D. Ivashchev, V. Gerasymov, O. Hurko

THE COMPATIBILITY OF CODE WRITTEN IN C# AND C++

During software development, situations arise when it is necessary to call code that has already been implemented in another programming language. Previously, this problem referred to, for example, the use of libraries written in C/C++ in Java applications, which is solved via the JNI mechanism [1]. Then, the task of using such C/C++ libraries in applications written in C# was also added. Let's take a closer look at the latter task. There are several ways to develop software in C# when there is a part of the required functionality already implemented in C++.

1. Rewrite the code completely in C#, adapting it according to the capabilities of C#.
2. Develop the program again, regardless of the existing C++ implementation.
3. Combine languages, i.e. use the ready-made C++ implementation in the C# project.

Comparing the options for using a ready-made implementation of C++, it should be noted that each of them can be used in software development. However, there are important points associated with the first option:

- C++ provides more opportunities for a programmer than C#, which is why code written in C++ can be much more efficient.
- The difference in memory management between C++ and C# can lead to difficulties in adapting code. In C++, for example, the developer usually manages memory allocation and deallocation independently, whereas in C#, a garbage collector does this.
- The significant difference in approaches to implementing multithreading in C++ and C# may require essential code modifications during such adaptation.

That is, it is necessary to delve into and thoroughly explore the work of C ++ code in order to qualitatively rewrite the code in C#. This can make things difficult if the C++ code is complex enough.

When using the second option, it will be necessary to re-solve tasks that have already been solved and tested by other developers. Therefore, this option is also inefficient.

The last, third option is the most efficient as it utilizes pre-existing code without the need for modification or rewriting. It only requires "connecting" to the C++ code in order to use it in C# code.

For such purposes, Microsoft recommends using the attributes for import/export dllexport and dllimport – respective extensions for C/C++ languages. They can be used to export and import functions, data, and objects to and from a DLL library written in C/C++ [2].

However, such a decision requires familiarity with the library interface – method names, parameters, and return values. It takes some to create the wrapper code, i.e., code that will be called in a C# project and serve as a bridge to the C/C++ code, passing parameters and results in the appropriate directions.

The process of creating wrapper classes can be automated using the SWIG tool – Simplified Wrapper and Interface Generator [3]. SWIG is a free tool for connecting programs and libraries written in C/C++ to programs and libraries written in other programming languages (TCL, Perl, Python, Ruby, PHP, Java, C#). To use this tool, the library developer creates a file with a description of the exported functions.

```
/* File : example.i */
%module example

%inline %{
extern int gcd(int x, int y);
extern double Foo;
%}
```

SWIG generates source code for linking C/C++ and a desired programming language and creates wrapper classes. After that, the developer can use the created wrapper classes in the code of their project to call functions from the C/C++ library.

Based on usage experience, it should be noted that despite SWIG being a proven code generator, errors can occur in wrapper classes. Additionally, such wrapper classes require the dll library to be located in the same directory as the executable file within the file system.

It is recommended to separate the generated code into a separate project and move the wrapper code for automatic DLL binding to a specific, designated directory, such as the Resources folder in C#. This will facilitate project setup and testing.

Below is a fragment of the generated file for the above-described functions after some minor modifications.

```
class examplePINVOKE
{
    public const string addressDll = "example.dll";
    protected class SWIGExceptionHelper {

    public delegate void ExceptionDelegate(string message);
...
  [global::System.Runtime.InteropServices.DllImport(addressDll,
EntryPoint="CSharp_gcd")]
  public static extern int gcd(int jarg1, int jarg2);

  [global::System.Runtime.InteropServices.DllImport(addressDll,
EntryPoint="CSharp_Foo_set")]
  public static extern void Foo_set(double jarg1);

  [global::System.Runtime.InteropServices.DllImport(addressDll,
EntryPoint="CSharp_Foo_get")]
  public static extern double Foo_get();
}
```

In conclusion, we would like to emphasize that the issue of interaction between code written in different programming languages, particularly C/C++ and C#, is presented. Moreover, the features of working with the SWIG library on conferred, which undoubtedly helps to reduce the development time of wrapper classes and, under certain conditions, enables the immediate start of developing subsequent modules after configuring and testing the wrapper code.

REFERENCES

1. JNI APIs and Developer Guides. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/jni/
2. dllexport, dllimport | Microsoft Learn. URL: https://learn.microsoft.com/en-us/cpp/cpp/dllexport-dllimport?view=msvc-170.
3. Simplified Wrapper and Interface Generator. URL: https://www.swig.org/

K. Kibalnik, R. Yecha, N. Kondratiuk, O. Posudiievska

NEW FOOD PRODUCTS IN THE COMBAT OF DIABETES

Diabetes mellitus is a disease of the endocrine system, which is characterized by a violation of the absorption of glucose in the body [1] and is one of the most common chronic diseases, which requires constant monitoring of health and adherence to a special diet. The diet of diabetic patients should include foods containing vitamins, minerals and antioxidant substances in increased quantities, but at the same time, the amount of sugar and carbohydrates should be limited. Unfortunately, such products and dishes that meet the requirements of dietary nutrition for diabetes do not have a very pleasant taste, so the main goal of modern food offerings should be the products with a powerful health-improving function and high organoleptic indicators.

One of these products is fruit-berry puree, which can be used as a separate dish, as well as a sauces for other dishes in the daily diets of patients with diabetes. It has been proved [2] that the use of berries and fruits stabilizes the level of glucose in the blood, as it contains a sufficient amount of vitamins, mineral compounds and antioxidants. Adding sweeteners to purees can solve the taste problem by giving products sweetness without involving the insulin machine.

When developing fruit-berry purees with the use of sweeteners, the need for the content of dietary fibers, such as pectins, was taken into account. The amount of beneficial substances that regulate the health status of people with diabetes has been increased by adding dry powders and extracts of fruits and berries. As a result of such