УДК 378.046.4

Разработка учебных примеров по смешанному программированию на языках C# и C++

Штанюк Антон Александрович
Нижегородский государственный технический университет им.Р.Е.Алексеева
доцент

Аннотация
Рассматриваются вопросы разработки учебных примеров с использованием языков программирования C# и C++ в рамках одного программного проекта.
Ключевые слова: программное обеспечение, курсы повышения квалификации, системы контроля версий.

Development of training examples on mixed programming in C# and C++

Anton Shtanyuk
Nizhny Novgorod State Technical University
Associate professor

Abstract
The issues of using software as part of continuing education courses in IT are considered. The set of free software for the educational process is described.
Keywords: freely distributed software, continuing education courses, version control systems

Понятие «смешанное программирование» в контексте данной работы представляет собой использование более одного языка программирования при разработке модулей готовой программы, в рамках одного программного проекта. В настоящее время, существует множество приложений, построенных с использованием нескольких языков. Данный подход широко используется в промышленном программировании, поскольку предоставляет разработчикам дополнительную возможность в выборе средств и технологии реализации программного кода. Часто модули, написанные на разных языках, могут содержать уникальный код, который было бы слишком дорого переписывать. К тому же, разные по функционалу части программы, создаются с привлечением подходящих инструментов.

Между тем, при обучении программированию, доминируют моноязычные подходы. Как правило, в учебном курсе используется один язык, на котором пишут все модули и одна среда разработки, с помощью которой программу доводят до реализации. Это может быть вызвано несколькими причинами.

- с точки зрения технического сопровождения проще и удобнее поддерживать моноязычные среды;

- с точки зрения преподавательского состава упор делается на один знакомый язык программирования;

- с точки зрения обучаемого проще создавать моноязычную программу, поскольку меньше усилий тратится на отладку и достижения межмодульного взаимодействия.

Можно отметить, что при использовании нескольких языков и форматов компонуемых модулей, наибольшую сложность представляют именно способы взаимодействия этих модулей, вызванные более сложными механизмами компоновки и несоответствием типов данных, принятых в разных языках программирования.

В качестве учебного примера рассмотрим взаимодействие модулей, написанных на популярных языках C# и C++. Данные языки имеют широкое распространение как в промышленной разработке, так и в обучении. Наш учебный пример будет состоять из двух модулей: динамической библиотеки, написанной на C++ и консольной программы на C#.

В чем может состоять практическая польза от такого сочетания? Дело в том, что на языке C#, благодаря его тесной зависимости от платформы .NET, довольно удобно разрабатывать интерфейсную, высокоуровневую часть приложения. Библиотека WinForms с поддержкой окон и элементов управления, многочисленные модули с поддержкой функционального подхода, готовые компоненты для решения множества задач, являются прекрасным строительным материалом для приложений любого уровня сложности. И при этом, реализация многих алгоритмов обработки данных, по-прежнему более предпочтительна на языке C++, предоставляющем низкоуровневый доступ к памяти, механизм указателей и готовые реализации. Поэтому, в учебном проекте будем использовать оба языка.

На первом шаге необходимо разработать прототип приложения, в котором не будет полезного функционала, но будет показан механизм взаимодействия разноязычных модулей. Построение модулей будем производить в командной строке Windows. В системе был установлен пакет Visual Studio .NET 15 CE.

Приведем содержимое файлов *TestDLL.h* и *TestDLL.cpp*, входящих в состав модуля TestDLL:

```
// TestDLL.h
#pragma once

extern "C" {
   _declspec(dllexport) double Add(double x, double y);
}

// TestDLL.cpp
#include "TestDll.h"

extern "C" {
   _declspec(dllexport) double Add(double x, double y)
   {
```

```
      return x+y;
   }
}
```

Кратко поясним содержимое этих файлов.

Использование конструкции `extern "C"` гарантирует, что компилятор C++ не изменит имя функции, что важно при связывании со внешними модулями.

Конструкция `_declspec(dllexport)` позволяет экспортировать, предоставлять содержимое динамической библиотеки для внешнего использования. В данном примере мы собираемся предоставить доступ к единственной функции *Add*, осуществляющей сложение двух вещественных чисел.

Для построения модуля в виде динамической библиотеки сформируем команду:

```
     cl.exe /LD TestDLL.cpp /link /libpath:"C:\Program Files
(x86)\Microsoft Visual Studio 14.0\VC\lib" /libpath:"C:\Program Files
(x86)\Windows Kits\8.1\Lib\winv6.3\um\x86" /libpath:"C:\Program Files
(x86)\Windows Kits\10\Lib\10.0.10240.0\ucrt\x86" /OUT:"TestDLL.dll"
/DLL
```

Большую часть содержимого этой команды занимают опции компоновщика с путями к различным библиотечным файлам (*/libpath*). Эти опции добавлялись последовательно, когда команда выдавала сообщении о невозможности найти определенный файл с расширением *.lib*. Очевидно, что в другой системе при наличии иных версий пакетов разработчика и Windows SDK, пути будут другими.

В результате построения был получен файл динамической библиотеки *TestDLL.dll*, расположенный в текущем каталоге проекта.

Вторым модулем проекта будет выступать консольное приложение на языке C#, вся задача которого сводится к вызову функции из динамической библиотеки.

```
// Файл CheckDLL.cs
using System;
using System.Text;
using System.Runtime.InteropServices;

namespace CheckDll
{
   class Program
   {
      [DllImport(@".\TestDll.dll",
        CallingConvention = CallingConvention.Cdecl)]

        static extern double Add(double x,double y);
      static void Main(string[] args)
      {
         Console.WriteLine(Add(3.0,2.0));
```

```
        }
    }
}
```

В данном приложении осуществляется импорт функции из внешней библиотеки, при это указывается местоположение dll-файла и соглашение о передаче параметров. Данный вопрос имеет важное значение, поскольку программы, написанные на разных языках, могут использовать свой порядок передачи аргументов в функцию. Для корректной работы с динамическими библиотеками, нужно придерживаться одного порядка (в данном случае используется порядок, принятый в C).

Компиляцию программы можно осуществить также из командной строки:

```
csc.exe -platform:x86 CheckDLL.cs
```

Существенным моментом при построении и библиотеки и приложения, является согласованность бинарных образов (сборок). В приведенном примере, используются настройки для платформы x86 (32-разрядная версия). Если хотя бы один модуль будет собран под другую платформу, то взаимодействия не произойдет. C# выбрасывает исключение *System.BadImageFormatException*.

После успешной сборки, можно запустить выполняемый файл консольного приложения и увидеть на экране число 5.

После успешного построения и испытания прототипа, создадим учебный проект, в котором на разных интервалах подсчитывается количество простых чисел и по результатам исследования строится график в виде столбчатой диаграммы. В исходный код динамической библиотеки добавляются две функции: *isPrime* для тестирования числа на простоту и *primeCount* для подсчета количества простых чисел на заданном интервале.

```
#include "TestDll.h"
extern "C" {
	_declspec(dllexport) int isPrime(unsigned long number)
	{
		if (number < 2) return 0;
		for (unsigned long i = 2; i*i <= number; i++) {
			if (number%i == 0)
				return 0;
		}
		return 1;
	}
	_declspec(dllexport) unsigned long primeCount(unsigned long left,
                                                  unsigned long right)
	{
		unsigned long count = 0;
		for (unsigned long i = left;i <= right;i++)
			if (isPrime(i))
				count++;
		return count;
```

```
    }
}
```

Директивы для импорта внешних функций учитывают соответствие системы типов языков C# и C++:

```
[DllImport(@".\TestDll.dll",
CallingConvention =CallingConvention.Cdecl)]
static extern uint primeCount(uint x, uint y);
```

Далее, создается стандартное приложение WinForms с главной формой, на которую наносятся несколько элементов управления: метки, поля ввода, кнопки и объект ‘’Диаграмма’’. Пользователь задает начало и конец числового отрезка для исследования и величину интервала, на которые разбивается исследуемый отрезок. В цикле для каждого интервала вызывется функция *countPrime* из динамической библиотеки и, возвращаемое ей число простых чисел на интервале, заносится в данные диаграммы.

Компиляция программы с использованием библиотеки WinForms производится следующей командой:

```
    csc.exe FormPrime.cs FormPrime.Designer.cs Program.cs
    /R:"C:\Program Files (x86)\Reference
Assemblies\Microsoft\Framework\.NETFramework\v4.5.2\System.Windows.For
ms.DataVisualization.dll"
```

В построении принимают участие три файла и сборка, отвечающая за работу со стоблчатой диаграммой. *FormPrime.cs* содержит основной код формы (обработчики событий элементов управления, вызовы функций из DLL), *FormPrime.Designer.cs* содержит основные определения элементов формы. Файл *Program.cs* содержит код запуска формы.

Приведем результаты выполнения расчетов в приложении WinForms:

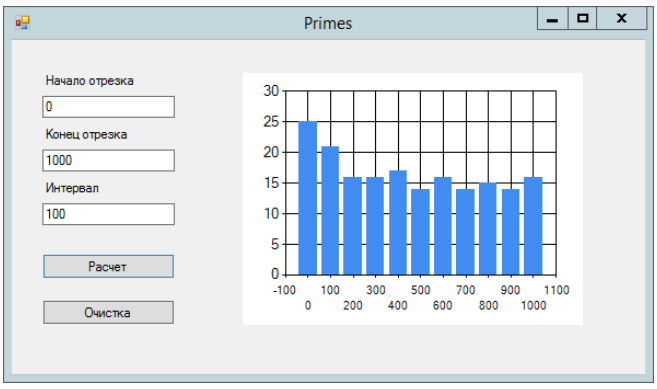

Рисунок 1. Результаты подсчета простых чисел на интервалах

В заключении можно подчеркнуть, что создание учебных примеров для изучения смешанного программирования на нескольких языках, может стать хорошим дополнением к существующим курсам по разработке ПО. Эти примеры позволяют глубже изучить механизмы взаимодействия модулей в

операционных системах, систему типов различных языков, а также параметры компоновки и настройки различных типов проектов в используемой IDE.

Библиографический список

1. Штанюк А.А. Обучение программированию на языках C/C в рамках курсов повышения квалификации // Международное научное издание «Современные фундаментальные и прикладные исследования». 2014. №1(12). С. 52 - 55.
2. Прайс М. C# 7 и .NET Core. Кросс-платформенная разработка для профессионалов. СПб.: Питер, 2018.
3. Рихтер Дж. Windows для профессионалов. Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows. СПб.: Питер, 2001.