド設定・ダイアログ」で「新規作成」ボタンを押し新しいビルド設定を作成する。
  - プロジェクトオプション・ダイアログを開き、「ビルド設定」の項で、新しいビルド設定が選べるようになるので、そこにコンパイラとリンカのパスの定義を設定する
    - リンカのパスの定義に、BDS / Turbo の ライブラリへのパスの設定を忘れないようにする
      - 例:$(BDS)¥lib¥debug
  - ビルド設定の切り替えは、「プロジェクト|ビルド設定」から行なえます
- テスト
  - 簡単なコンソールアプリケーションを作成し、テストする

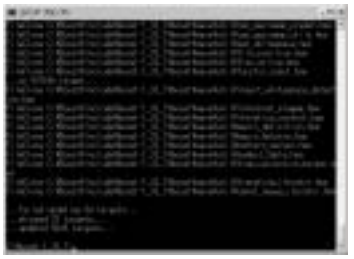
Boost の Turbo C++ 2006 へのインストール例 (1)

- Turbo C++ のインストール
- boost_1_33_1.exe を「C:¥」に展開
- boost-jam-3.1.13-1-ntx86.zip 内のファイルを「c:¥ boost_1_33_1」に展開
- BcbBoost を展開し、[bcbboost¥boost]の中身 (3つのディレクトリ) を「c:¥ boost_1_33_1」へ上書きコピーする
- コマンドプロンプトを起動し、「c:¥ boost_1_33_1」へ移動する

```
CD c:¥boost_1_33_1
```

- 次のコマンドを入力する

```
bjam –sTOOLS=borland-5_8_2 install
```



Boost の Turbo C++ 2006 へのインストール例 (2)

- ビルド終了後、「C:¥Boost」が作成されているか確認する
- Turbo C++2006 を起動する
  - プロジェクトが開いている場合は、「ファイル|全てを閉じる」でプロジェクトを全て閉じておく
- 環境を設定する
  - メニューの「プロジェクト|デフォルトオプション|C++Builder」から
    - 「C++コンパイラ (BCC32) |パスと定義」の「インクルードファイルの検索パス」に次のディレクトリへのパスを追加
      - c:¥Boost¥include¥boost-1_33_1
    - 「リンカ (ilink32) |パスと定義」の「ライブラリパス」に次のディレクトリへのパスを追加します。
      - c:¥Boost¥lib

Boost の Turbo C++ 2006 へのインストール例 (3)

- 簡単なテストアプリケーションを作成してみる
  - メニューの「ファイル|新規作成|その他」から、コンソールアプリケーションを選択

  - 「新規コンソールアプリケーション」ダイアログで「VCLを使う」と「マルチスレッド」のチェックを外すよう設定し、「OK」ボタンを押す

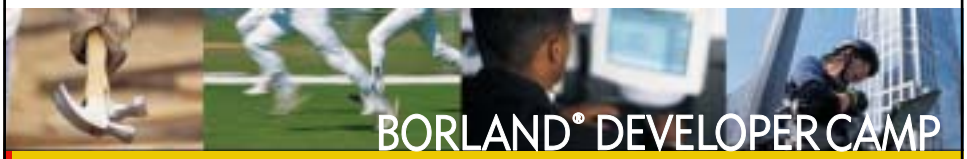
Boost の Turbo C++ 2006 へのインストール例 (3)

- boost を利用したコードを記述する
  - ここでは、boost::format を利用した簡単なアプリケーションで boost が利用できるようになっているか、確認してみる

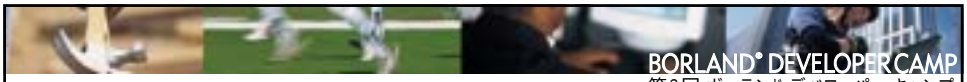
```
#include<iostream>
#include<boost/format.hpp>

int main(int argc, char* argv[]){
  int x = 1024;
  std::cout << "1024 を 16 進数で出力" << std::endl;
  std::cout << boost::format("%1$#x") % x << std::endl;
  return 0;
}
```

※bcc32 で利用できるライブラリであっても、プリコンパイルヘッダを利用している場合に不具合を引き起こす事がありますので、boost を利用する場合、その設定をプロジェクトオプションで切るか、「#pragma hdrstop」の後に boost ライブラリのヘッダを include する事を推奨します。



Boost を使ってみる



regex

- 正規表現ライブラリ
  - std::string::find などではできない強力な文字列の検索、置き換え等が可能となる
    - regex_match
      - 正規表現を利用した文字列のマッチを確認
    - regex_search
      - 正規表現を利用した文字列の検索
    - regex_replace
      - 正規表現を利用した文字列の置き換え

regex_match

```
#include <vcl.h>
#include <boost/regex.hpp>
…中略
bool entry_check(const std::string& s){
  //UIDで始まる数字七桁の「ユーザーID」の入力チェック
  boost::regex r("UID[¥¥d]{7}");
  return boost::regex_match(s, r);
}
void __fastcall TForm1::Button1Click(TObject
*Sender){
  if (entry_check(Edit1->Text.c_str())){
            ShowMessage("正しい入力形式です。");
  }else{
            ShowMessage("入力形式が違います。");
  };
}
```

- 正規表現での入力チェック
  - フォーム上のEditコンポーネントの入力をチェックしている
  - UIDで始まる数字七桁からなる「ユーザーID」の入力にフォーマットチェックを行なうコードを実装
  - UID[¥d]{7}

regex を利用する際の問題

- boost::regex はビルドされたライブラリモジュールを必要とする
  - 現時点でライブラリが要求する lib のファイル名と実際のファイル名がずれているため、リンカエラーとなる
  - 解決策
    - コンパイル時の条件定義に明示的に BOOST_REGEX_NO_LIB を指定し、手動で libboost_regex-bcb58-mt-d-1_33_1.lib を追加する方法が考えられる。
  - 手法
    - プロジェクトのオプションで、[条件定義]に BOOST_REGEX_NO_LIB を追加する
    - C:¥Boost¥lib¥libboost_regex-bcb58-mt-d-1_33_1.lib をプロジェクトに追加

tuple

- 決まった個数の要素からなるコレクション
  - std::pairが2つの要素であったのに対し2つ以上のオブジェクトを扱える
  - 2個以上の値を返す関数を定義する場合などに有効に利用できる

tuple

```
#include <vcl.h>
#include <iostream>
#include <boost/tuple/tuple.hpp>
#include <boost/tuple/tuple_io.hpp>
…中略
boost::tuple<int, int, int, int> FormStatus(){
  return boost::make_tuple(Form2->Top, Form2->Left,
                           Form2->Width,  Form2->Height);
}
void __fastcall TForm2::Button1Click(TObject *Sender){
  boost::tuple<int, int, int, int> t = FormStatus();
  Memo1->Lines->Clear();
  Memo1->Lines->Add(IntToStr(t.get<0>()));
  Memo1->Lines->Add(IntToStr(t.get<1>()));
  Memo1->Lines->Add(IntToStr(t.get<2>()));
  Memo1->Lines->Add(IntToStr(t.get<3>()));
}
```

- 4つの int を扱う tuple でフォームの位置、大きさ情報を扱う関数を作成
  - 出力は、フォーム上のTMemo へ出力している
- tuple からの値の取り出し
  - tuple.get<n>();

array / random

- array
  - 容量が固定された静的な配列
  - その大きさをコンパイル時に決定し動的メモリを利用しない
    - std::vectorと違う部分
- random
  - 擬似乱数生成をサポートするライブラリ
  - C標準ライブラリの rand, srand より質の良い乱数生成をサポート
    - 乱数ルーチン、シードをグローバルに持たない
    - 乱数生成エンジンと分布のタイプを分け、組み合わせにより柔軟な擬似乱数生成に対応
      - メルセンヌ・ツイスタ法 / 線形合同法
      - ベルヌーイ分布 / 正規分布

BORLAND® DEVELOPER CAMP
第3回 ボーランド デベロッパー キャンプ

array / random

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  boost::mt19937       gen; //メルセンヌ・ツイスタ法の乱数生成機
  boost::uniform_smallint<> dst(1, 6); //一様整数分布 (範囲0～6)
  //現在時刻でseedを設定
  gen.seed(static_cast<unsigned int>(time(0)));
  boost::variate_generator<boost::mt19937&, boost::uniform_smallint<> > dice(gen, dst);

  //配列を作成し要素を設定
  boost::array<int, 10> myArray;
  for(int i=0; i < myArray.size(); ++i ){myArray[i] = dice();}

  //ためしにメモへ出力
  for(int i=0; i < myArray.size(); ++i ){Memo1->Lines->Add(IntToStr(myArray[i]));}
}
```

スマートポインタ

- shared_ptr
  - 参照カウント方式スマートポインタ
  - スマートポインタ側で、参照カウントを実装している
- weak_ptr
  - shared_ptr と同じオブジェクトを指すポインタとなれるが参照カウントは増やさない
  - shared_ptr の循環参照を防ぐ等を実現するスマートポインタ
- intrusive_ptr
  - 参照カウント方式スマートポインタ
  - shared_ptr と違い、ポインタが指すオブジェクトに参照数を数える実装が必要
    - 独自の参照カウント実装を持つライブラリ等を利用する場合等に利用する
- 他
  - **scoped_ptr**
  - **scoped_array**
  - **shared_array**

shared_ptr

```
boost::array<std::string*, 10> ar1;
boost::array<boost::shared_ptr<std::string>, 10> ar2;
void __fastcall TForm1::Button1Click(TObject *Sender){
  for(int i=0; i < 10; ++i ){
      //適切な解放処理を行なわないと文字列オブジェクトのメモリーリークが発生する
      ar1[i] = new std::string("abc");
     //文字列オブジェクトは、このまま参照元が無くなれば delete される
      ar2[i] = boost::shared_ptr<std::string>(new std::string("abc"));}
}
```

- 必要のなくなったオブジェクトを確実に解放できる方法の一つ
  - boost::shared_ptr は 参照カウンタ方式のスマートポインタ
  - std::auto_ptr との違い
    - ポインタを共有でき、全ての参照元が無くなると、delete される
    - また、STLのコンテナの要素にもなることが可能

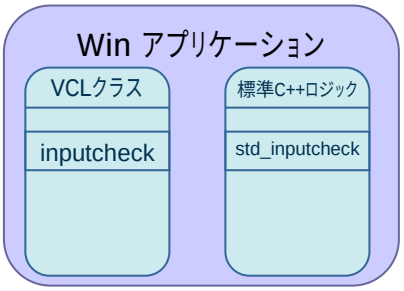
標準C++と特定のフレームワークに依存するコードの相性

- 特定のフレームワークを利用する部分と標準C++のみで記述されるライブラリを分けることで C++ ソースの再利用性 / 移植性を高める
- C++Builder が採用している VCL は C++ 標準ライブラリとの共存を考えた場合、その住み分けをはっきりさせる必要がある
  - VCL は Delphi で記述されており、Delphiと互換性を維持できるように設計されている例：
    - VCL が採用している文字列は AnsiString クラス
      - Sysmac.h で「typedef AnsiString String;」されているだけ
      - 標準 C++ ライブラリの文字列は std::string
      - 互換性がない
        - c_str メンバ関数を利用して char* でやり取りを行なう

```
{
  AnsiString AnsiStr = "ABCDEFGHIJKLMN";
  std::string str = AnsiStr.c_str();
}
```

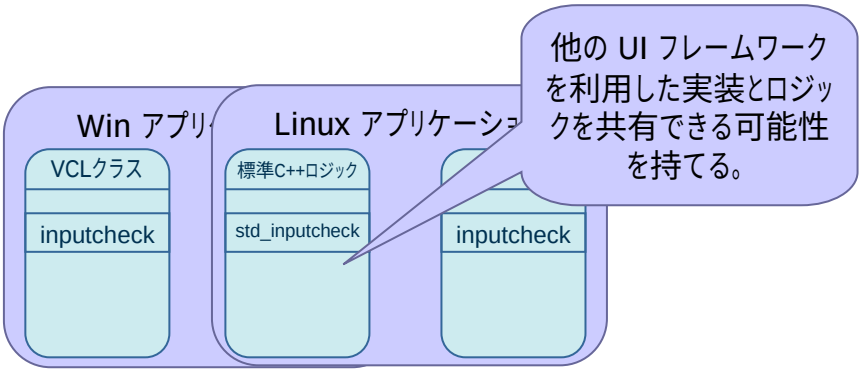

標準C++で記述されたコードの分離

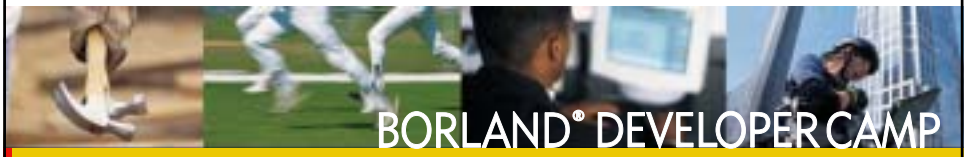
- VCLを利用して記述する部分とC++標準で記述するものを分ける
  - 特定のフレームワークを利用する部分と標準C++のみで記述されるライブラリを分けることで C++ ソースの再利用性 / 移植性を高める



C++標準で記述されたものを分ける

- VCLを利用して記述する部分とC++標準で記述するものを分ける
  - 特定のフレームワークを利用する部分と標準C++のみで記述されるライブラリを分けることで C++ ソースの再利用性 / 移植性を高める





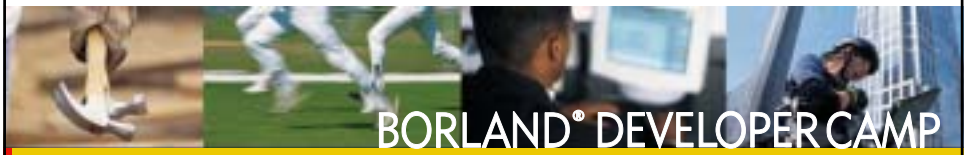
Appendix: Boost Software License

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



Thank you