

BCC32C コンパイラを 使ってみよう

2017 年 12 月

エンバカデロ・テクノロジーズ

目次

BCC32C コンパイラを使う	1
BCC32C コンパイラのインストール.....	1
環境変数 C の設定	2
BCC32C の動作チェック	3
はじめての C++アプリケーションを コンパイルする	4
Hello World アプリケーション	4
その他のツール.....	6
C++Builder でもっと効率的に プログラミングする.....	7
C++Builder とは	7
C++Builder で Hello World アプリケーション作成を体験する.....	9
コンソールアプリケーションの新規作成	9
プロジェクトマネージャ	11
エディタのコーディング支援機能.....	12
コンパイルエラーを調べる.....	14
デバッガを使う	15

BCC32C コンパイラを使う

BCC32C コンパイラは、Windows 32-bit 向けの Clang ベースのコンパイラです。エンバカデロのビジュアル C++開発環境 C++Builder に搭載されており、BCC64 コンパイラとともに、C++11 準拠のコードを記述できます（互換性のために従来の Classicbcc32 コンパイラも C++Builder には搭載されています）。

この新しい BCC32C コンパイラは、エンバカデロの Web サイトから無料ダウンロードでき、コマンドライン コンパイラとして、（Windows のコマンドプロンプトから）C++11 準拠の C++コードをコンパイルできます。BCC32C コンパイラには、Dinkumware STL と構築のために必要なヘッダーインポートライブラリも含まれます。

BCC32C コンパイラのインストール

エンバカデロの Web サイトから BCC32C コンパイラをダウンロードします。



図 1 BCC32C コンパイラのダウンロード

ファイルを解凍すると、bin, lib, include フォルダと「Embarcadero Freeware Software License Agreement.txt」「Installing and Using the Embarcadero C++ 10.1 Berlin Command-line Compiler.txt」が展開されます。

これらのファイルを、適当なフォルダにコピーします。今回は、「C:\data\BCC101」下にコピーしました。

環境変数の設定

BCC32C コンパイラに含まれるツールを使用するには、これらが保管されている bin フォルダに PATH を通しておく必要があります。次のように PATH 設定に bin フォルダを追加します（この場合は、C:\data\BCC101\bin）。

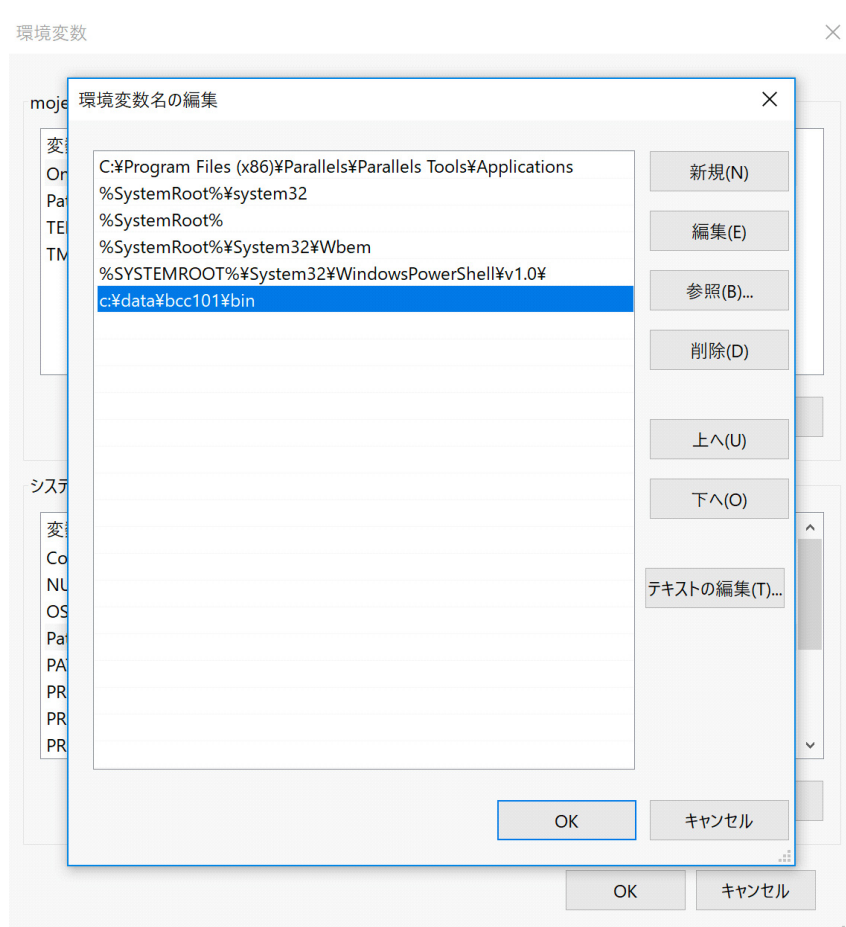


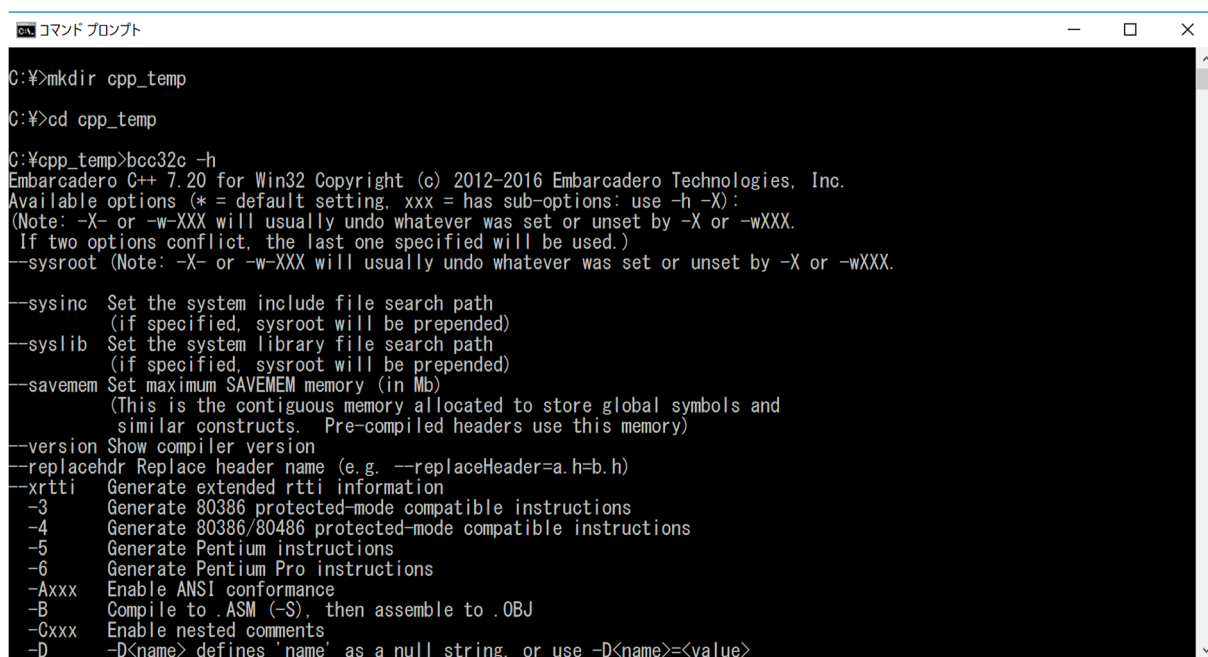
図2 環境変数の設定

BCC32C の動作チェック

コンパイラ ファイル名は `bcc32c.exe` ですので、それぞれコマンドラインで実行できるかチェックします。チェックのためのフォルダは「`c:\cpp_temp`」とします。Windows メニューで、コマンド プロンプトを起動したら、以下のように入力します。

```
mkdir cpp_temp
cd cpp_temp
bcc32c -h
```

`bcc32c` の引数に指定する `-h` は、コンパイラのヘルプ情報を表示するためのスイッチです。次のように表示されれば、インストールは完了です。



```
コマンド プロンプト
C:\>mkdir cpp_temp
C:\>cd cpp_temp
C:\cpp_temp>bcc32c -h
Embarcadero C++ 7.20 for Win32 Copyright (c) 2012-2016 Embarcadero Technologies, Inc.
Available options (* = default setting, xxx = has sub-options: use -h -X):
(Note: -X- or -w-XXX will usually undo whatever was set or unset by -X or -wXXX.
If two options conflict, the last one specified will be used.)
--sysroot (Note: -X- or -w-XXX will usually undo whatever was set or unset by -X or -wXXX.

--sysinc Set the system include file search path
         (if specified, sysroot will be prepended)
--syslib Set the system library file search path
         (if specified, sysroot will be prepended)
--savemem Set maximum SAVEMEM memory (in Mb)
         (This is the contiguous memory allocated to store global symbols and
         similar constructs. Pre-compiled headers use this memory)
--version Show compiler version
--replacehdr Replace header name (e.g. --replaceHeader=a.h=b.h)
--xrtti Generate extended rtti information
-3 Generate 80386 protected-mode compatible instructions
-4 Generate 80386/80486 protected-mode compatible instructions
-5 Generate Pentium instructions
-6 Generate Pentium Pro instructions
-Axxx Enable ANSI conformance
-B Compile to .ASM (-S), then assemble to .OBJ
-Cxxx Enable nested comments
-D -D<name> defines 'name' as a null string, or use -D<name>=<value>
```

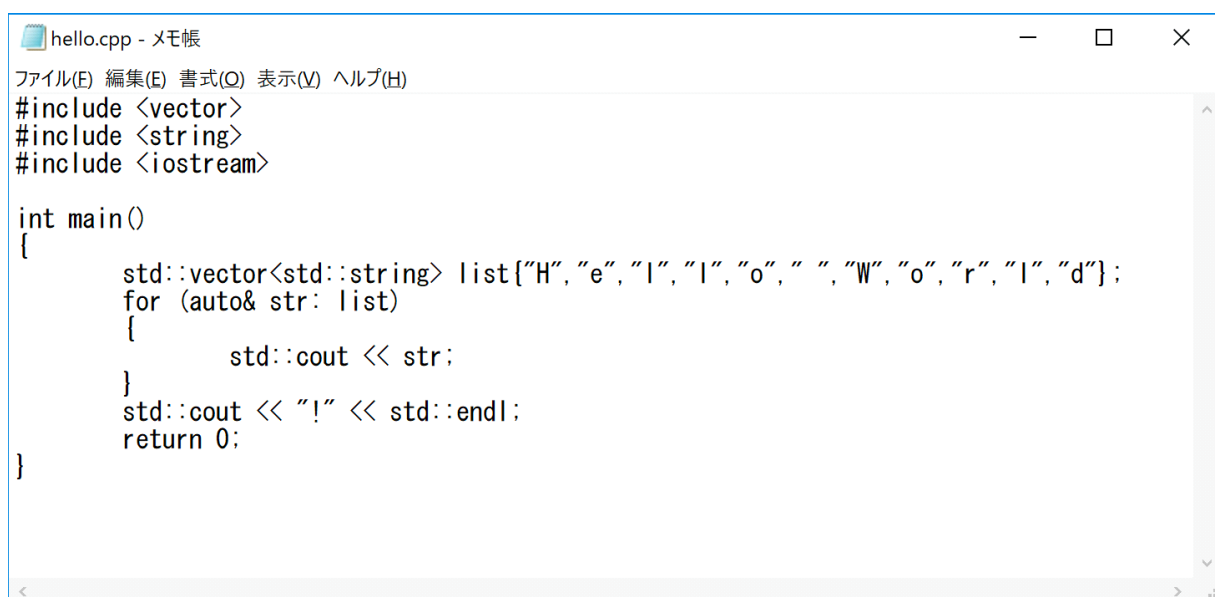
図 3 bcc32c の実行

はじめての C++ アプリケーションを コンパイルする

それでは、簡単な C++ プログラムを作成して、BCC32C コンパイラの使い方を見ていきましょう。

Hello World アプリケーション

ここで作成するのは、プログラミング入門の定番「Hello World」です。メモ帳などのエディタを使って、Hello World を表示するプログラムを記述します。

A screenshot of a Notepad window titled "hello.cpp - メモ帳". The window contains the following C++ code:

```
ファイル(E) 編集(E) 書式(O) 表示(V) ヘルプ(H)
#include <vector>
#include <string>
#include <iostream>

int main()
{
    std::vector<std::string> list{"H","e","l","l","o"," ","W","o","r","l","d"};
    for (auto& str: list)
    {
        std::cout << str;
    }
    std::cout << "!" << std::endl;
    return 0;
}
```

図 4 メモ帳でコードを記述

作成したコードは、「hello.cpp」という名称で、先ほど作成した「cpp_temp」フォルダ内に保存します。全体のコードは次のとおりです。

```
#include <vector>
#include <string>
#include <iostream>

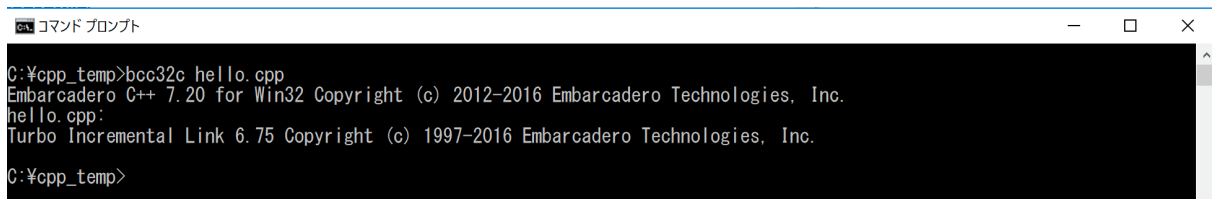
int main()
```

```
{  
    std::vector<std::string> list{"H","e","l","l","o"," ","W","o","r","l","d"};  
    for (auto& str: list)  
    {  
        std::cout << str;  
    }  
    std::cout << "!" << std::endl;  
    return 0;  
}
```

コマンドラインで次のように、bcc32c を実行します。

```
bcc32c hello.cpp
```

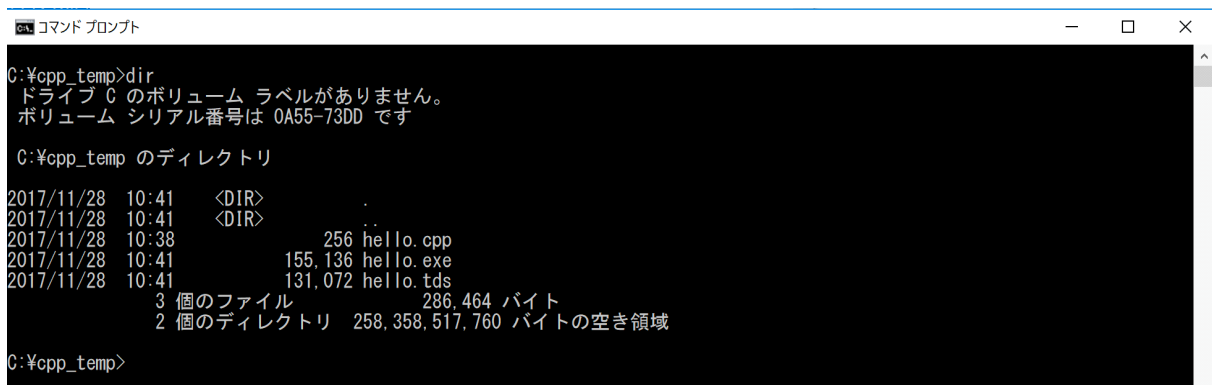
プログラムに記述ミスがなければ、以下のように、コンパイル、リンクが実行されます。もし、何らかの記述ミスがあると、エラーメッセージが表示されます。



```
コマンド プロンプト  
C:\>cd C:\cpp_temp  
C:\cpp_temp>bcc32c hello.cpp  
Embarcadero C++ 7.20 for Win32 Copyright (c) 2012-2016 Embarcadero Technologies, Inc.  
hello.cpp:  
Turbo Incremental Link 6.75 Copyright (c) 1997-2016 Embarcadero Technologies, Inc.  
C:\cpp_temp>
```

図 5 bcc32c の実行

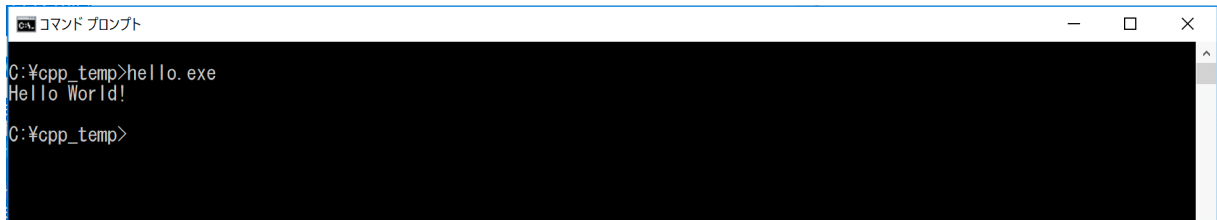
フォルダ内を確認してみると、作成した hello.cpp のほかに 2 つのファイルが作成されていることが分かります。



```
コマンド プロンプト  
C:\>cd C:\cpp_temp  
C:\cpp_temp>dir  
ドライブ C のボリューム ラベルがありません。  
ボリューム シリアル番号は 0A55-73DD です  
  
C:\cpp_temp のディレクトリ  
  
2017/11/28 10:41 <DIR> .  
2017/11/28 10:41 <DIR> ..  
2017/11/28 10:38          256 hello.cpp  
2017/11/28 10:41        155,136 hello.exe  
2017/11/28 10:41        131,072 hello.tds  
3 個のファイル          286,464 バイト  
2 個のディレクトリ    258,358,517,760 バイトの空き領域  
C:\cpp_temp>
```

図 6 cpp_temp フォルダの内容

ここに作成されている hello.exe が実行可能なプログラムです。hello.exe を実行すると、次のように、「Hello World!」と表示されます。



```
コマンドプロンプト
C:\%cpp_temp>hello.exe
Hello World!
C:\%cpp_temp>
```

図7 hello.exe の実行

その他のツール

BCC32C コンパイラには、このほかにも以下のようなツールが含まれています。

名称	機能
bcc32c	Clang ベース 32bit コンパイラ
cpp32c	プリプロセッサコンパイルされる前の処理を行います
grep	ドキュメント文字列検索ツール
ilink32	32bit リンカ
implib	DLL またはモジュール定義ファイルのどちらかまたは両方を、入力として受け取り、インポート ライブラリ (.lib) を出力として生成します。
make	プロジェクトのコンパイルとリンクのサイクルの管理を支援するコマンドライン ユーティリティです
tdump	ファイルを構造的に分解し、ファイルの拡張子を元に出力表示形式を判断
tlib	.OBJ (オブジェクト モジュール) ファイルのライブラリを管理するユーティリティ
touch	指定されたファイルのタイムスタンプをシステムの現在日時または指定日時に変更します

C++Builder でもっと効率的に プログラミングする

BCC32C は、コマンドラインツールなので、外部のエディタと組み合わせてプログラミングを行う必要があります。コンパイル時のエラートレースやバグの発見など、コードを精査したい場合には、便利とはいえません。

統合開発環境は、このようなプログラミング時の利便性を高めるために、エディタとコンパイラとデバッガをひとつの環境にまとめて連携するようにしたものです。ここでは、BCC32C を搭載した統合開発環境である C++Builder について説明しましょう。

C++Builder とは

C++Builder は、エンバカデロが提供する C++ 開発環境で、BCC32C をはじめとする複数のコンパイラが搭載されています。C++Builder の統合開発環境では、C++アプリケーションのコーディング、ビルド、デバッグが可能で、C++コードを効率的に作成することができます。

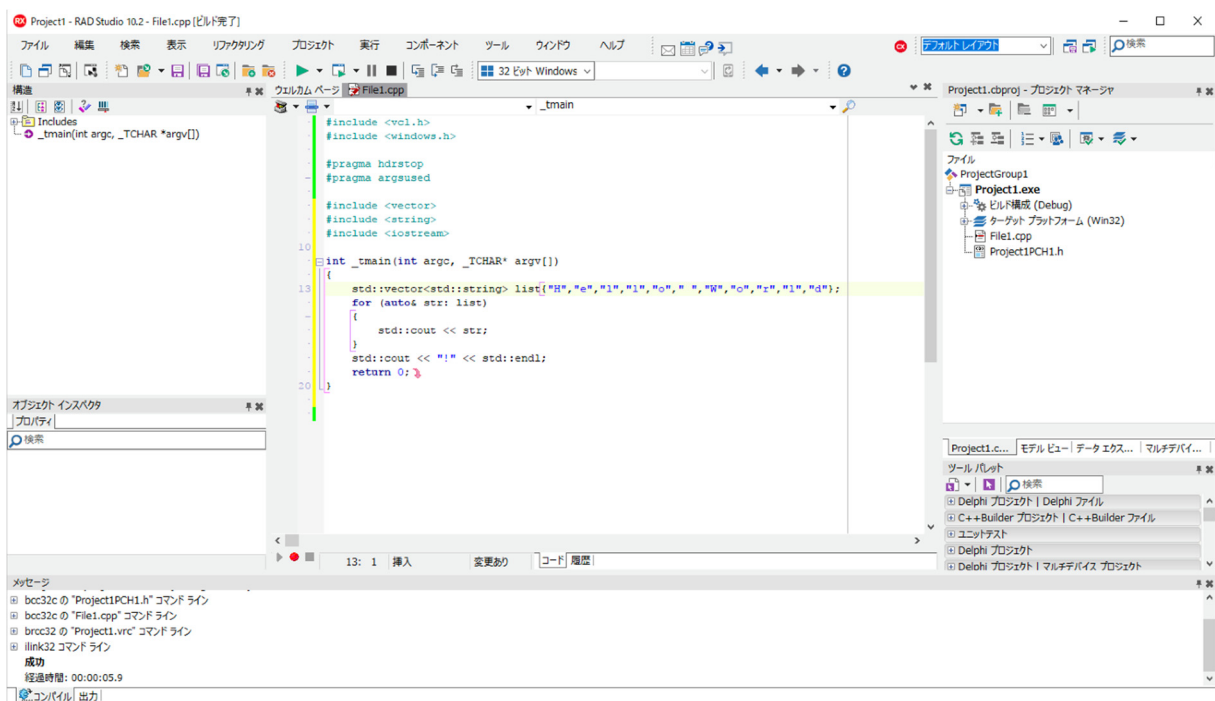


図 8 C++Builder の統合開発環境

さらに、Delphi や Visual Basic と同じようにドラッグ&ドロップのビジュアル操作でユーザーインターフェイスやデータアクセス機能を作成できるコンポーネント指向の開発をサポートしています。そのため、他の C++開発環境と比較して、ユーザーインターフェイスの作成にかかる工数を劇的に削減できます。

加えて、C++Builder では、Windows だけでなく、macOS、iOS、Android 向けのアプリケーションも構築できます。ひとつの C++コードから、デスクトップ、スマートフォン、タブレット、さらにはウェアラブル端末までのネイティブアプリケーションを作成できるのは大変魅力的です。

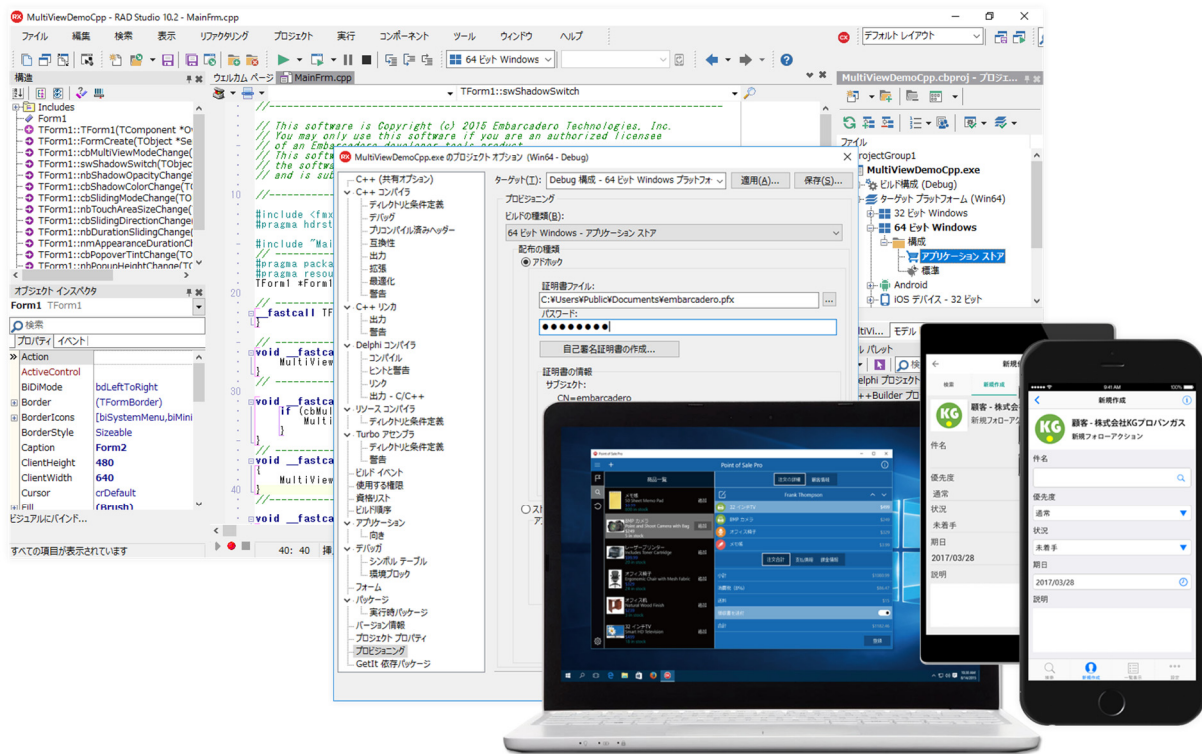


図9 C++Builder のマルチデバイス開発

C++Builder には、個人やスタートアップ企業が利用できる無料の C++Builder Starter、業務アプリケーション開発に利用できる C++Builder Professional や C++Builder Enterprise といった製品ラインナップが用意されています。

C++Builder で Hello World アプリケーション作成を体験する

C++Builder が提供する基本的なプログラム作成支援機能を理解するため、先ほどコマンドライン向けに作成した Hello World アプリケーションを、再度 C++Builder 上で作成してみましょう。ここでは、C++Builder が得意とするビジュアル開発の要素は使いませんが、エディタ機能、プロジェクト管理機能、デバッグ機能など、すべてのアプリケーション構築に関連する主要機能を理解するにはよい例です。

コンソールアプリケーションの新規作成

C++Builder を起動したら、メインメニューで [新規作成 | その他] を選択し、表示された「新規作成」ダイアログで、「C++Builder プロジェクト」内の「コンソール アプリケーション」を選択します。

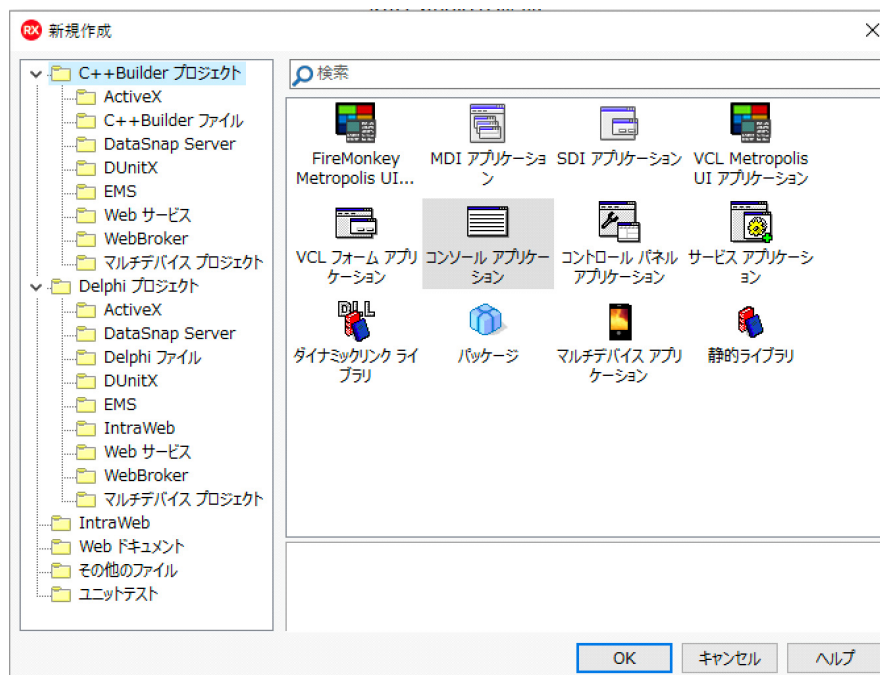


図 10 新規作成

次のように、「新規コンソール アプリケーション」ウィザードが表示されるので、

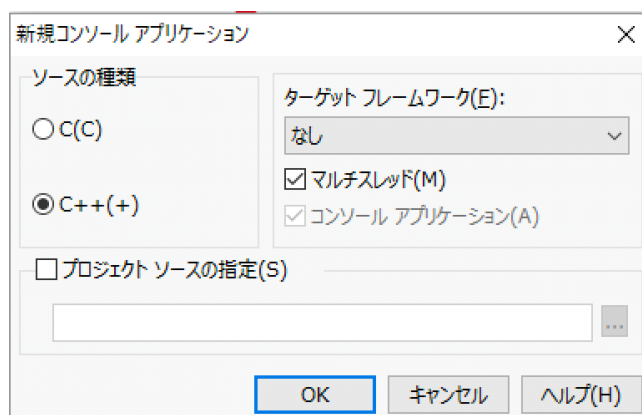


図 11 「新規コンソール アプリケーション」ウィザード

「ターゲット フレームワーク」に、「なし」もしくは「ビジュアル コンポーネント ライブラリ」を指定して、[OK] ボタンをクリックします。

すると、次のように C++プロジェクトが作成されます。

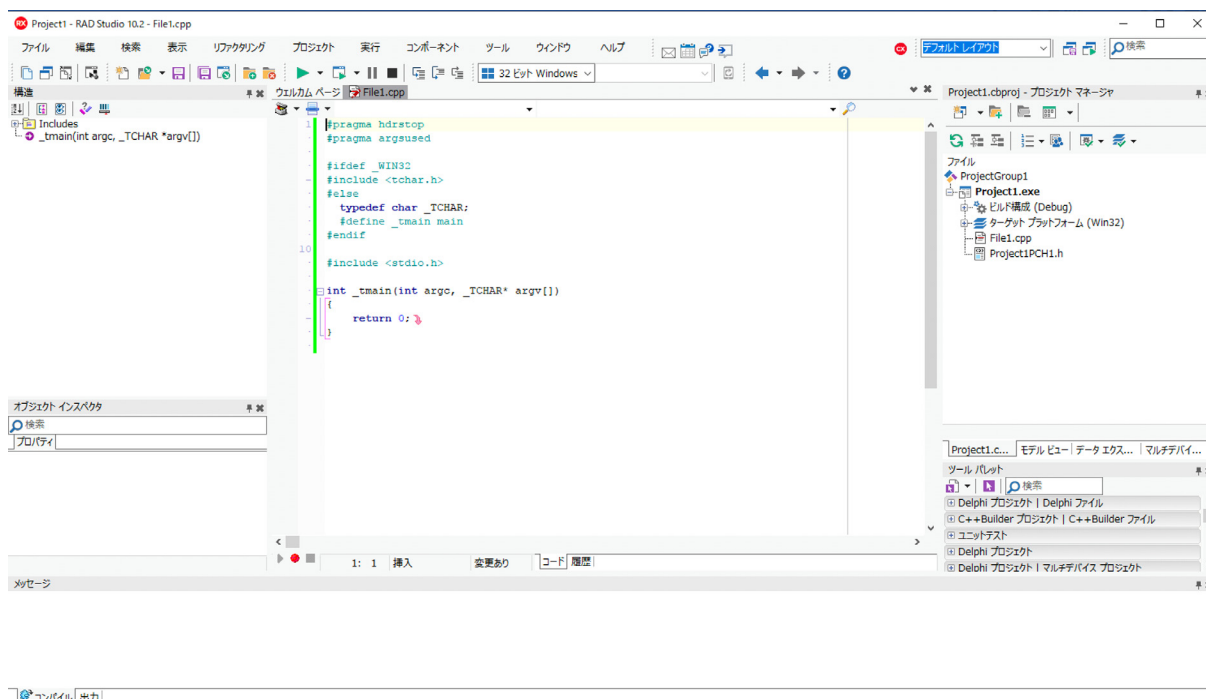


図 12 新規 C++プロジェクト

プロジェクトマネージャ

画面右側にあるプロジェクトマネージャには、作成した C++プロジェクト（デフォルトで「Project1」のように名前が付けられています）を構成する要素が表示されています。

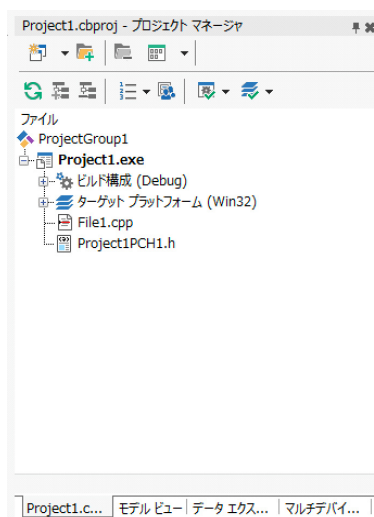


図 13 プロジェクトマネージャ

プロジェクトマネージャには、アプリケーションを作成するために必要となるファイルを追加、管理できるほか、ビルド構成やターゲットプラットフォームを設定できます。

この例では、「Win32」がターゲットプラットフォームで、「Debug」ビルド（デバッガによるトレースができるようにデバッグ情報が含まれるビルド）が行われます。C++Builder では、ターゲットを「Win64」や「macOS」などに変更することも可能です（エディションによって異なります）。

C++Builder のプロジェクトには、以下のようなファイルが含まれます。今回作成した、コンソールアプリケーションでは、以下のようなファイルによって構成されています。

ファイル名	詳細
[プロジェクトファイル名].cbproj	最終アウトプットされる EXE ファイル名
[C++ソースファイル名].cpp	main 関数が入る C++ソースファイル
[プリコンパイルヘッダ]PCH1.h	プリコンパイルヘッダファイル

プロジェクト名は、作成される実行ファイル（EXE ファイル）の名称となります。

エディタのコーディング支援機能

先ほどメモ帳に記述したのと同じコードを入力してみましょう。C++Builder のコードエディタには、構文強調表示機能が搭載されており、コードを構成する予約語や文字列リテラルなどを、視認しやすくしてくれます。

また、複数取り消し機能、コード補完機能、言語要素の状況依存型ヘルプなども備えており、効率的なコーディング作業が可能です。以下は、コード補完機能を使用しているところです。コード補完機能は、現在のカーソル位置で使用できるシンボルのドロップダウンリストを表示します。

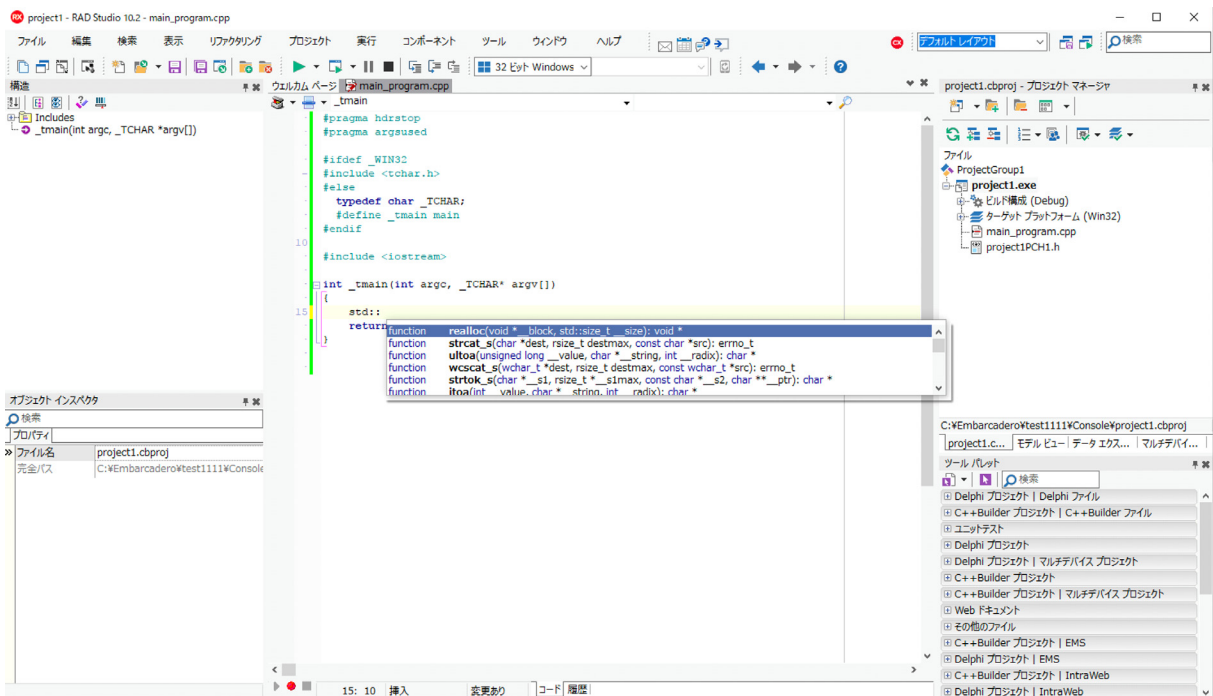
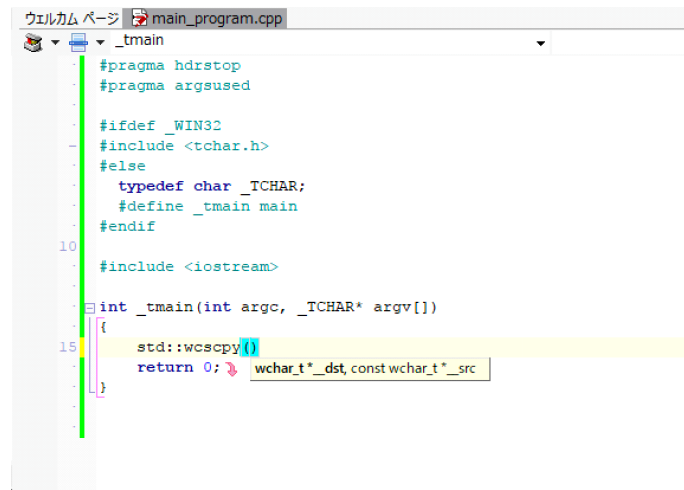


図 14 コード補完機能

特定の言語のコード補完を呼び出すには、コードエディタを使用している状態で、Ctrl+Space を押します。

コーディング中にメソッドを記述していくと、呼び出すメソッドの引数の名前と型をヒントとして表示します。



```
main_program.cpp
_tmain
#pragma hdrstop
#pragma argsused
#ifdef _WIN32
#include <tchar.h>
#else
typedef char _TCHAR;
#define _tmain main
#endif
10 #include <iostream>
int _tmain(int argc, _TCHAR* argv[])
{
15     std::wscpy();
    return 0;
}
```

The screenshot shows a code editor window titled 'main_program.cpp' with a file explorer on the left showing a folder named '_tmain'. The code contains preprocessor directives, includes, and a function definition for `_tmain`. A yellow highlight is under the `std::wscpy()` call, and a tooltip is displayed below it showing the function signature: `wchar_t* _dst, const wchar_t* _src`.

図 15 引数のヒント表示

構造強調表示では、コードブロックをグループ化し、該当するキーワードを識別します。次のようにネストしたブロックでは、交互に変わる 4 色を使用して、特定のブロックの深さを区別します。



```
main_program.cpp
_tmain
#pragma hdrstop
#pragma argsused
#ifdef _WIN32
#include <tchar.h>
#else
typedef char _TCHAR;
#define _tmain main
#endif
10 #include <iostream>
#include <vector>
int _tmain(int argc, _TCHAR* argv[])
{
20     std::vector<int> vil{0,1,2,3};
    if (vil[0] == vil[1])
    {
        if (vil[2] == vil[3])
        {
            while (1)
            {
                for (int i = 0; i < 10; i++)
                {
                }
            }
        }
    }
30     return 0;
}
```

The screenshot shows the same code editor window as in Figure 15, but with structural highlighting. The code is color-coded to show nesting: the outermost block is green, the `if (vil[0] == vil[1])` block is blue, the `if (vil[2] == vil[3])` block is purple, the `while (1)` block is red, and the `for` loop is yellow. The `return 0;` line is highlighted in yellow.

図 16 構文強調表示

コンパイルエラーを調べる

コードを入力したらプログラムをビルドしてみましょう。C++Builder では、[実行] メニューを選択するだけで、必要なファイルをコンパイルして、アプリケーションを作成、実行します。

ビルドを実行すると、次のようにコンパイラメッセージが表示されます。

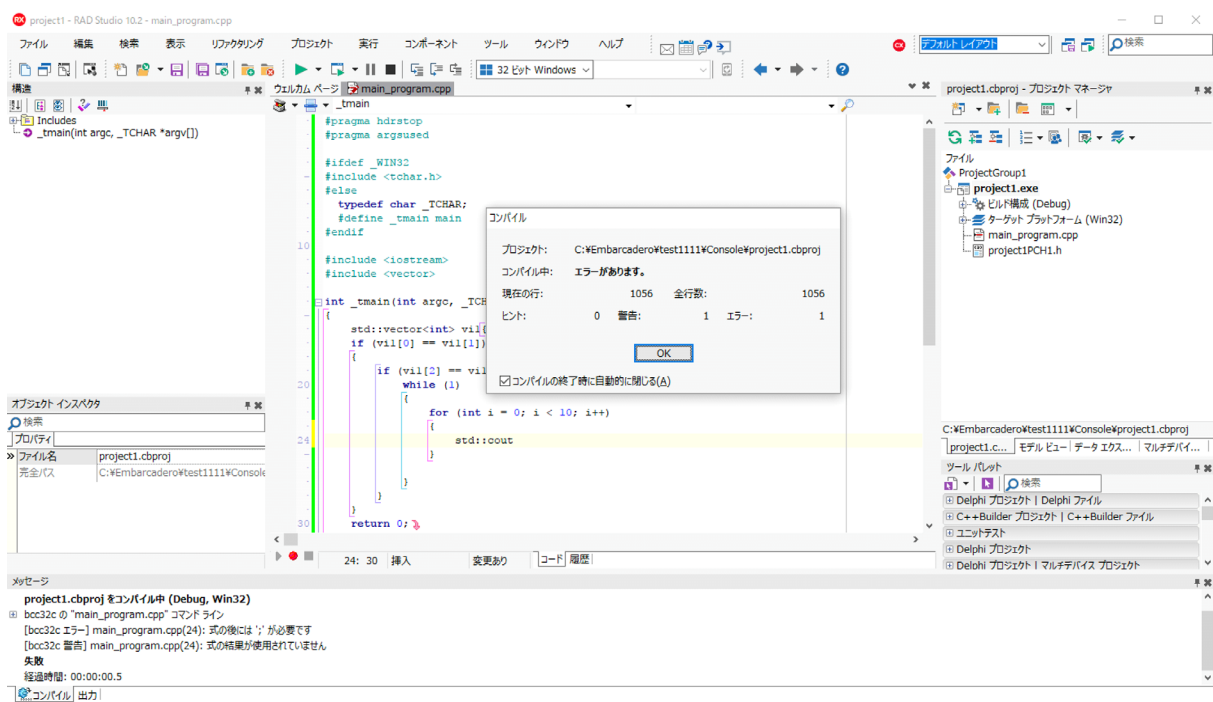


図 17 コンパイル実行

上記の図では、残念ながらコードの記述にミスがあり、コンパイルエラーになってしまったようです。BCC32C コンパイラを使っていたときは、コンパイルエラーが発生したら、エラーメッセージをひとつずつ確認しながら、エディタで該当する行を探さなければなりません。

C++Builder では、エラーメッセージが表示されるメッセージウィンドウとエディタが連動するので、エラーメッセージを確認するときに、同時に該当するコードを見つけることができます。

次の例では、式のうしろにセミコロン「;」の入力を忘れてエラーになってしまいました。エディタの該当する行が赤く表示されているので、すぐに修正すべき箇所を発見できます。

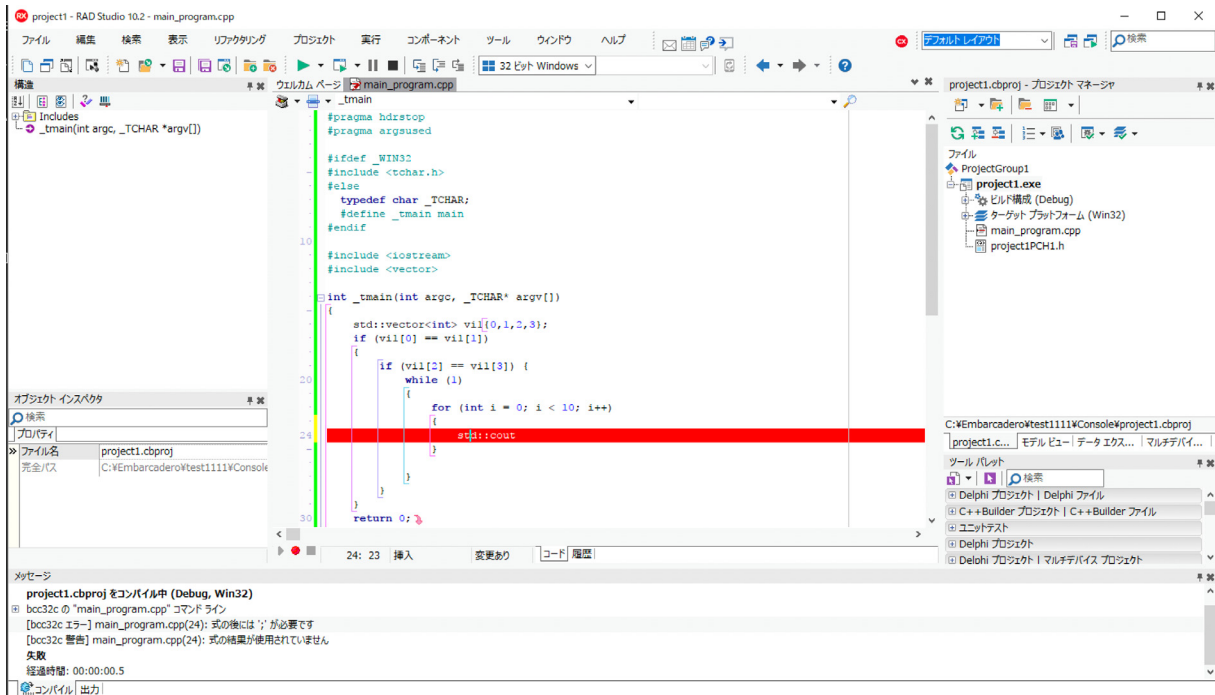


図 18 コンパイルエラー箇所の表示

デバッガを使う

エラー箇所を修正したら、再びプログラムをビルド／実行します。プログラム実行時には、ソースコードの特定箇所まで、プログラムを一時停止させる「ブレークポイント」を設定できます。

ブレークポイントを設定するには、停止させたい行のソースコードの左余白をクリックします。ブレークポイントでプログラムを停止させることで、変数の内容を確認したり、1 行ずつ実行して動きを確認することができます。

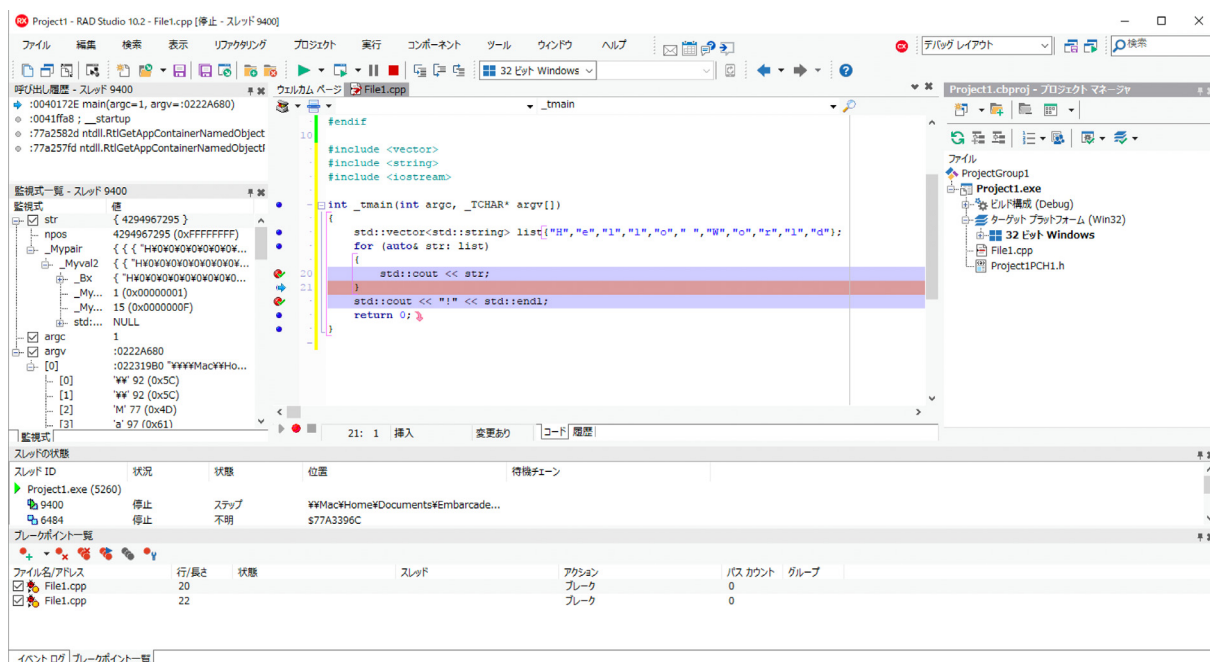


図 19 デバッガでプログラムを停止

C++Builder の製品情報、ならびに無料トライアル版、無料の Starter Edition のダウンロードについては、以下のページをご覧ください。

<https://www.embarcadero.com/jp/products/cbuilder>

Embarcadero、Embarcadero Technologies ロゴならびにすべてのエンバカデロ・テクノロジーズ製品またはサービス名は、Embarcadero Technologies, Inc.の商標または登録商標です。その他の商標はその所有者に帰属します。