C#
ミックスドプログラミング

北山洋幸◉著

■サンプルファイルのダウンロードについて

本書掲載のサンプルファイルは、一部を除いてインターネット上のダウンロードサービスからダウンロードすることができます。詳しい手順については、本書の巻末にある袋とじの内容をご覧ください。

なお、ダウンロードサービスのご利用にはユーザー登録と袋とじ内に記されている番号が必要です。そのため、本書を中古書店から購入されたり、他者から貸与、譲渡された場合にはサービスをご利用いただけないことがあります。あらかじめご承知おきください。

- 本書の内容についてのご意見、ご質問は、お名前、ご連絡先を明記のうえ、小社出版部宛文書（郵送または E-mail）でお送りください。
- 電話によるお問い合わせはお受けできません。
- 本書の解説範囲を越える内容のご質問や、本書の内容と無関係なご質問にはお答えできません。
- 匿名のフリーメールアドレスからのお問い合わせには返信しかねます。

本書で取り上げられているシステム名／製品名は、一般に開発各社の登録商標／商品名です。本書では、™ および ® マークは明記していません。本書に掲載されている団体／商品に対して、その商標権を侵害する意図は一切ありません。本書で紹介している URL や各サイトの内容は変更される場合があります。

はじめに

本書は、C# で開発したプログラムと C や C++ 言語などで開発したプログラムを協調させて動作させる方法を解説した書籍です。C# や C++ 言語の入門レベルを終えた方々やソフトウェア開発に長く従事してきた方々を対象とします。

C# は、C や C++ に比較して、簡単なコードで効率の良いプログラムを記述できます。新しいシステムやプロジェクトをゼロから開発する場合であれば、C# のみで開発するのは良い選択でしょう。しかし、システムが複雑で過去の資産に頼らざるをえない場合やオープンソースなどと連携させたい場合など、C# のみで完結できないことはよくあります。本書は、そのようなときに必要となるマネージドコードとアンマネージドコードを連携させることを主に解説します。

対象読者

- C# と C/C++ を融合させたい人
- C# から C/C++ で開発されたライブラリやオープンソースを利用したい人
- C や C++ で開発した過去の資産を C# から有効利用したい人

謝辞

出版にあたり、株式会社カットシステムの石塚勝敏氏に深く感謝いたします。

2020 年秋 新型コロナの流行で外出自粛中の自宅にて　北山洋幸

本書の使用にあたって

本書に掲載したプログラムは、無償の Visual Studio Community 2019 を使用して開発しています。また、プログラムの動作確認は Windows 10 Pro（64 ビット）で行っています。Windows 10 Home（64 ビット）でもほとんどのプログラムの動作を確認しています。

本書に記載されている URL は執筆時点のものであり、変更される可能性があります。リンク先が存在しない場合はキーワードなどから自分で検索してください。

本書の記述における用語の使用について説明します。

- カタカナ語の長音表記

  「メモリー」や「フォルダー」など、最近は語尾の長音符を付ける表記が一般的になっていますので、本書でもなるべく付けるようにしていますが、開発環境やドキュメントなどに従来の用語を使用している場合も多いため、長音符の有無は厳密には統一していません。

- .NET と .NET Framework と .NET Core

  本書で紹介するプログラムは、.NET Framework と .NET Core のどちらで開発しても構わないものが多いです。このため、.NET Framework と .NET Core を特別区別せず、.NET と表現する場合があります。

- DLL

  DLL はダイナミックリンクライブラリ（Dynamic Link Library、動的リンクライブラリ）の略です。

- オブジェクト

  インスタンスと表現した方が良い場合でも、オブジェクトと表現している場合があります。両方を、厳密に使い分けていませんので、文脈から判断してください。

目次

はじめに iii

第1章 はじめての連携 1

1.1 DLLの開発 2
1.1.1 __declspec 8
1.1.2 __stdcall 9
1.1.3 extern "C" 9
1.2 呼び出し側の開発 10
1.3 アンマネージド呼び出しの概要 18
1.3.1 関数名のエクスポート 19
1.4 .NET Framework と .NET Core 20
1.4.1 .NET Core 20
1.4.2 .NET Framework と .NET Core 20

第2章 ソリューションにまとめる 23

2.1 ソリューションにまとめる 23

第3章 ビット数の混在 33

3.1 ビット数の混在 33
3.2 プラットフォーム 34
3.3 64ビット環境で32ビットプロセス 35
3.4 64ビット環境で64ビットプロセス 37

第4章 データ型 39

4.1 マネージドとアンマネージドのデータ型 39
4.2 マネージドとアンマネージドのデータサイズ 40

4.3 データモデル......41
4.4 データサイズの比較......42
4.4.1 アンマネージドのデータサイズ......42
4.4.2 マネージドのデータサイズ......44

第5章 数値型をマネージドからアンマネージドへ渡す......47

5.1 Byte 型と Char 型......47
5.2 整数型......50
5.3 浮動小数点型......53

第6章 配列の受け渡し......57

6.1 配列の受け渡し......57
6.2 バッファ受け渡しの実例......61
6.3 参照渡しと配列のリサイズ......68

第7章 構造体の受け渡し......73

7.1 構造体の受け渡し......73
7.2 構造体ポインターを含む構造体の受け渡し......76
7.3 構造体を含む構造体の受け渡し......80
7.4 配列を含む構造体の受け渡し......83

第8章 文字列の受け渡し......87

8.1 文字列の受け渡し......87

第9章 アンマネージドからマネージド......91

9.1 アンマネージドからマネージドの変数をアクセス......91
9.1.1 byte 型と int 型......92

9.2　アンマネージドからマネージドを利用 94
9.2.1　DLL の開発 96
9.2.2　呼び出し側の開発 96
9.3　定義済みデリゲートで書き直し 98

第 10 章　DLL のメソッド名など 101

10.1　DLL のメソッド名を変更 101
10.2　.DEF ファイルを使ったエクスポート 103
10.2.1　呼び出し側の開発 104
10.2.2　DLL の開発 104
10.2.3　.DEF ファイルを使う場合の利点と欠点 108

第 11 章　GUI を持つ例 109

11.1　呼び出し側を WPF で開発 109
11.2　呼び出し側を Windows フォームで開発 116

第 12 章　WindowsAPI の呼び出し 123

12.1　WindowsAPI の呼び出し 123

第 13 章　DllMain 127

13.1　DllMain について 127
13.1.1　DllMain の引数 127
13.2　DLL の説明 129
13.2.1　data_seg の説明 130
13.2.2　DEF ファイル 131
13.2.3　SECTIONS 文 131
13.3　呼び出し側の説明 132

第 14 章 メモリー共有 135

14.1 DLL を使って通信 135

14.2 プロセス間のメモリー共有概論 135

14.3 DLL の説明 137

14.3.1 DEF ファイル 138

14.4 呼び出し側 A の説明 139

14.5 呼び出し側 B の説明 140

第 15 章 応用 143

15.1 OpenCL のプラットフォーム数を表示 143

15.2 OpenCL のプロフィールを表示 146

15.3 OpenCL で配列の乗算 154

15.4 OpenCV 160

15.4.1 DLL の説明 160

15.4.2 C# の説明 163

15.4.3 可変長配列に対応させる 172

15.4.4 呼び出し規約を変更 176

参考文献 180

索引 181

1 はじめての連携

マネージドとアンマネージドを連携させる例を紹介します。C# から C++ 言語などで開発した DLL を呼び出すときの前提を先に解説すると、読む気が失せてしまうでしょう。そこで、本章では、簡単な例題を示し、具体的に C# プログラムから C++ で開発した DLL を呼び出す方法を先に解説します。その例を参考に、順次 C# と C/C++ の連携について解説します。

紹介するプログラムは、C# から 2 つの値を渡し、DLL で 2 つの変数を加算し、それを C# 側へ返します。

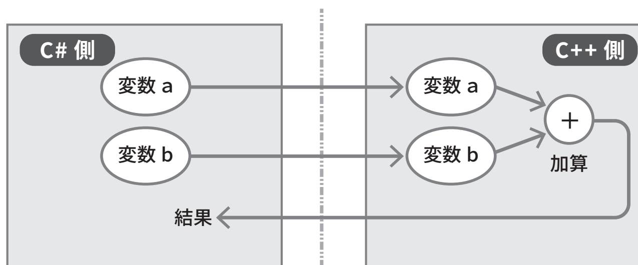

図1.1●プログラムの概要

なお、Windows 10 には 32 ビット版と 64 ビット版があり、64 ビット版の Windows 10 では、32 ビットと 64 ビットのアプリケーションソフトウェアを動作させることができます。この組み合わせを理解しておくことは重要ですが、最初に説明を行うと混乱しますので、本件についてはしばらくしてから解説します。本書では、基本的に 64 ビットの Windows 10 で 64 ビッ

トのアプリケーションソフトウェアを動作させることを前提に説明します。なお、これらの組み合わせについては、一通り C# から DLL を呼び出す方法を解説したあとで、説明します。

1.1 DLL の開発

まず、C# から呼び出される DLL の開発を段階を追って説明します。本書では Visual Studio Community 2019 を使用します。

（1）Visual Studio を起動し、［新しいプロジェクトの作成］を選びます。あるいは、［コード無しで続行］を選び、Visual Studio を起動したのち、メニューから［ファイル］→［新規作成］→［プロジェクト］を選んでも構いません。

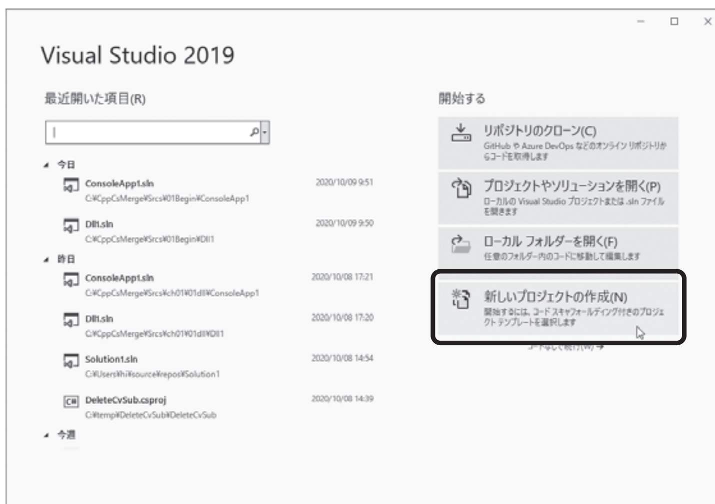

図1.2●新しいプロジェクトの作成

(2)「新しいプロジェクトの作成」ダイアログが現れますが、テンプレートが多いので、[C++]を選びます。

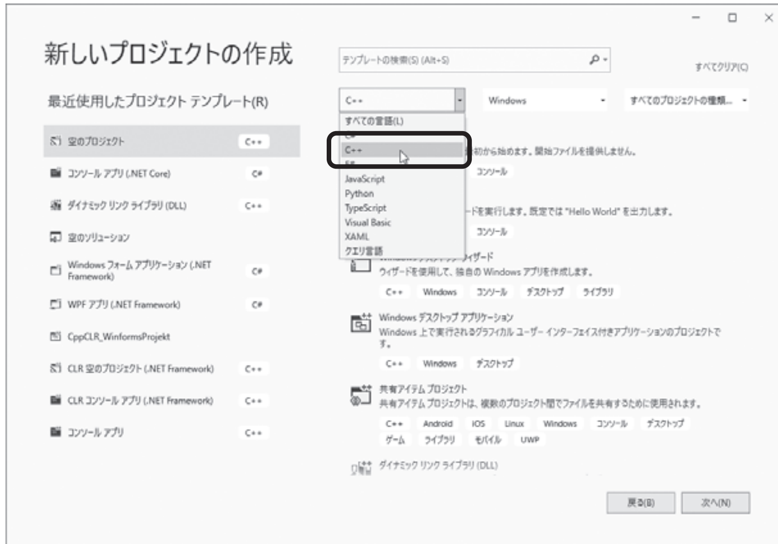

図1.3●C++用のテンプレートを表示

(3)C++ のみのテンプレートが現れますので、「空のプロジェクト」を選び、[次へ]を選択します。

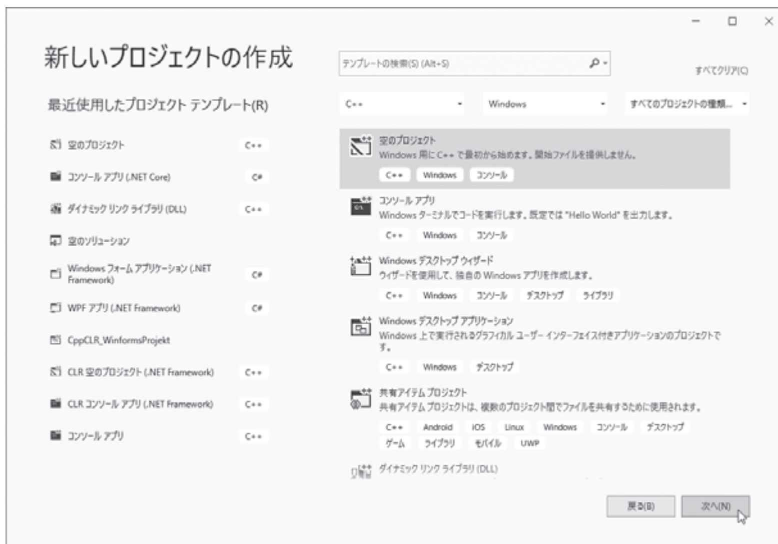

図1.4●「空のプロジェクト」を選ぶ

(4)プロジェクトの「場所」と「プロジェクト名」などを入力します。この例では、「場所」はデフォルトのまま、「プロジェクト名」は分かりやすい名前に変更します。

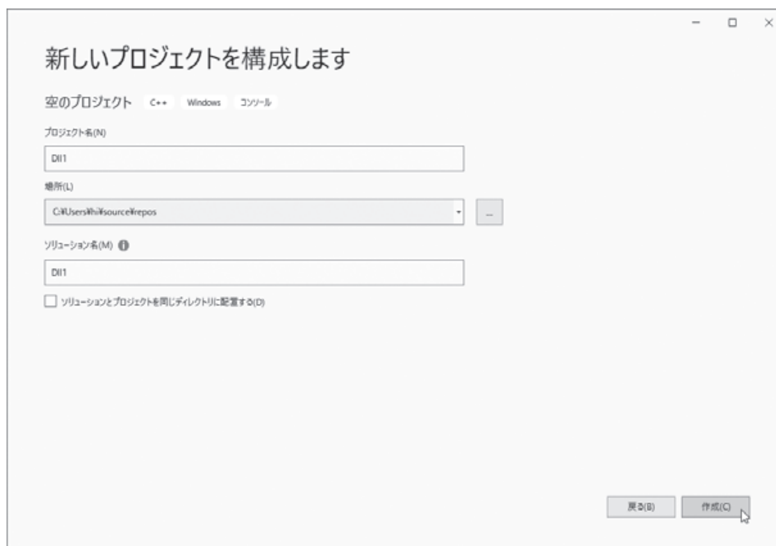

図1.5●プロジェクトの場所と名前などを入力

(5) すると、空のプロジェクトが作成されます。このプロジェクトはファイルなどを含みません。

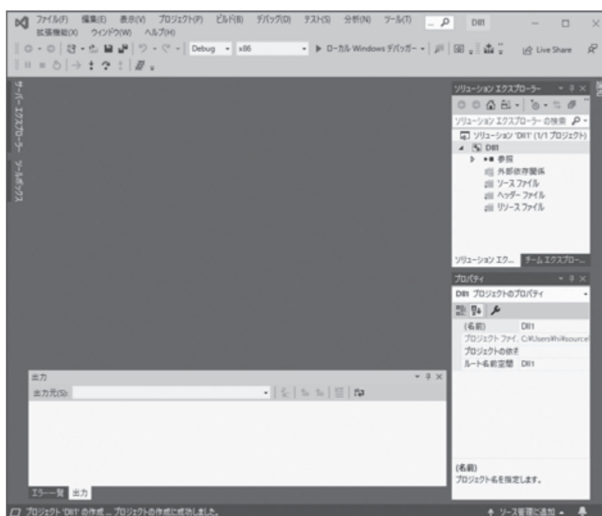

図1.6●空のプロジェクトが作成された様子

（6）このままでは DLL は 32 ビット用で生成されるため、［x86］を［x64］へ変更します。

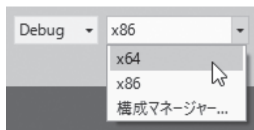

図1.7●［x86］を［x64］へ

（7）また、このままではプロジェクトは DLL ではないため、［プロジェクト］→［プロパティ］を選びます。

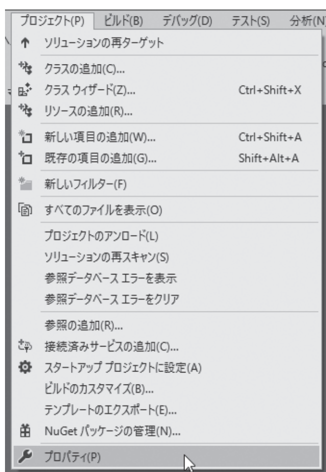

図1.8●［プロジェクト］→［プロパティ］を選ぶ

(8)［全般］→［構成の種類］のドロップダウンから［ダイナミックライブラリ (.dll)］を選びます。Debug と Release で同じ構成になるように、構成は［すべての構成］を選択します。

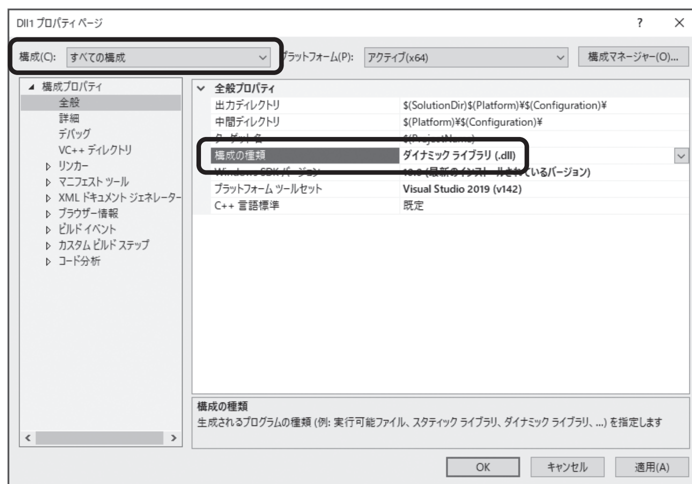

図1.9●［ダイナミックライブラリ (.dll)］を選ぶ

(9) このプロジェクトにソースファイルを追加します。ソリューションエクスプローラのソースファイル上でマウスの右ボタンを押し、［追加］→［新しい項目］を選びます。

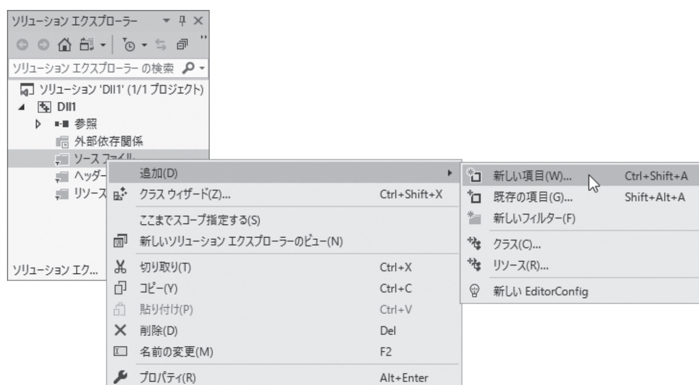

図1.10●新しい項目を追加

（10）「新しい項目の追加」ダイアログで「C++ ファイル (.cpp)」を選び、ソースファイル名を指定します。

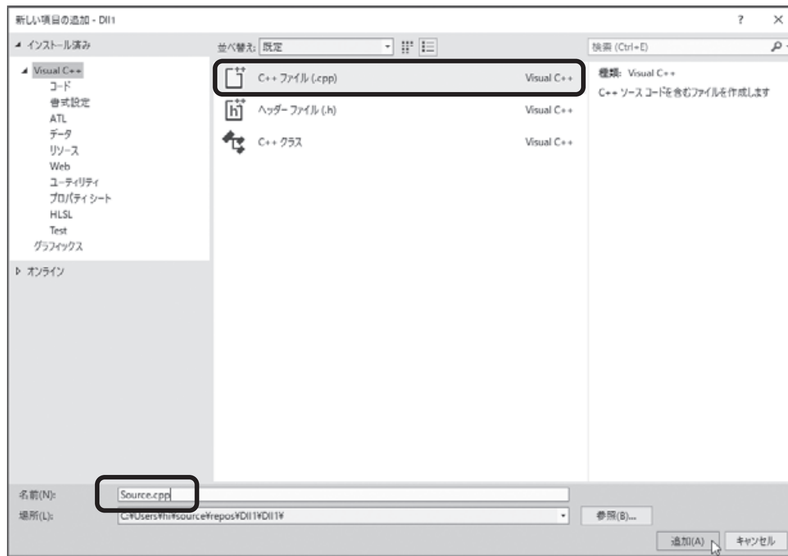

図1.11●ソースファイルを追加

（11）ファイルが追加され、そのファイルに、コードを記述した様子を示します。

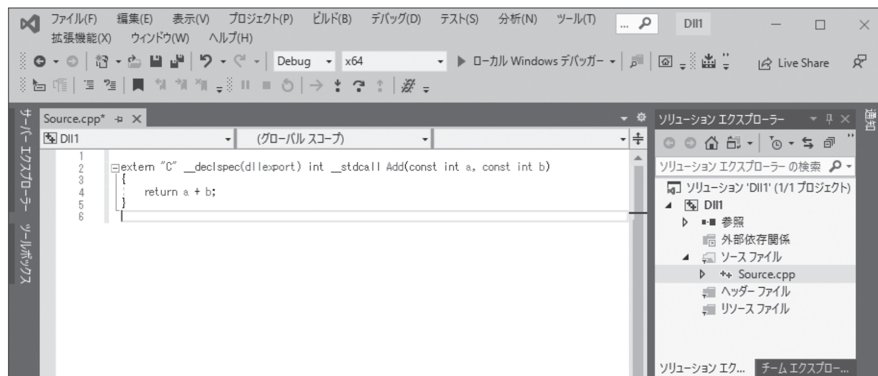

図1.12●追加されたファイルにコードを記述

（12）ビルドボタンを押して、DLL をビルドします。

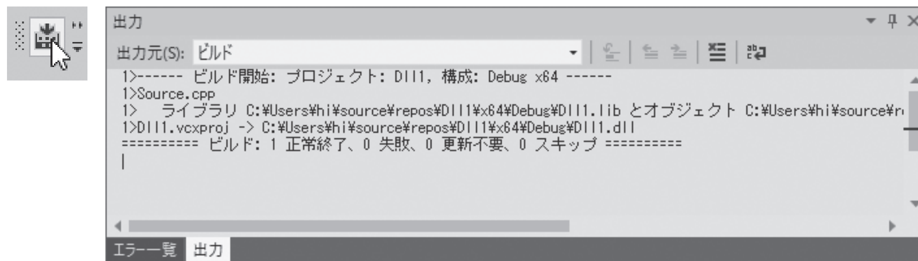

図1.13●DLLをビルド

これで 64 ビットの DLL が生成されます。以降に、ソースコードを示します。

リスト 1.1 ● Dll1 - Source.cpp

```
extern "C" __declspec(dllexport) int __stdcall Add(const int a, const int b)
{
    return a + b;
}
```

渡された 2 つの int 型変数を加算し、結果を呼び出し元へ返す単純な関数です。

1.1.1 __declspec

DLL のレイアウトは EXE とよく似ています。大きな違いは、DLL ファイルにはエクスポートテーブルが含まれているということです。エクスポートテーブルには、DLL が外部に対してエクスポートする関数の名前が含まれています。エクスポートテーブルに記述されたエクスポート関数のみが、別の実行可能ファイルからアクセスできます。エクスポートしていない DLL 内の関数は、その DLL 内でしか使えません。DLL から関数をエクスポートする方法には、次の 2 つがあります。

- 関数の定義に __declspec(dllexport) キーワードを使う方法
- モジュール定義ファイル（DEF）を作成して、DLL のビルド時に .DEF ファイルを使う方法

一般的には __declspec(dllexport) キーワードを使用します。.DEF ファイルを使う方法に

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

ついては後述します。

__declspec() は、関数宣言の好きな場所に置くことができます。たとえば、

```
__declspec(dllexport) int __stdcall Add(const int a, const int b)
```

でも、

```
__stdcall __declspec(dllexport) int Add(const int a, const int b)
```

でもかまいません。__declspec(dllexport) を使うメリットは、DEF ファイルのメンテナンスを行う必要がないことです。ただし、コンパイラが生成するエクスポート序数を管理することはできません。DLL の呼び出しに名前ではなく序数を使う方法がありますが、__declspec(dllexport) を使って関数をエクスポートした場合、序数を管理できません。

1.1.2　__stdcall

エクスポート宣言には、__stdcall が付加されています。__stdcall 呼び出し規約は、API やオープンソースで提供される DLL 関数などを呼び出すときに使用します。__stdcall 規約では、スタックは呼び出された側がクリアします。DLL 関数が __cdecl を使用している場合は、呼び出し側も、それに合わせる必要があります。__stdcall 呼び出し規約と __cdecl 呼び出し規約を混在して使用すると、スタックの処理に不都合が発生し、プログラムは正常に動作しません。呼び出し側と、呼び出され側の呼び出し規約は一致させなければなりません。本書は、__stdcall 呼び出し規約と __cdecl 呼び出し規約の実例も示します。

1.1.3　extern "C"

C++ で開発したプログラムは、外部シンボルを装飾するため実際の関数名と異なってしまいます。この命名規約はバージョンによっても異なるため、少々やっかいな問題を引き起こします。そこで、extern "C" 構文により、C++ の名前装飾を取りやめる方法を採用します。extern "C" 構文は、C++ からほかの言語への呼び出しを整合させたり、ほかの言語から呼び出される C++ 関数の名前付け規約を変えたりするために使います。ただし、extern "C" は C++ でしか

使えません。C++ コードが extern "C" を使っていない場合は、名前の装飾を調べ、ほかの言語からその名前を使用しない限り、該当する C++ 関数を呼び出すことができません。どのように関数名が装飾されたかは、DUMPBIN などのユーティリティを使って調べることができます。

1.2 呼び出し側の開発

次に、DLL を呼び出す側のプログラムを開発します。呼び出し側はコンソールアプリケーションで開発します。

(1) Visual Studio を起動し、［新しいプロジェクトの作成］を選びます。あるいは、［コード無しで続行］を選び、Visual Studio を起動したのち、メニューから［ファイル］→［新規作成］→［プロジェクト］を選んでも構いません。

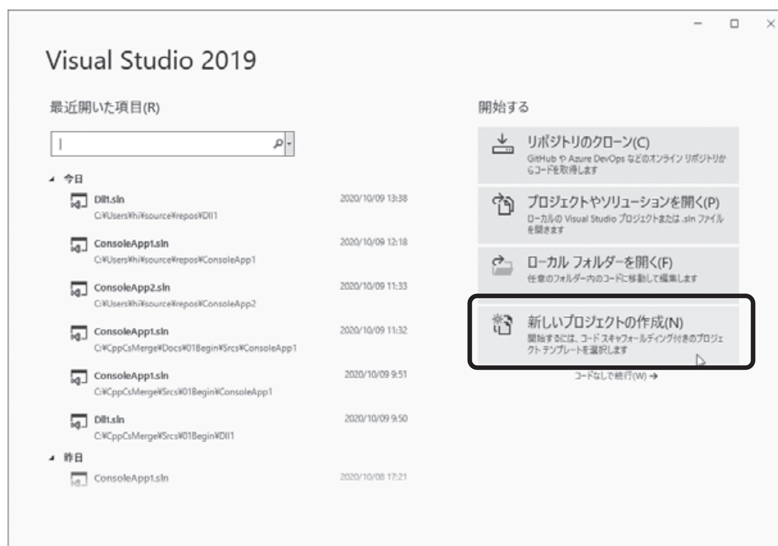

図1.14●新しいプロジェクトの作成

(2)「新しいプロジェクトの作成」ダイアログが現れますが、あまりにもテンプレートが多いので、［C#］を選びます。

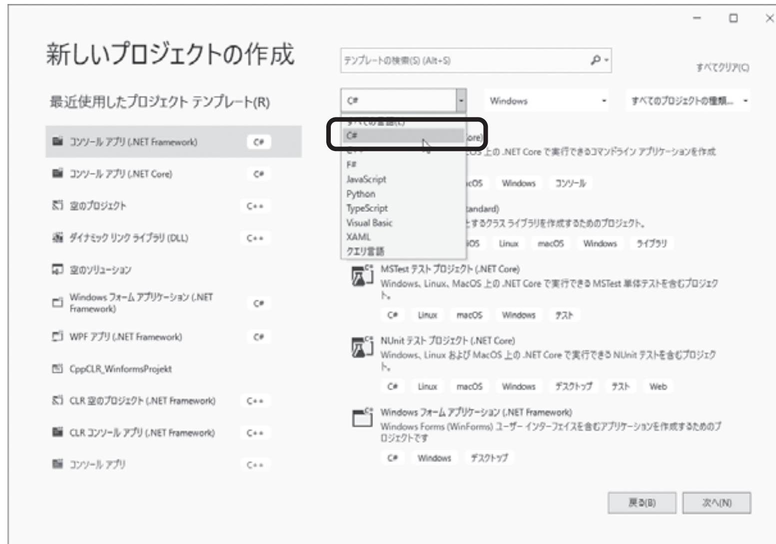

図1.15●C#用のテンプレートを表示

（3）C# のみのテンプレートが現れますので、「コンソール アプリ (.NET Framework)」を選び、［次へ］を選択します。

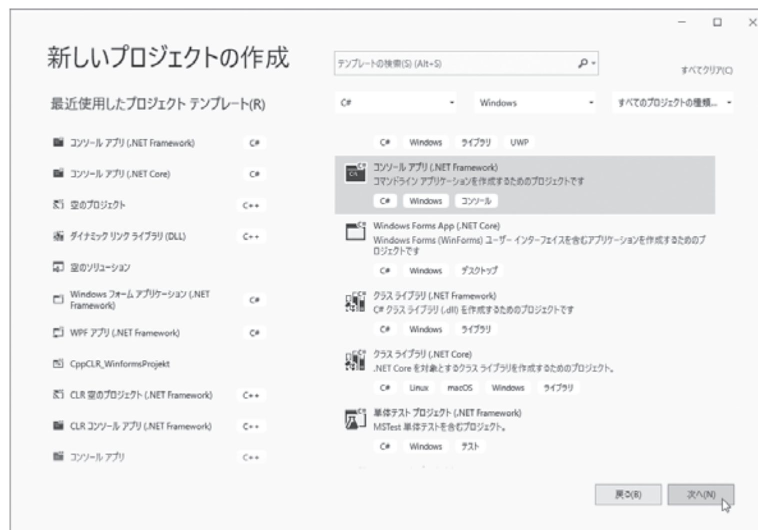

図1.16●「コンソール アプリ(.NET Framework)」を選ぶ

.NET Core

　ここでは、［コンソール アプリ (.NET Framework)］を選びますが、［コンソール アプリ (.NET Core)］を使用しても構いません。.NET Framework と .NET Core の違いについては、簡単に後述します。

（4）プロジェクトの場所と名前などを入力します。

図1.17●プロジェクトの場所と名前などを入力