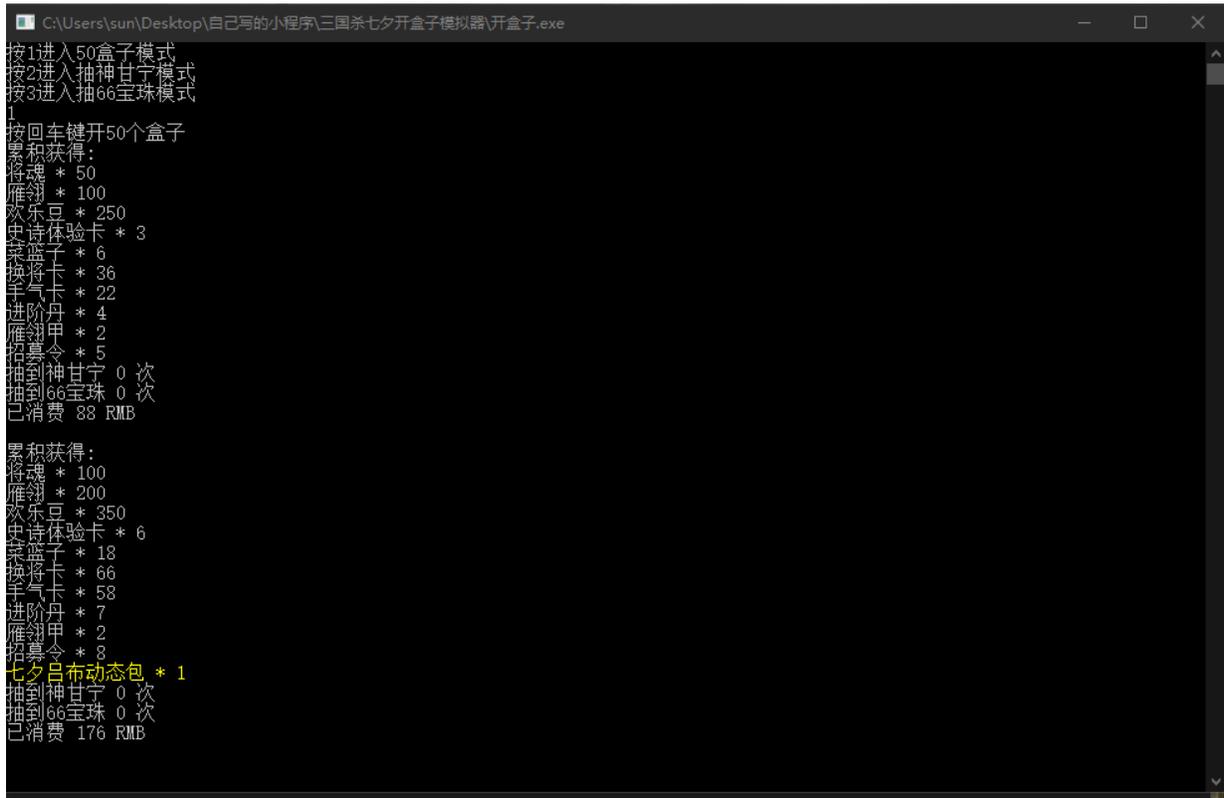


不知道各位刚开始学编程的朋友是否和我一样，每当看到黑白的编译器窗口，总会觉得缺了些什么。即使是花费大量时间写了非常复杂的程序，依旧绕不开黑白的输出界面。这样的程序真的适合给别人用吗？

我曾今有一段时间非常喜欢打三国杀，为了获得武将氪了不少，而氪的钱主要花费在了开盒子上，点一次 88，非常离谱。于是我就想，不如写一个开盒模拟器过过瘾：



```
C:\Users\sun\Desktop\自己写的小程序\三国杀七夕开盒子模拟器\开盒子.exe
按1进入50盒子模式
按2进入抽神甘宁模式
按3进入抽66宝珠模式
1
按回车键开50个盒子
累积获得:
将魂 * 50
雁翎 * 100
欢乐豆 * 250
史诗体验卡 * 3
菜篮子 * 6
换将卡 * 36
手气卡 * 22
进阶丹 * 4
雁翎甲 * 2
招募令 * 5
抽到神甘宁 0 次
抽到66宝珠 0 次
已消费 88 RMB

累积获得:
将魂 * 100
雁翎 * 200
欢乐豆 * 350
史诗体验卡 * 6
菜篮子 * 18
换将卡 * 66
手气卡 * 58
进阶丹 * 7
雁翎甲 * 2
招募令 * 8
七夕吕布动态包 * 1
抽到神甘宁 0 次
抽到66宝珠 0 次
已消费 176 RMB
```

于是开盒模拟器 1.0 就诞生了。

我将这款模拟器分享给了一些杀友，然而效果并不是很好。虽然它能够正确的模拟每个道具出现的概率以及需要氪多少，但是界面太粗糙了，在这个互联网时代，又有谁会愿意使用这样粗糙界面的程序呢？

于是，开盒模拟器 2.0 诞生了：

Form1

本次获得:

- 将魂*50
- 雁翎*100
- 欢乐豆*150
- 史诗体验卡*4
- 菜篮子*6
- 换将卡*28
- 手气卡*40
- 进阶丹*2
- 雁翎甲*1
- 招募令*2
- 史诗宝珠碎片*1

累积获得:

- 将魂*50
- 雁翎*100
- 欢乐豆*150
- 史诗体验卡*4
- 菜篮子*6
- 换将卡*28
- 手气卡*40
- 进阶丹*2
- 雁翎甲*1
- 招募令*2
- 史诗宝珠碎片*1

开神甘宁

开66宝珠

元宝: 109112

氪金999RMB

累计氪金: 999 RMB

累计消费: 8888 元宝

累计开盒: 50 个

重置

获得神甘宁次数: 0

66宝珠次数: 0

作者: WolfeGump

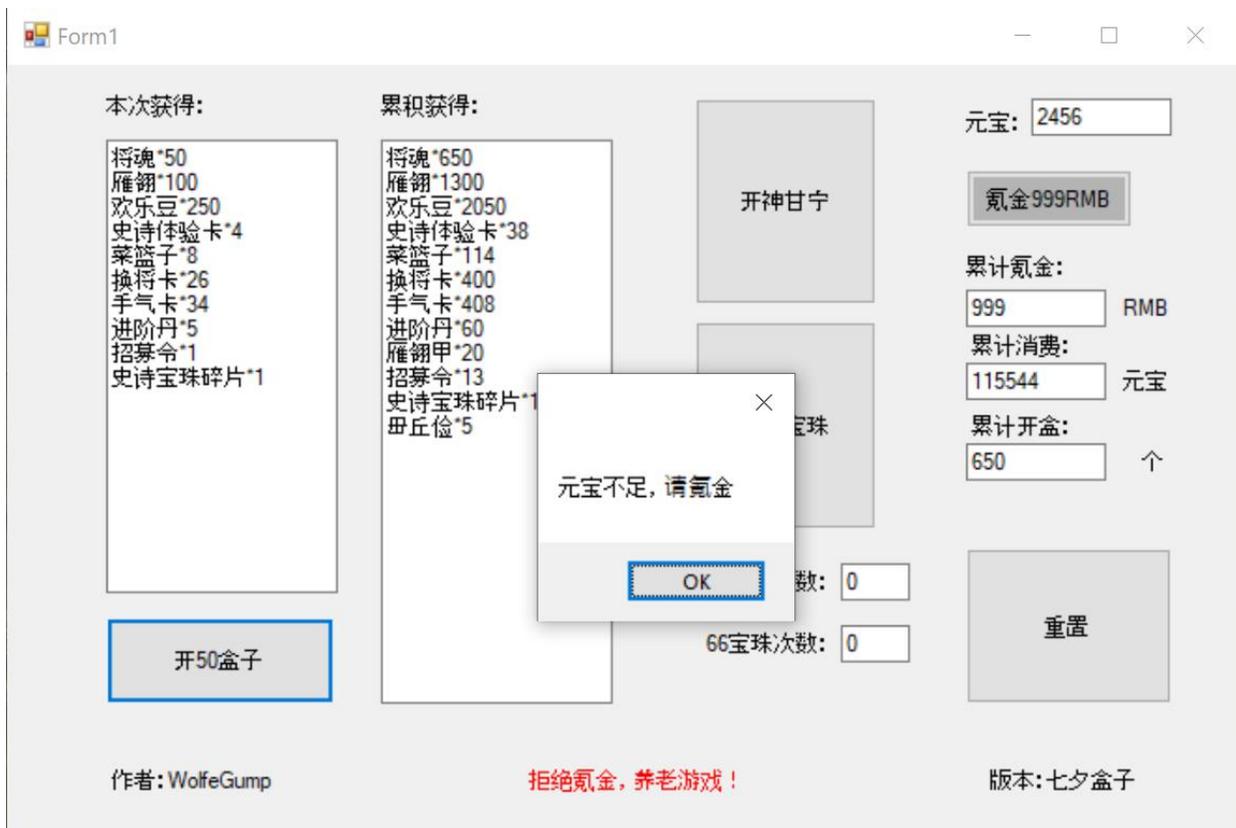
拒绝氪金, 养老游戏!

版本: 七夕盒子

这款模拟器更加真实的还原了充值优惠，是的，一次氪金 999 可以获得 118000 元宝而不是 99900，并且 50 盒子的价格调整为 8888 元宝，可以使得计算结果更加精确。

即使没有说明书，用户也能够理解各个按钮的作用，各个输出框输出了什么内容，点击氪金就能够增加元宝，点开盒子就能够打开盒子等等。

另外，在元宝不足的时候，这款模拟器甚至还可以：



善解人意的提醒你该氪金了!

那么以上只是例子，这篇推文不是用来介绍我的模拟器的哈哈哈，那么就由我来给大家介绍，如何用 Visual Studio 2019 就能写出这样一个带图形用户界面（GUI）的程序吧！

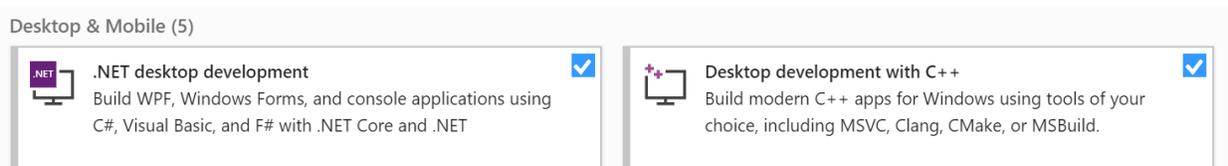
大家可能会觉得这会很难，其实一点也不，即便是编程新手，在读完这篇推文后也能够写出这样的程序！

那么首先，你需要安装 Visual Studio 2019，需要注意的是 Visual Studio 2019 和 Visual Studio Code 是不一样的，该网站可以直接前往下载：

[Visual Studio Community 2019 - Free IDE and Developer Tools \(microsoft.com\)](https://visualstudio.microsoft.com/zh-hans/whats-new/vs-2019/)

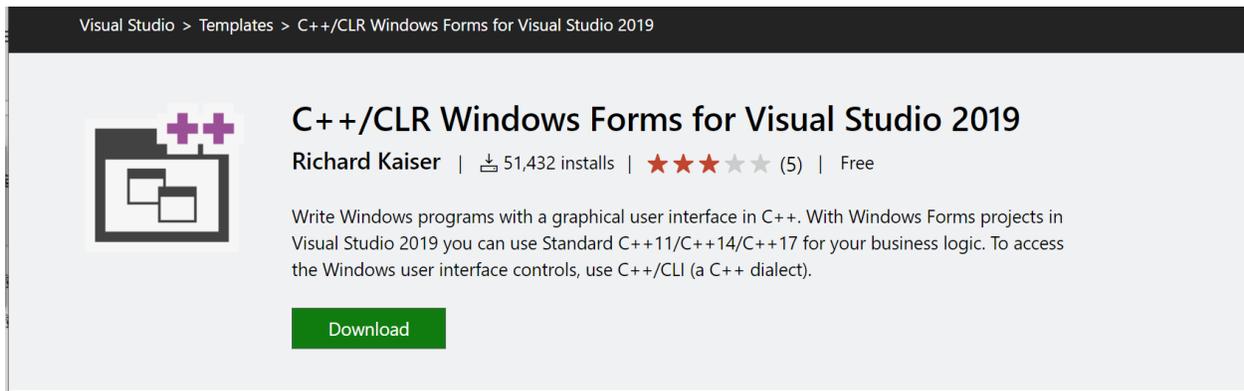
Visual Studio 的账号或是密钥也是可以通过 UCI 的学生账号免费获取的。

在安装完 VS 之后，你还要确保安装以下开发工具：



最后，在该网站下载最后一个“扩展包”就可以啦！

[C++/CLR Windows Forms for Visual Studio 2019 - Visual Studio Marketplace](#)



Visual Studio > Templates > C++/CLR Windows Forms for Visual Studio 2019

C++/CLR Windows Forms for Visual Studio 2019

Richard Kaiser | 51,432 installs | ★★★★★ (5) | Free

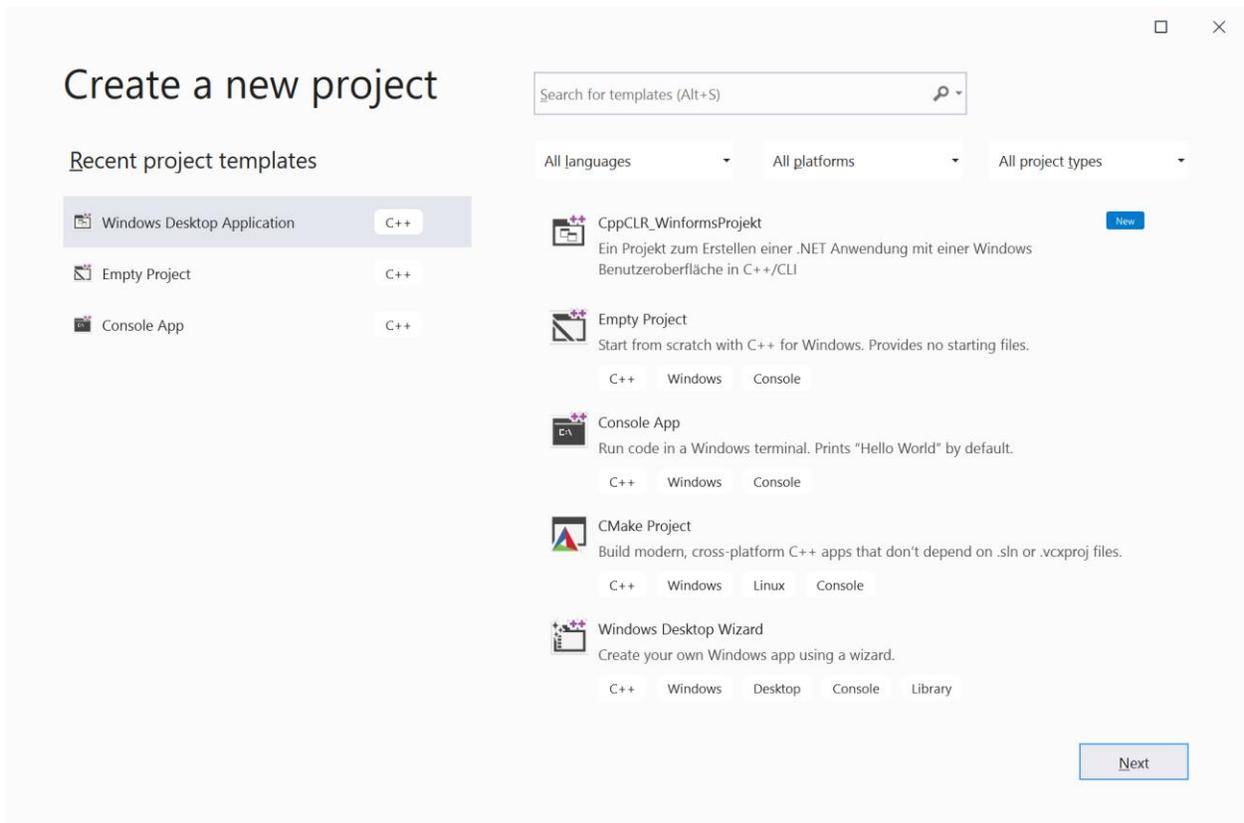
Write Windows programs with a graphical user interface in C++. With Windows Forms projects in Visual Studio 2019 you can use Standard C++11/C++14/C++17 for your business logic. To access the Windows user interface controls, use C++/CLI (a C++ dialect).

[Download](#)

直接点击下载然后安装即可。

那么万事俱备，接下来只要打开 VS 创建一个新的 project 就可以了！

选择刚刚下载好的模板，找不到的话直接搜索 CppCLR 就能出来了：



Create a new project

Search for templates (Alt+S)

All languages | All platforms | All project types

Recent project templates

- Windows Desktop Application (C++)
- Empty Project (C++)
- Console App (C++)

CppCLR_WinformsProjekt (New)
Ein Projekt zum Erstellen einer .NET Anwendung mit einer Windows Benutzeroberfläche in C++/CLI

Empty Project
Start from scratch with C++ for Windows. Provides no starting files.
C++ Windows Console

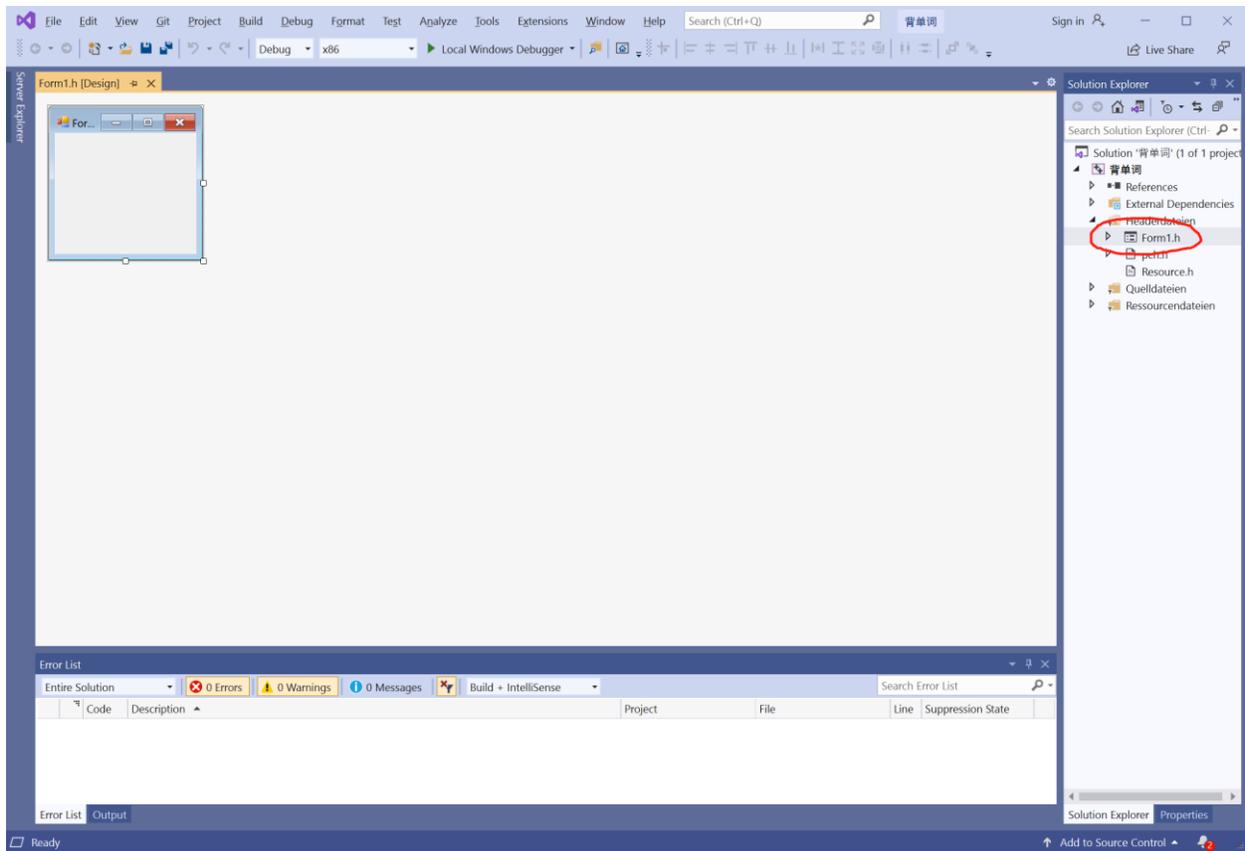
Console App
Run code in a Windows terminal. Prints "Hello World" by default.
C++ Windows Console

CMake Project
Build modern, cross-platform C++ apps that don't depend on .sln or .vcxproj files.
C++ Windows Linux Console

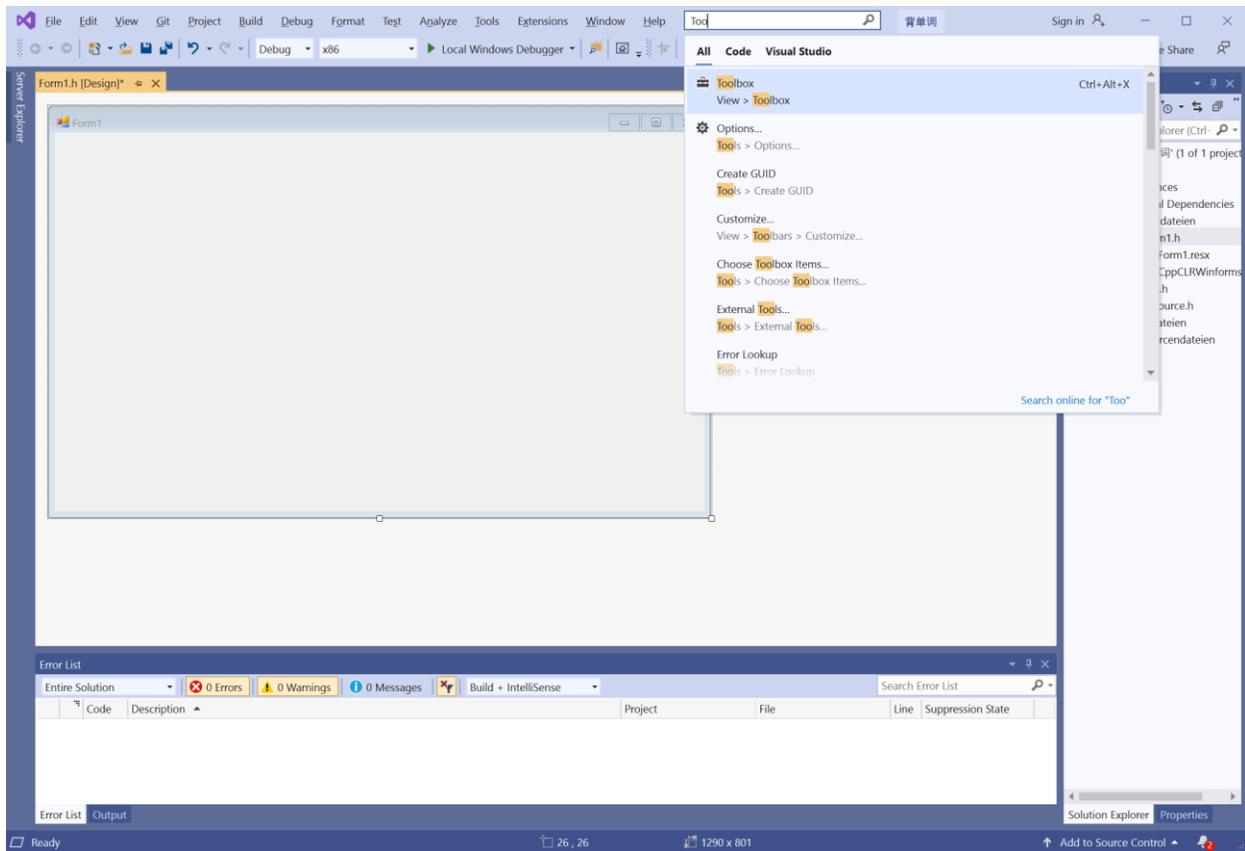
Windows Desktop Wizard
Create your own Windows app using a wizard.
C++ Windows Desktop Console Library

[Next](#)

那么我就来给大家展示一下如何写一个简单的背单词辅助程序：

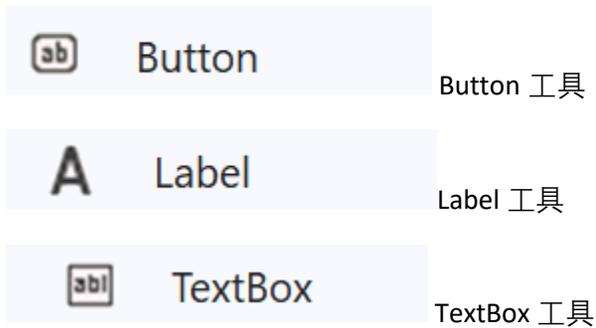


进入界面后点击 Form1.h, 可以看到一个什么都没有的小窗口, 你可以先将窗口拉成你想要的大小。



在搜索框搜索 Toolbox，打开工具箱。

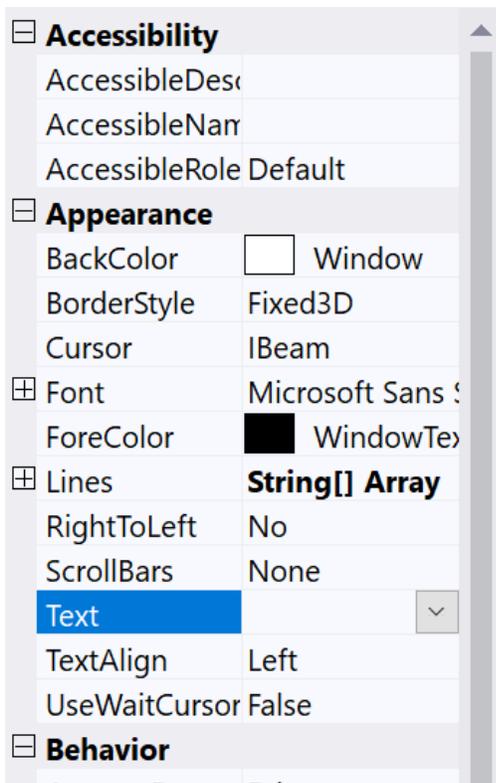
工具箱中有着各种各样实用的工具，但是由于篇幅限制，显然是不可能全部介绍，这次就先介绍三个最基本也是最常用的工具：



我们想要写一个背单词的程序，那么输入单词就是一个必须的事情，找到 `TextBox`，单击之后即可在 `form` 上放置：



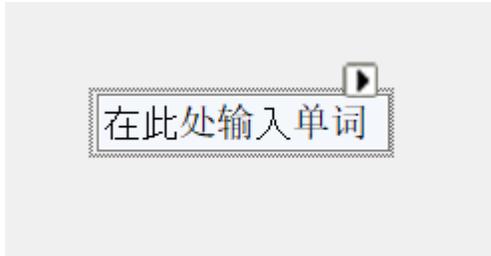
右键放置好的文本框，点击属性：



可以看到这个文本框有着各种各样的属性，通过修改这些属性，我们可以办到非常多的事情：

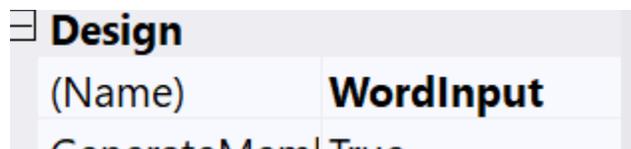
ScrollBars	None
Text	在此处输入单词
TextAlign	Left

比如通过修改 Text 可以改变他的初始输出，



，通过修改颜色可以改变文本框的初始颜色等等。

然而最重要的是，



我们需要修改他的名字使得代码可读性更高，并且可以防止在写代码的时候将不同的元件搞混。

那么当用户在文本框中填入单词后，程序如何将单词框中的单词录入并进行后续的计算呢？用户应该如何交互呢？此时按钮就派上了用处。

同样的单击按钮并在合适的地方放置。



双击按钮，我们即可快速进入编辑按钮的交互：

```

    }
    #pragma endregion
    private: System::Void Input_Click(System::Object^ sender, System::EventArgs^ e) {
    }
    };
}

```

直接在该区域编写代码即可。（我将已经按钮名称修改为 Input）。

同时可以看到，我们创建的组件名称也已经归类进 form 类中：

```

    }
    private: System::Windows::Forms::TextBox^ WordInput;
    protected:
    private: System::Windows::Forms::Button^ Input;
}

```

这时可能有小伙伴要问了，既然如此，那我们在双击按钮快速创建函数后，再修改按钮元件的名字，会产生 bug 吗？

答案是，会的。

所以这也是一个需要注意的地方，并且除此之外，还有各种各样容易出错的地方。

比如我放置了一个按钮组件，双击快速创建函数后又不想要了，于是将该组件从 form 上删除，再回到代码区将该组件函数删除，这样会引发错误码？

答案是，会的。

因为快速创建函数并不单单只是在该地方创建一个函数而已，他同时也在类中声明了该函数以及一些相关联的东西，需要删除的代码有很多，漏删一行或多删一个标点都会引发错误。

因此，我强烈建议：

先将窗口设计完后在进行代码。

那我们回到窗口的设计。

在刚刚我们修改了 TextBox 的属性使得他有一个初始的输出，然而 TextBox 中的显示是可以被用户直接修改的，那么有没有什么办法既能够输出提示，又可以防止用户修改呢？

那就到了 Label 元件出场的时候啦！

Label 的用法和 textbox 基本一样，可以用来输出提示信息，但是用户不能直接修改 Label 的显示。

那么我们的表格现在看起来就像这样：

The image shows a screenshot of a Windows application window titled "Form1". The window has a standard Windows title bar with minimize, maximize, and close buttons. Inside the window, there are two text input fields. The first field is preceded by the label "在此处输入单词" (Enter word here). The second field is preceded by the label "在此处输入注释" (Enter note here). Below these two fields is a button with the text "录入单词" (Enter word).

显然作为一个背单词辅助器，光是录入单词是不够的，我们还需要能够查找单词，能够给自己做一些小测验，比如随机输出单词，用户输入注释再与已单词库内注释比较看是否正确，还要能够修改已输入单词的注释等等...

那么我设计的界面大概是这样，大家可以随意发挥，包括可以改颜色使得界面更加美观，我这里就将需要的部分编排上就好了。

在设计完表格之后，我们终于要开始代码环节啦！芜湖！

由于我们使用的窗口程序实际上是通过访问.NET 来实现的，因此我们需要使用托管类型。我们需要定义两个一维托管数组（或者一个二维数组）以及一个用于表示单词数量的 size 变量：

```
public: array<String^>^ word = gcnew array<String^>(3000); //要你命3000
public: array<String^>^ meaning = gcnew array<String^>(3000);
public: unsigned size = 0;
```

位置在 `public ref class Form1 : public System::Windows::Forms::Form` 中。（关于托管数组在本篇文章中不做描述，要讲的话有非常多，这次只是教大家怎么简单的写一个窗口程序。）

```

namespace CppCLRWinformsProjekt {
    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;
    /// <summary>
    /// Zusammenfassung für Form1
    /// </summary>
    public ref class Form1 : public System::Windows::Forms::Form
    {
    public: array<String^>^ word = gcnew array<String^>(3000); //要你命3000
    public: array<String^>^ meaning = gcnew array<String^>(3000);
    public: unsigned size = 0;
    public:
        Form1(void)
        {
            InitializeComponent();
            //
            //TODO: Konstruktorcode hier hinzufügen.
            //
        }

    protected:

```

在定义完数组之后，我们就可以开始实现录入单词的功能了，然而录入单词并非只是将单词与注释存入数组这么简单，我们还需要考虑一些额外的情况：

1. 用户没有同时输入单词或注释。
2. 用户输入的单词已经存在于数组之中

代码如下：

```

private: System::Void Input_Click(System::Object^ sender, System::EventArgs^ e) {
    if (WordInput->Text == String::Empty || meaningInput->Text ==
String::Empty)
    {
        MessageBox::Show(L"请同时输入单词和注释! ");
        return;
    }
    else
    {
        for (unsigned i = 0; i < size; i++)
        {
            if(word[i] == WordInput->Text)
            {
                meaning[i] = meaningInput->Text;
                MessageBox::Show(L"该单词已存在，以更新单词注释! ");
                return;
            }
        }
        word[size] = WordInput->Text;
        meaning[size] = meaningInput->Text;
        size++;
    }
}

```

```
}
```

MessageBox 就是像这样的输出提示:



通过以上代码, 我们可以看到, 我们可以通过 `ObjectName ->Text` 的方式来获取 `Object` 的输出字符, 同理, 我们也可以通过这种方式来输出字符达成与用户交互的目的, 接下来演示下一个单词按钮交互的代码:



```
private: System::Void NextWord_Click(System::Object^ sender, System::EventArgs^ e) {  
    unsigned number = Convert::ToInt16(No->Text);  
    number++;  
    No->Text = Convert::ToString(number);  
    if (number > size)  
    {  
        MessageBox::Show(L"编号超出存放单词数量");  
    }  
    else  
    {  
        WordOutput->Text = word[number - 1];  
        MeaningOutput->Text = meaning[number - 1];  
    }  
}
```

在该代码中, 我们首先读取了单词编号文本框中的数字, 然后输出了下一个单词。然而这串代码并不完善, 我们有一些需要考虑的地方:

1. 尽量在开启程序时就给单词编号文本框一个初始值, 这样就可以直接使用下一个单词功能。
2. 如果用户在单词编号文本框输入了非数字类型字符串, 需要提示用户重新输入。

那么如何给文本框一个初始值呢？除了在属性栏修改，还有另一种方法，双击表格的任意位置（除了有元件的位置），则会快速创建一个：

```
private: System::Void Form1_Load
```

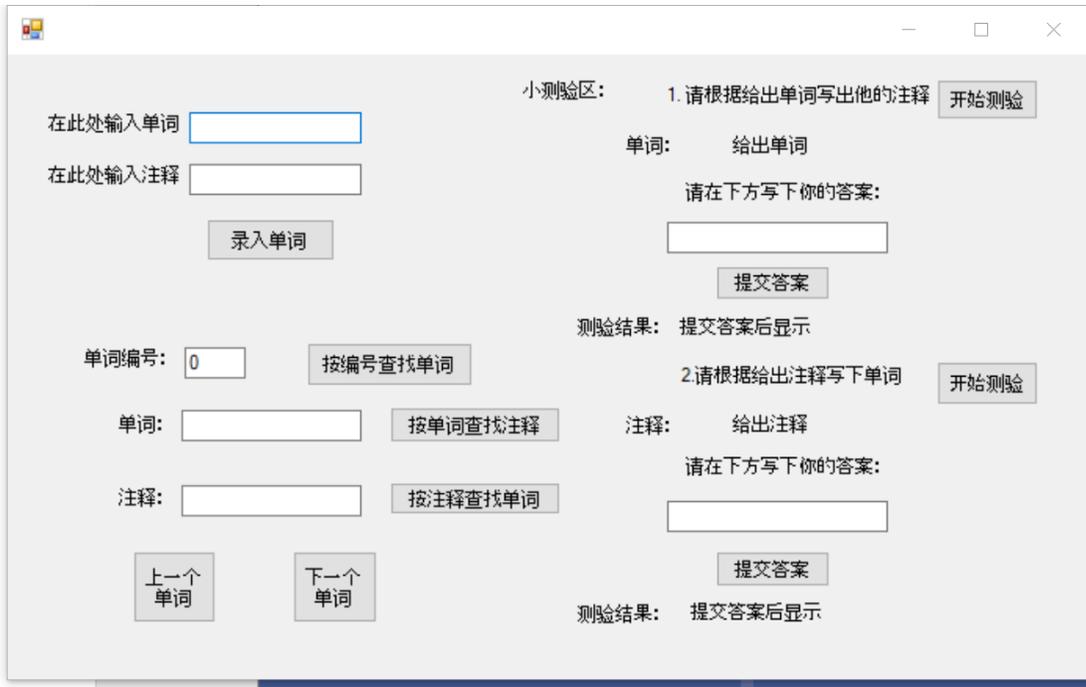
函数，该函数在程序运行时自动执行，可以看作是“构造函数”，因此我们可以在该函数中给定一些初始值：

```
private: System::Void Form1_Load(System::Object^ sender, System::EventArgs^ e) {  
    No->Text = L"0";  
}
```

而对于第二个问题，我们可以采用抛出异常的方式处理，修改完后：

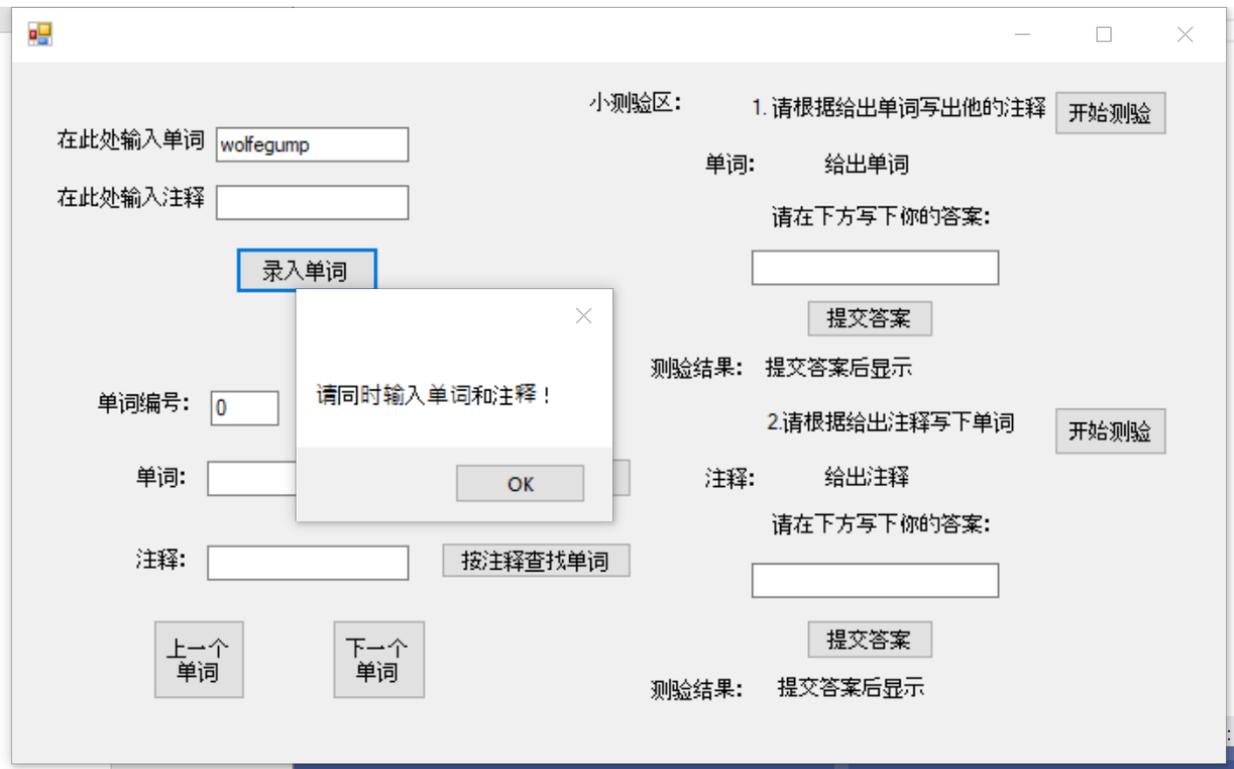
```
private: System::Void NextWord_Click(System::Object^ sender, System::EventArgs^ e) {  
    try {  
        unsigned number = Convert::ToInt16(No->Text);  
        number++;  
        No->Text = Convert::ToString(number);  
        if (number > size)  
        {  
            MessageBox::Show(L"编号超出存放单词数量");  
        }  
        else  
        {  
            WordOutput->Text = word[number - 1];  
            MeaningOutput->Text = meaning[number - 1];  
        }  
    }  
    catch (...)  
    {  
        MessageBox::Show(L"请在编号处输入数字! ");  
    }  
}
```

我们可以使用 debug 来测试一下我们的程序：



可以看到，单词编号栏已经有了一个初始值 0。

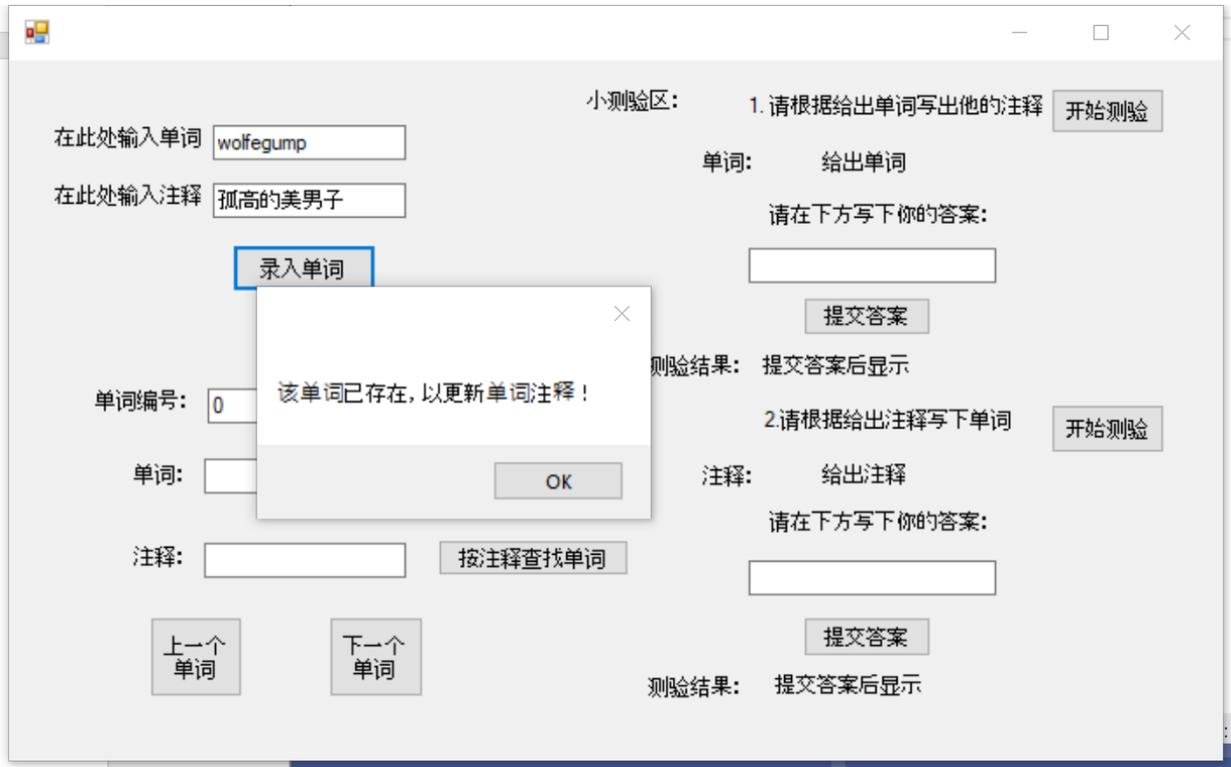
当我们只输入单词不输入注释时：



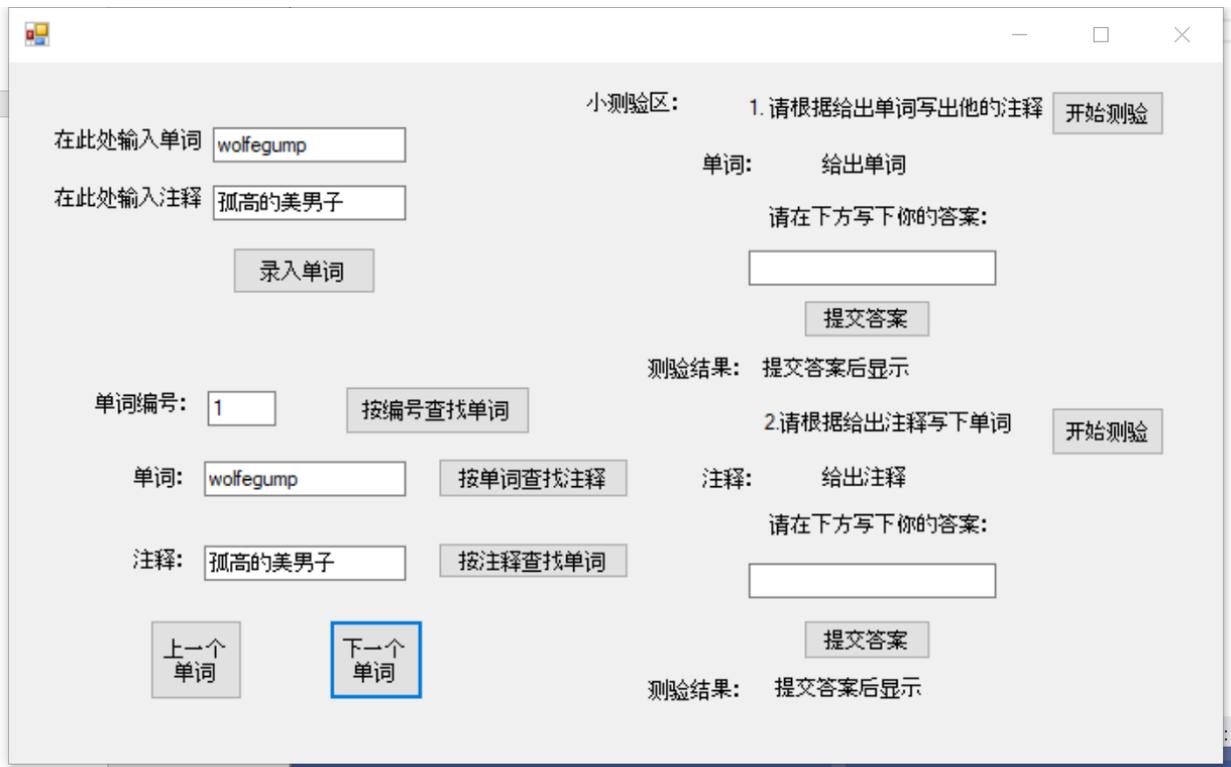
当我们同时输入单词和注释时：

似乎缺少了一些交互，我们可以先记下来，回头新增一个 label 或是用 messageBox 的形式输出一些录入成功类的交互。

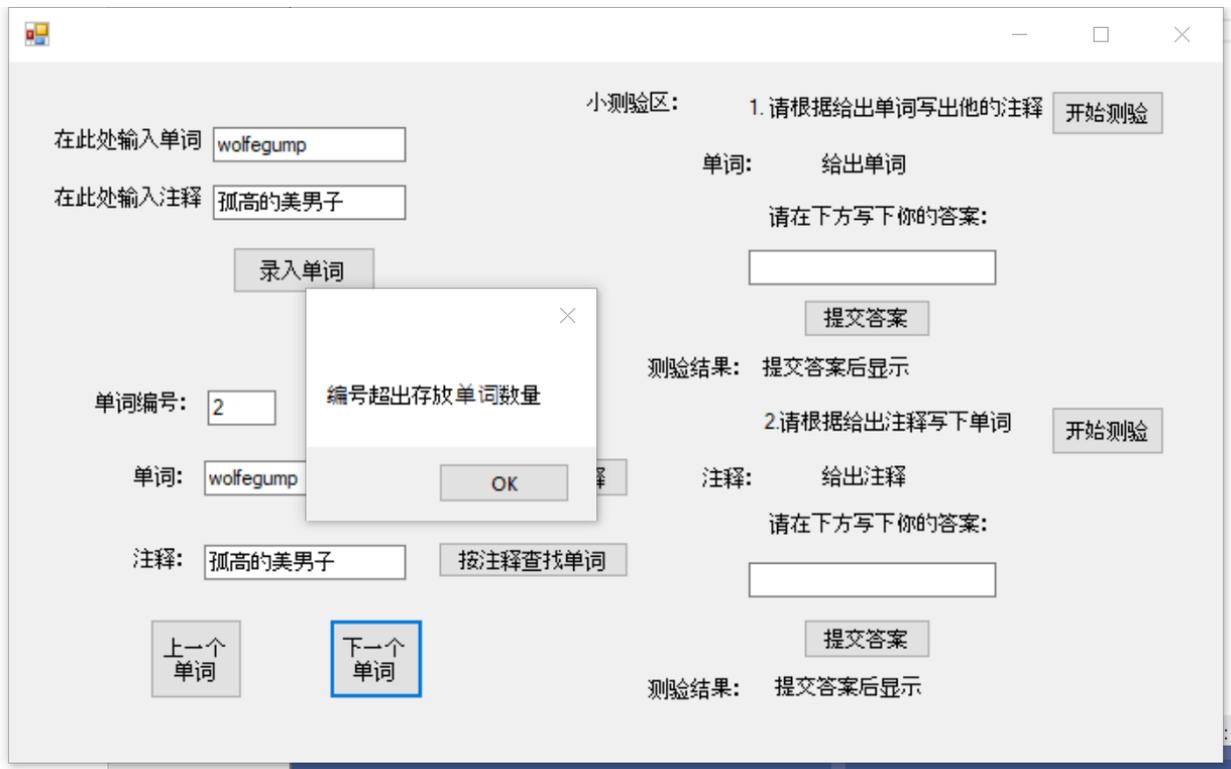
当我们第二次输入同样的单词时：



接下来测试下一个单词功能:

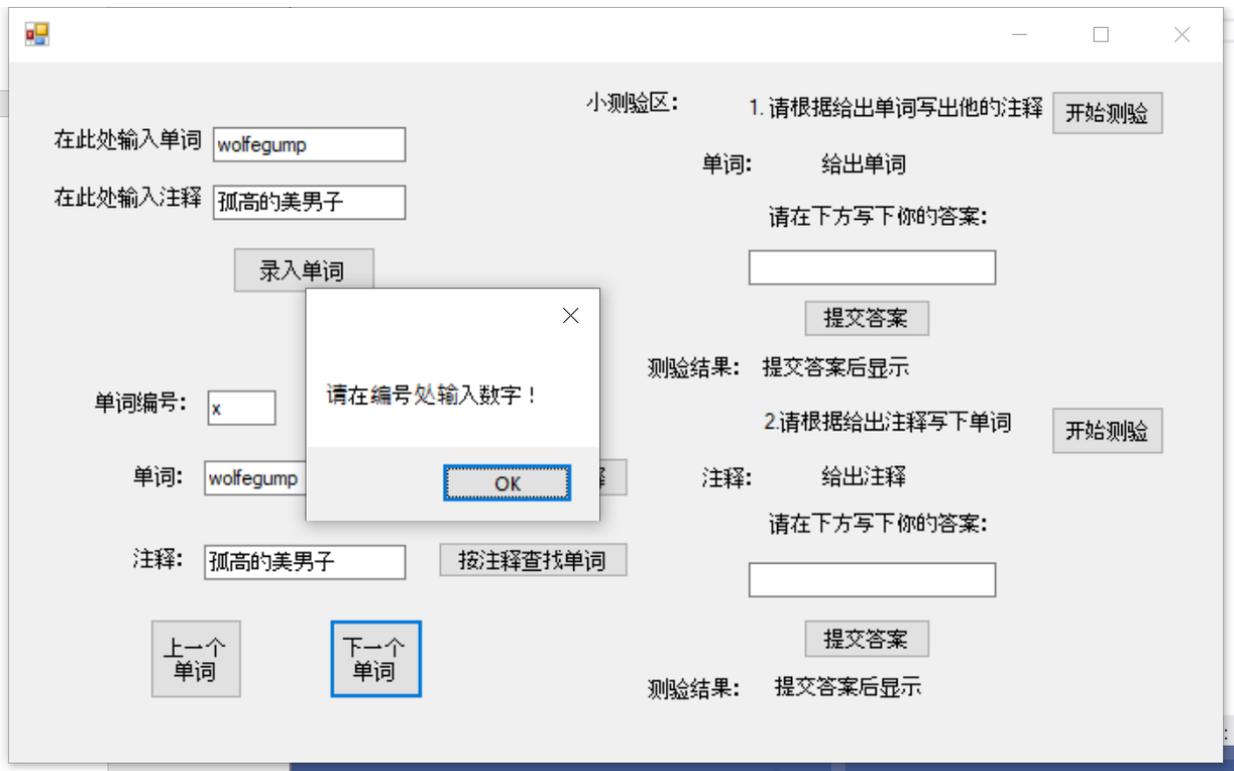


可以看到成功的输出了我们录入的单词, 再次点击:



由于我们是修改了单词注释，而没有输入新单词，因此 size 依旧为 1。

在单词编号处输入其他字符：



完美!

上一个单词按钮以及按编号查找按钮的实现方法也是大同小异，在此不多做描述。

按单词查找注释:

```
private: System::Void searchByWord_Click(System::Object^ sender, System::EventArgs^ e) {
    for (unsigned i = 0; i < size; i++)
    {
        if (word[i] == WordOutput->Text)
        {
            No->Text = Convert::ToString(i);
            MeaningOutput->Text = meaning[i];
            return;
        }
    }
    MessageBox::Show(L"不存在该单词! ");
}
```

遍历数组，如果不存在单词则输出不存在单词。注释查找的方法也一样。

测验 1 开始的代码:

```
private: System::Void Quiz1Start_Click(System::Object^ sender, System::EventArgs^ e) {
```

```

    if (size == 0)
    {
        MessageBox::Show(L"请先录入单词! ");
        return;
    }
    srand(time(0));
    Quiz1Number = rand() % size;
    Quiz1Question->Text = word[Quiz1Number];
}

```

在此处我们需要定义一个新的全局变量 Quiz1Number, 以便提交答案后进行对比:

Quiz1 提交答案的代码:

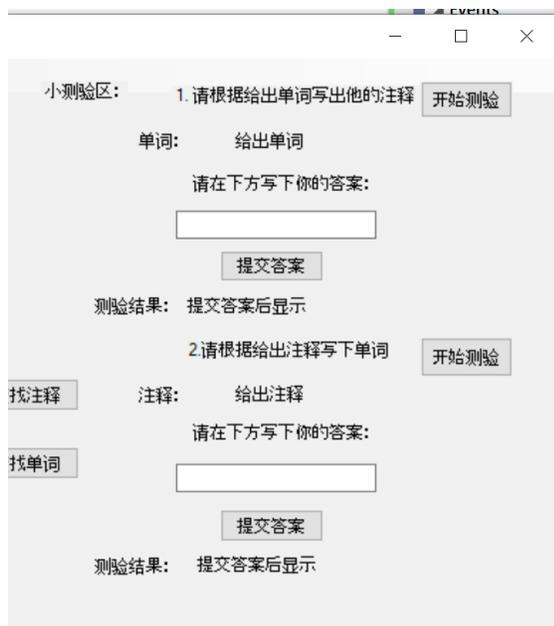
```

private: System::Void Quiz1Check_Click(System::Object^ sender, System::EventArgs^ e) {
    if (Quiz1Question->Text == L"给出单词")
    {
        MessageBox::Show(L"请先点击开始测验! ");
        return;
    }
    if (Quiz1Input->Text == meaning[Quiz1Number])
    {
        Quiz1Result->Text = L"答对啦! 你真棒! ";
    }
    else
    {
        Quiz1Result->Text = L"很遗憾, 你答错了, " + word[Quiz1Number] + L" 的意思是: "
+ meaning[Quiz1Number];
    }
}

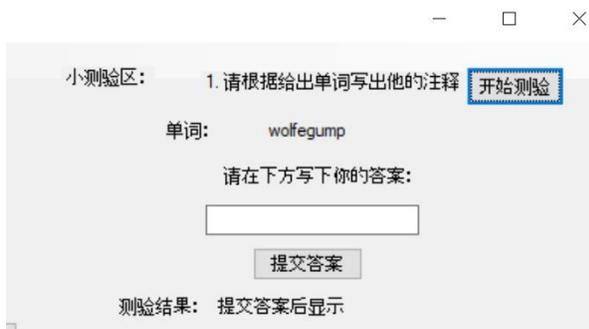
```

测验 2 的代码大同小异, 在此不多做描述。

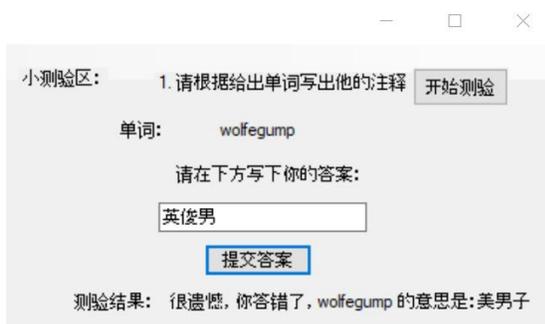
以下是测试:



点击开始测验：



输入单词后输入错误答案并点击提交答案：



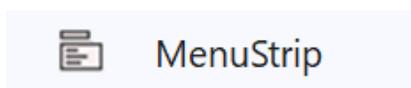
再次点击开始测验并输入正确答案：

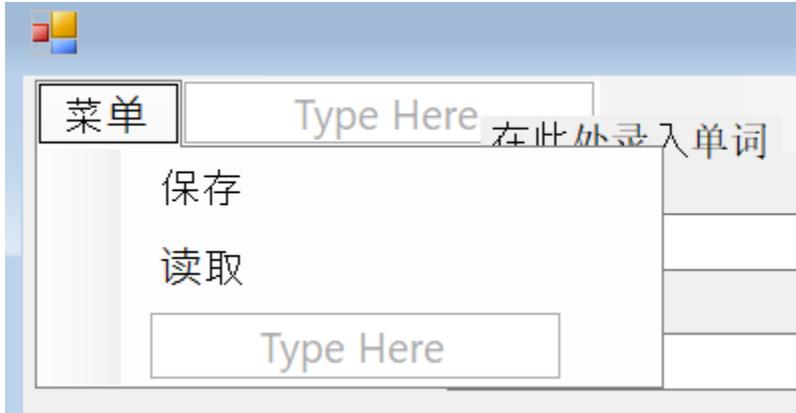


那么在实现了全部功能之后，出现了一个问题。虽然这个程序很方便，但是每次打开需要重新输入单词，有没有什么办法能够将单词保存下来，下次直接开始测试呢？

答案是，有的，接下来介绍 save 以及 open 功能的实现。

首先在 ToolBox 中找到 MenuStrip 工具来创建一个菜单：



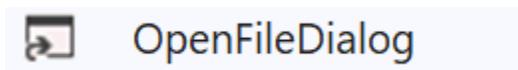


可以放置在左上角，同时输入保存和读取来为 save 和 open 功能做准备。

接下来找到 SaveFileDialog 工具：



以及 OpenFileDialog 工具：



拖入设计界面的最下方的菜单栏中：



然后我们需要在代码界面添加一句代码:

```
namespace CppCLRWinformsProjekt {  
    using namespace System;  
    using namespace System::ComponentModel;  
    using namespace System::Collections;  
    using namespace System::Windows::Forms;  
    using namespace System::Data;  
    using namespace System::Drawing;  
    using namespace System::IO;
```

以及一个新的成员变量:

```
public: unsigned quizNumber;  
public: bool save = false;
```

该变量的作用是: 如果你以打开已存在文件的方式打开程序, 那么保存时就不会创建新的保存文件。

接下来双击菜单中的保存按钮快速创建保存函数:



作为一个背单词程序，我们需要保存的内容有：

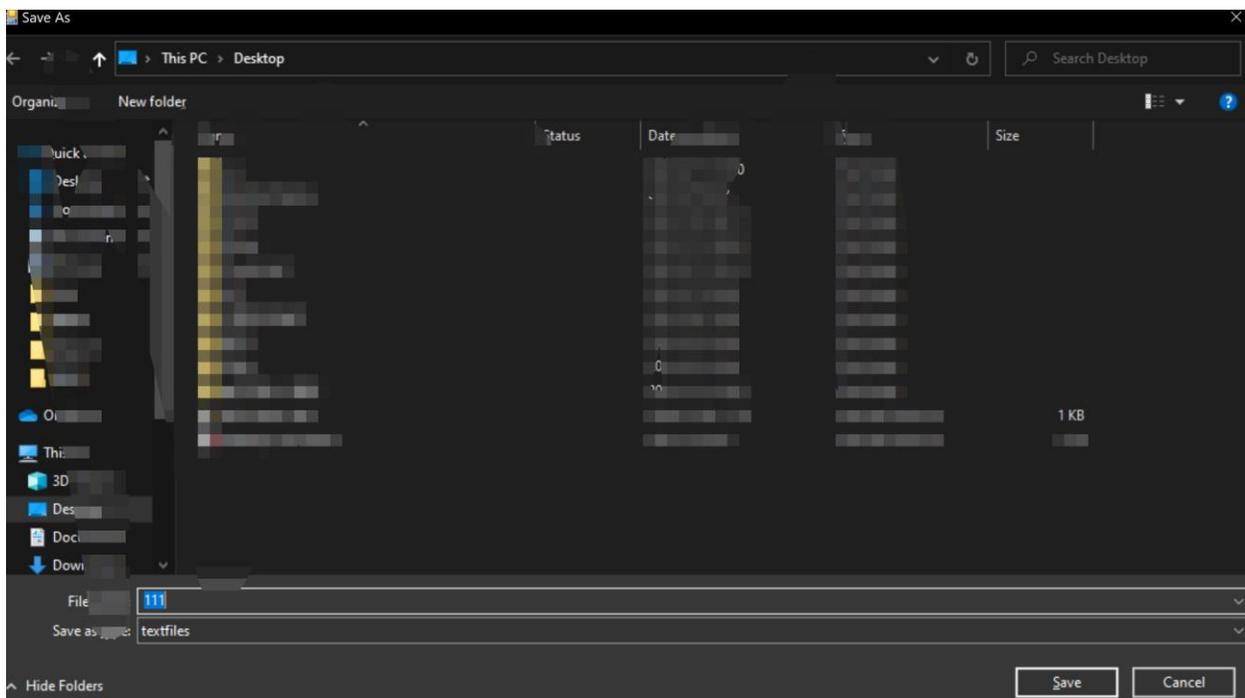
单词的总量，单词以及单词的注释：

```
private: System::Void 保存ToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {
    if (save == true)
    {
        FileStream^ f = gcnew FileStream(saveFileDialog1->FileName,
        FileMode::Create);
        StreamWriter^ r = gcnew StreamWriter(f);
        r->WriteLine(Convert::ToString(size)); //先将单词总数记下
        for (unsigned i = 0; i < size; i++)
        {
            r->WriteLine(word[i]); //使用循环一次存入单词
            r->WriteLine(meaning[i]);
        }
        r->Close();
    }
    else
    {
        saveFileDialog1->Filter = "textfiles | *.txt";
        saveFileDialog1->FileName = "";
        if (saveFileDialog1->ShowDialog() ==
        System::Windows::Forms::DialogResult::OK)
        {
            FileStream^ f = gcnew FileStream(saveFileDialog1->FileName,
            FileMode::Create);
            StreamWriter^ r = gcnew StreamWriter(f);
            r->WriteLine(Convert::ToString(size));
            for (unsigned i = 0; i < size; i++)
            {
                r->WriteLine(word[i]);
                r->WriteLine(meaning[i]);
            }
            r->Close();
        }
    }
}
```

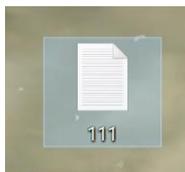
读取时，我们也只需要先读取单词总数 size，然后循环 size 次依次将单词和注释存入数组中：

```
private: System::Void 读取ToolStripMenuItem_Click(System::Object^ sender,
System::EventArgs^ e) {
    openFileDialog1->FileName = "";
    openFileDialog1->Filter = "text files|*.txt";
    openFileDialog1->DefaultExt = "*.txt";
    if (openFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)
    {
        FileStream^ f = gcnew FileStream(openFileDialog1->FileName,
        FileMode::OpenOrCreate);
        StreamReader^ r = gcnew StreamReader(f);
        size = Convert::ToInt16(r->ReadLine());
        for (unsigned i = 0; i < size; i++)
        {
            word[i] = r->ReadLine();
            meaning[i] = r->ReadLine();
        }
        r->Close();
        save = true;
    }
}
```

好的，那让我们来试一试保存和读取功能吧！



点击保存后跳出保存界面，输入文件名后点击保存。

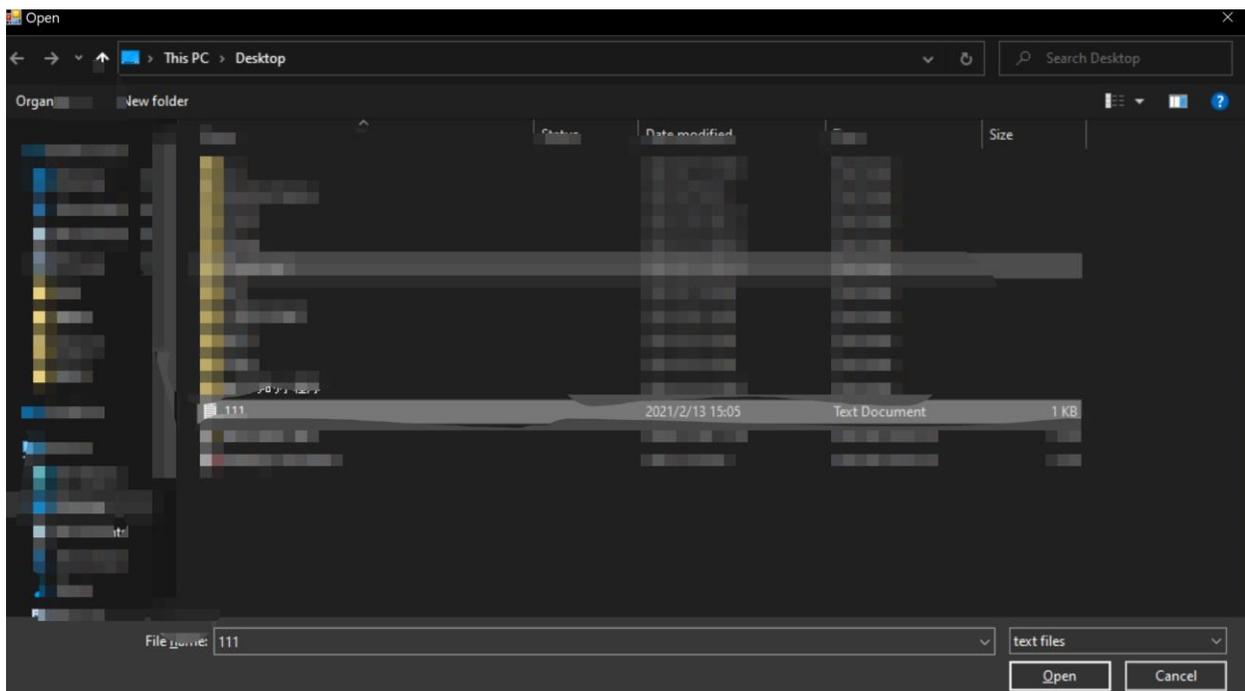


桌面上出现了保存的 txt 文件，

打开后：



接下来尝试读取，重新 debug 程序，点击读取按钮：



选择要读取的文件，并打开



可以看到原先的单词已被成功读取，再存入新的单词：happy，并保存。

此次保存并没有跳出新建文件界面。

再次点击 111.txt:



我们新输入的单词也已经保存进去了！

另外还有许许多多的其他功能，比如 save as, print 等等，大都大同小异，大家可以自己慢慢尝试！

感谢各位能花时间阅读到这里！希望看完这篇文章后，你也可以写出自己想要的程序！

该程序的 exe 以及源代码已上传 google drive，有兴趣的可以自行下载使用（稍微做了一些修改）：

<https://drive.google.com/file/d/1zFhhJApY17QQARVg0iD24ddmax2HOSgF/view?usp=sharing>

再次感谢各位的阅读！