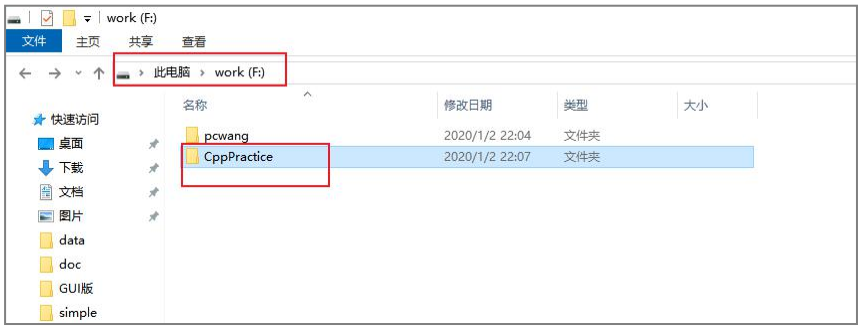
目录

C++实习指导手册（2019）

1 程序代码框架

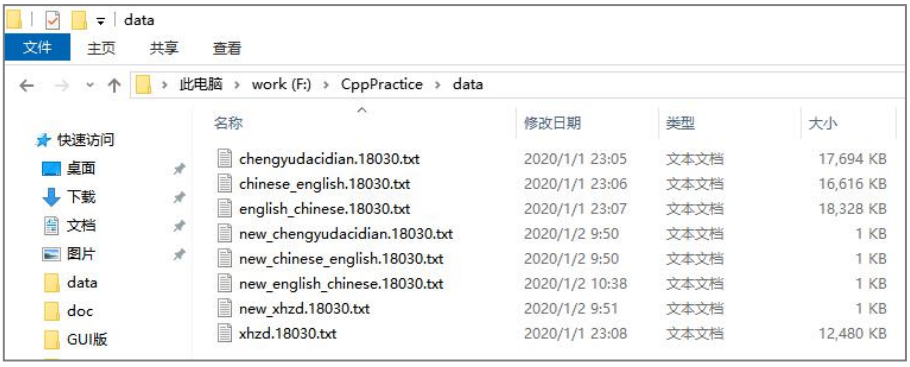
1.1 规划一下目录结构

新建一个目录 CppPractice，我们此次实习的所有相关文档、数据和程序都将放到这个目录里面。如下图，我是在 F 盘根目录下建的 CppProtice。



在 CppPractice 目录下再建两个子目录 data 和 src，词典数据也就是词库将放在 data 目录，程序源码也就是 VS 建立的工程将放在 src 目录中。

将此次实习用到的词典数据文件拷贝到 data 目录。如下图：



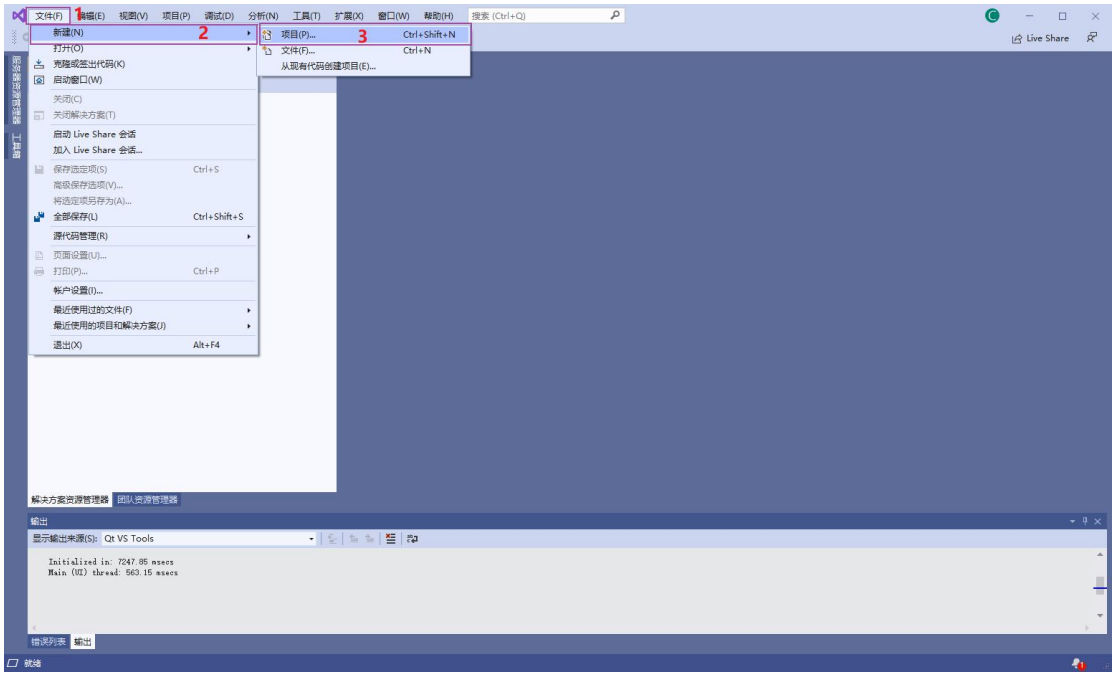
文件名与对应的词库表如下：

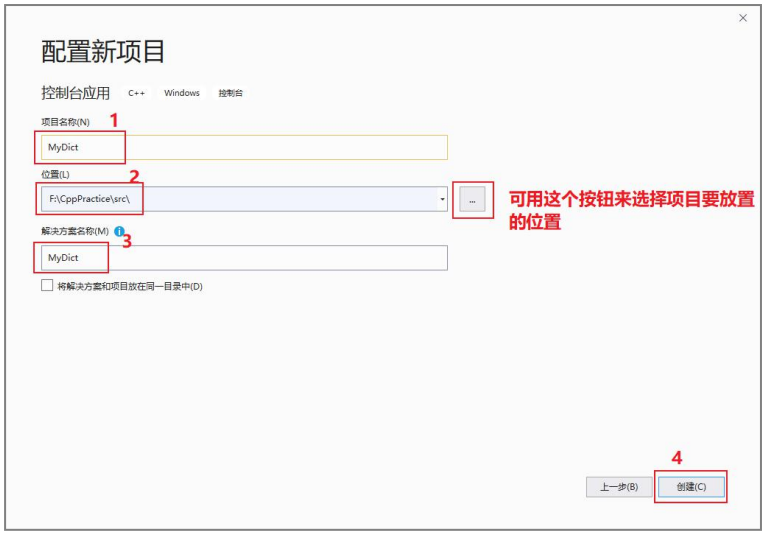
词库	文件名	生词本文件名
成语词典	chengyudacidian.18030.txt	new_chengyudacidian.18030.txt
英汉词典	english_chinese.18030.txt	new_english_chinese.18030.txt
汉英词典	chinese_english.18030.txt	new_chinese_english.18030.txt
新华字典	xhzd.18030.txt	new_xhzd.18030.txt

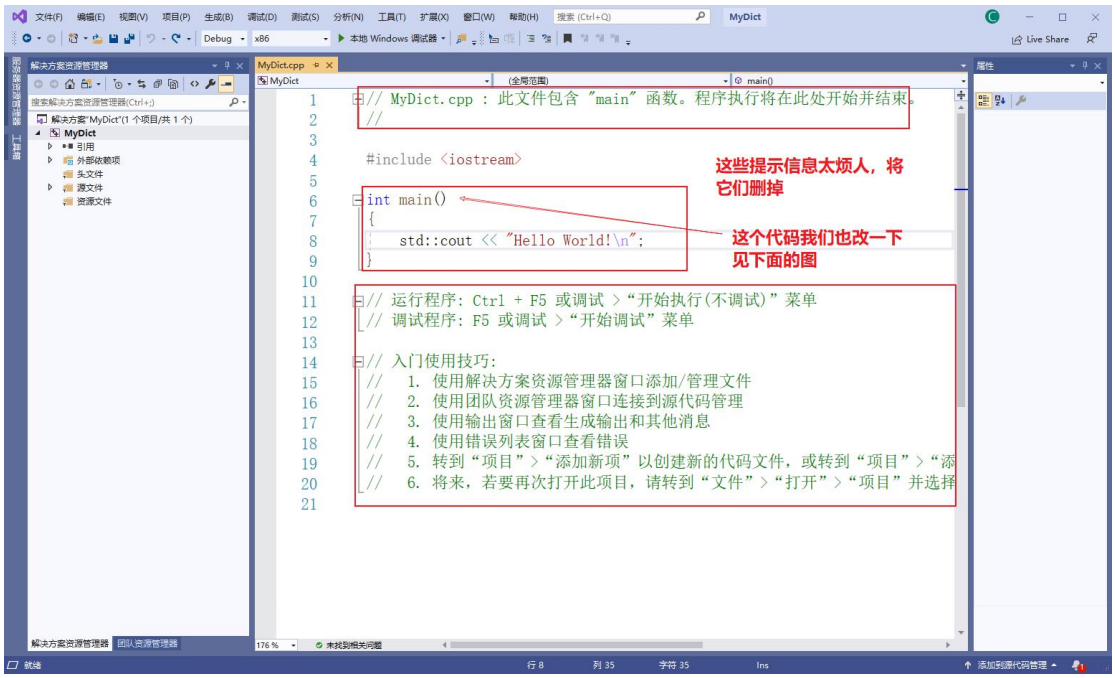
文件名中的 18030 表示此文件采用的字符集为 gb18030，前缀 new_ 表示生词本文件。

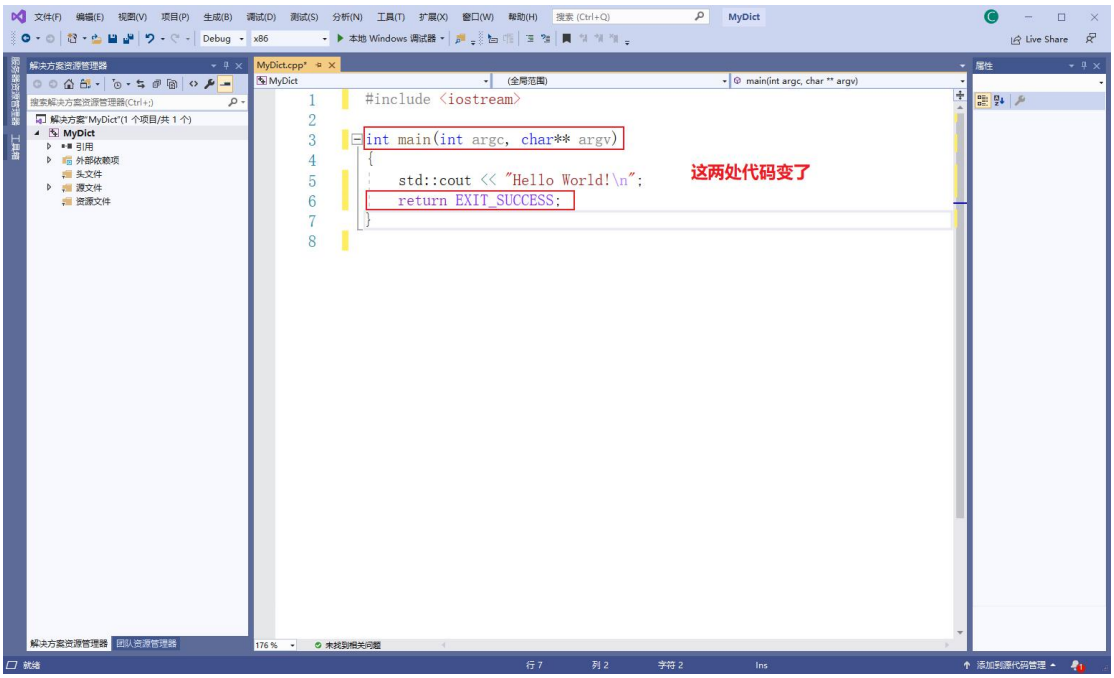
1.2 初始项目

新建 C++控制台项目：

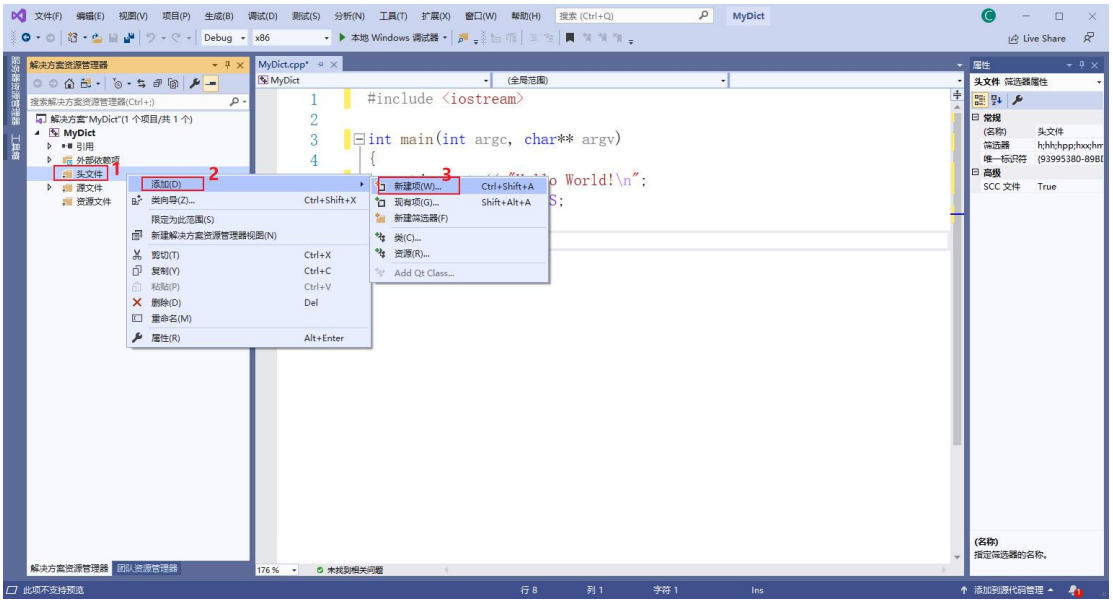


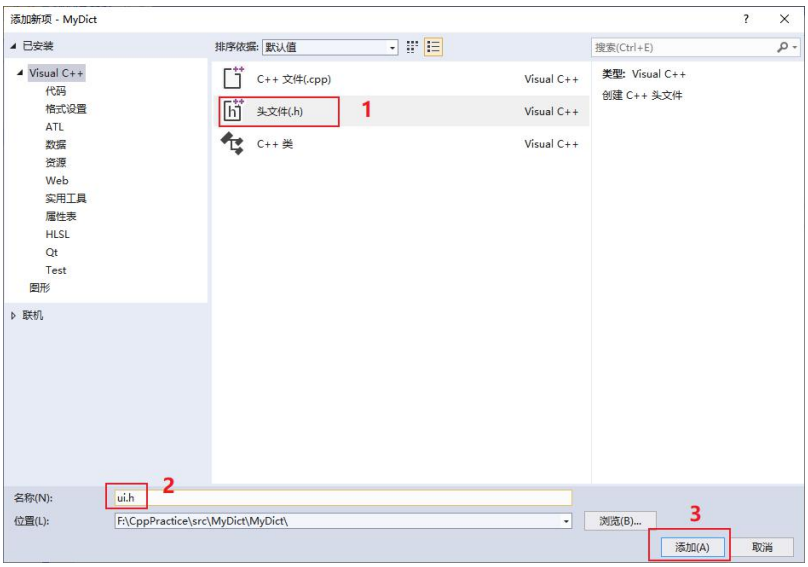

配置新项目
控制台应用
C++
Windows
控制台
项目名称(N)
1
MyDict
位置(L)
2
F:\CppPractice\src\
...
可用这个按钮来选择项目要放置的位置
解决方案名称(M)
3
MyDict
将解决方案和项目放在同一目录中(D)
4
上一步(B)
创建(C)

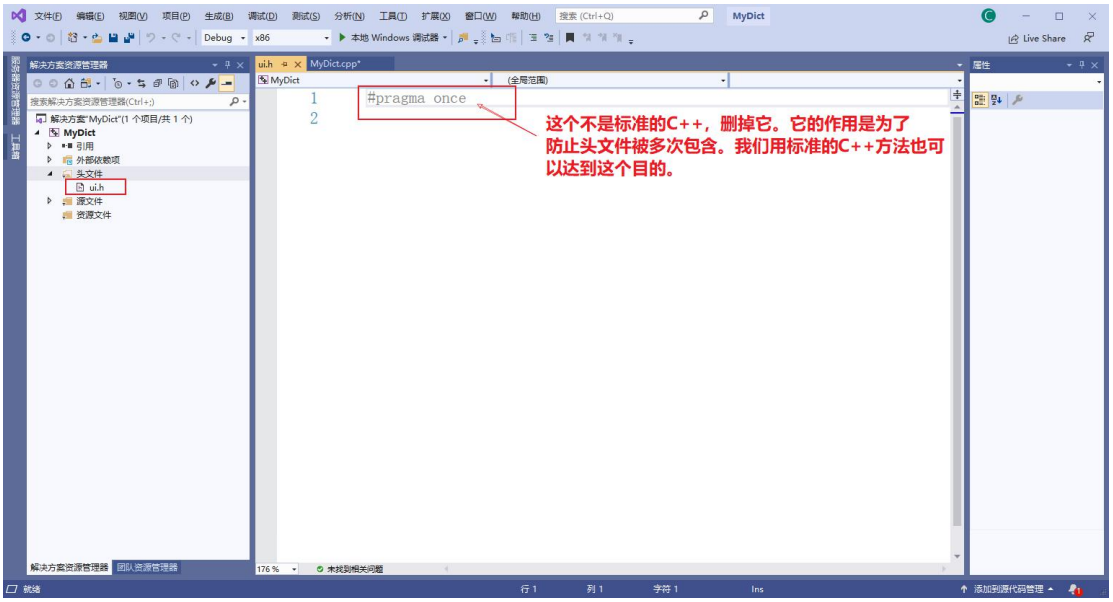

MyDict.cpp
// MyDict.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
//
#include <iostream>
这些提示信息太烦人，将它们删掉
int main()
{
std::cout << "Hello World!\n";
}
这个代码我们也改一下见下面的图
// 运行程序: Ctrl + F5 或调试 > "开始执行(不调试)" 菜单
// 调试程序: F5 或调试 > "开始调试" 菜单
// 入门使用技巧:
// 1. 使用解决方案资源管理器窗口添加/管理文件
// 2. 使用团队资源管理器窗口连接到源代码管理
// 3. 使用输出窗口查看生成输出和其他消息
// 4. 使用错误列表窗口查看错误
// 5. 转到 "项目" > "添加新项" 以创建新的代码文件，或转到 "项目" > "添
// 6. 将来，若要再次打开此项目，请转到 "文件" > "打开" > "项目" 并选择


#include <iostream>
int main(int argc, char** argv)
{
std::cout << "Hello World!\n";
return EXIT_SUCCESS;
}
这两处代码变了

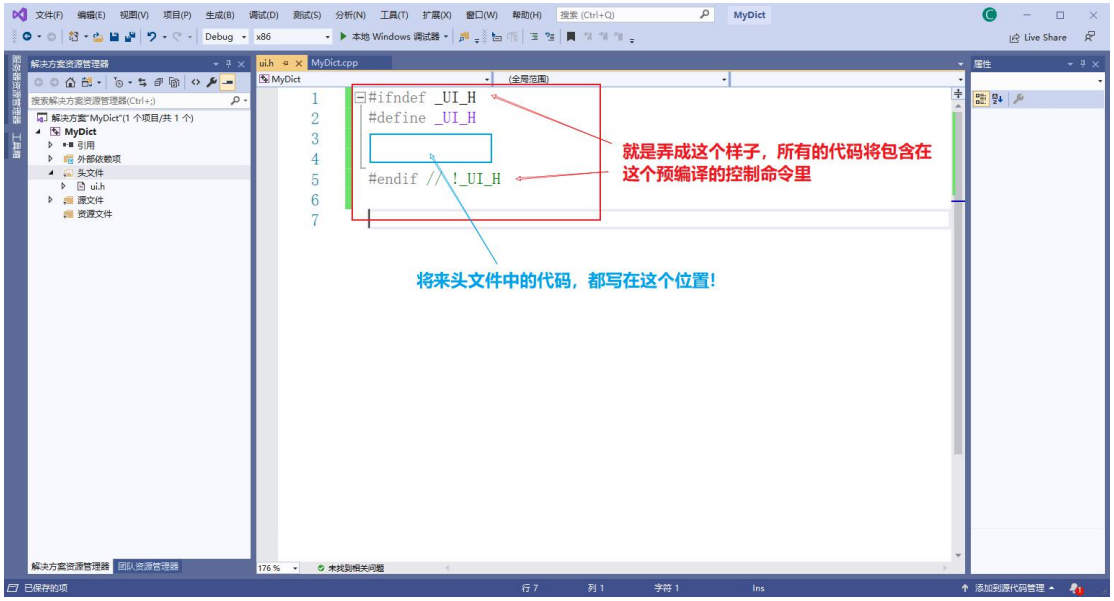
添加文件 ui.h:


1
2
3
添加(D)
新建项(W)... Ctrl+Shift+A
现有项(G)... Shift+Alt+A
新建筛选器(F)
类(C)...
资源(R)...
Add Qt Class...

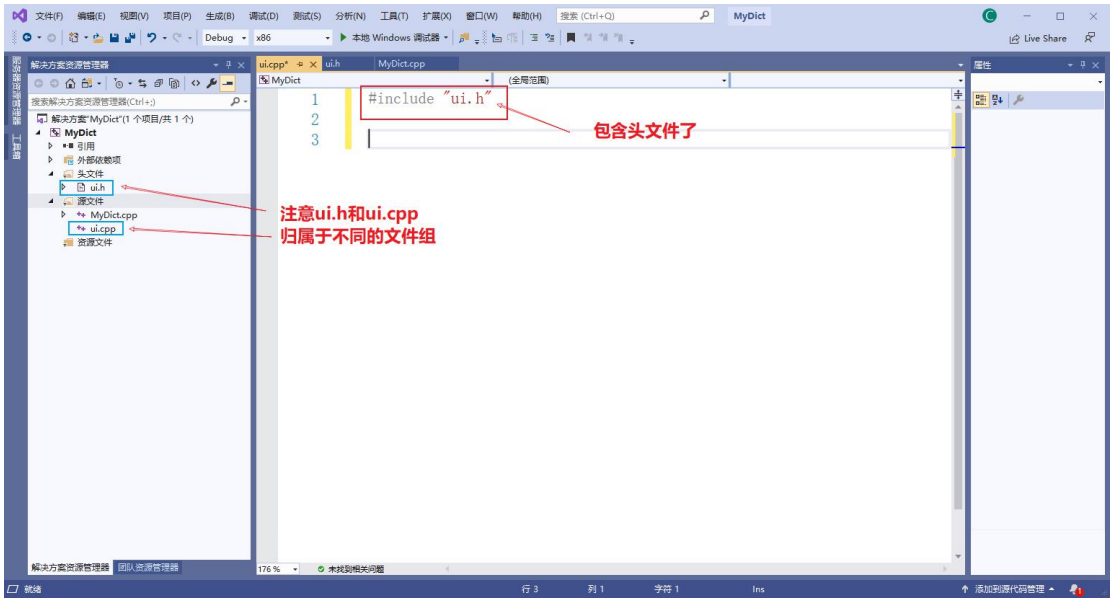

添加新项 - MyDict
已安装
Visual C++
代码
格式设置
ATL
数据
资源
Web
实用工具
属性表
HLSL
Qt
Test
图形
联机
排序依据: 默认值
C++ 文件(.cpp)
头文件(.h)
C++ 类
Visual C++
1
类型: Visual C++
创建 C++ 头文件
名称(N): ui.h
2
位置(L): F:\CppPractice\src\MyDict\MyDict\
浏览(B)...
3
添加(A)
取消


#pragma once
这个不是标准的C++，删掉它。它的作用是为了防止头文件被多次包含。我们用标准的C++方法也可以达到这个目的。

将 ui.h 改成这样：



同样的方法，添加 ui.cpp:



在 ui.h 中添加代码：

```
#ifndef _UI_H
#define _UI_H

#include <string>

#define PAUSE() system("pause")
#define CLEAR() system("cls")

using namespace std;

void init_console();

#endif // !_UI_H
```

PAUSE宏，程序暂停

CLEAR宏，清除控制台上的内容

这两个宏只对windows系统起作用，如果是其他系统，重新换一个宏定义哦。

在 ui.cpp 中添加代码：

```
#include "ui.h"

void init_console() {
    system("mode con cols=64 lines=30");
    system("chcp 936");
    system("color f0");
    system("cls");
}
```

设置控制台行列数

设置内码页为936才能更好地支持中文

设置控制台的前景色和背景色f0即背景白色，前景黑色

清屏，其实也可以用刚才的CLEAR()宏

修改 MyDict.cpp：

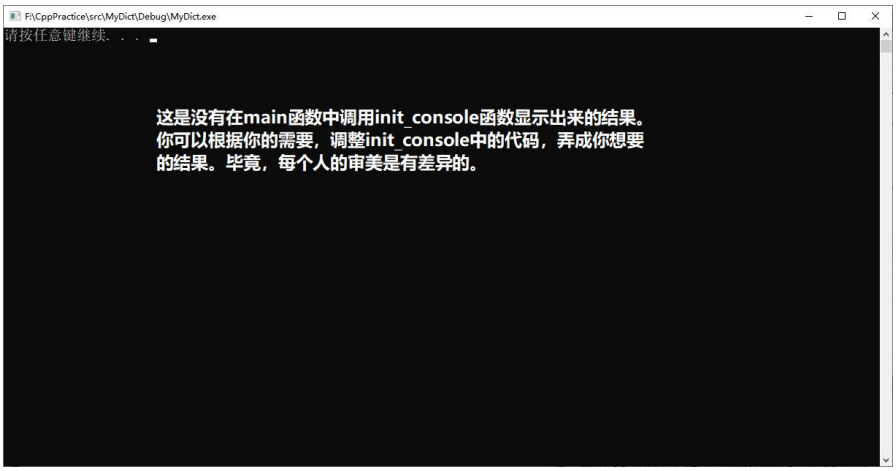
```
#include <iostream>
#include "ui.h"

int main(int argc, char** argv)
{
    init_console();
    PAUSE();
    return EXIT_SUCCESS;
}
```

初始化好控制台的显示状态

让程序停一下，否则我们在调试的时候程序会一闪而过。看不到啥东西。

编译运行它：



1.3 主界面实现

为 ui.h 添加代码：

```
1  #ifndef _UI_H
2  #define _UI_H
3
4  #include <string>
5
6  #define PAUSE() system("pause")
7  #define CLEAR() system("cls")
8
9  using namespace std;
10
11 void init_console();
12 void main_ui();
13
14 #endif // !_UI_H
15
```

新增函数原型

修改 ui.cpp:

```
1  #include "ui.h"
2  #include <iostream>
3
4  void init_console() {
5      system("mode con cols=64 lines=30");
6      system("chcp 936");
7      system("color f0");
8      system("cls");
9  }
10
```

引入头文件

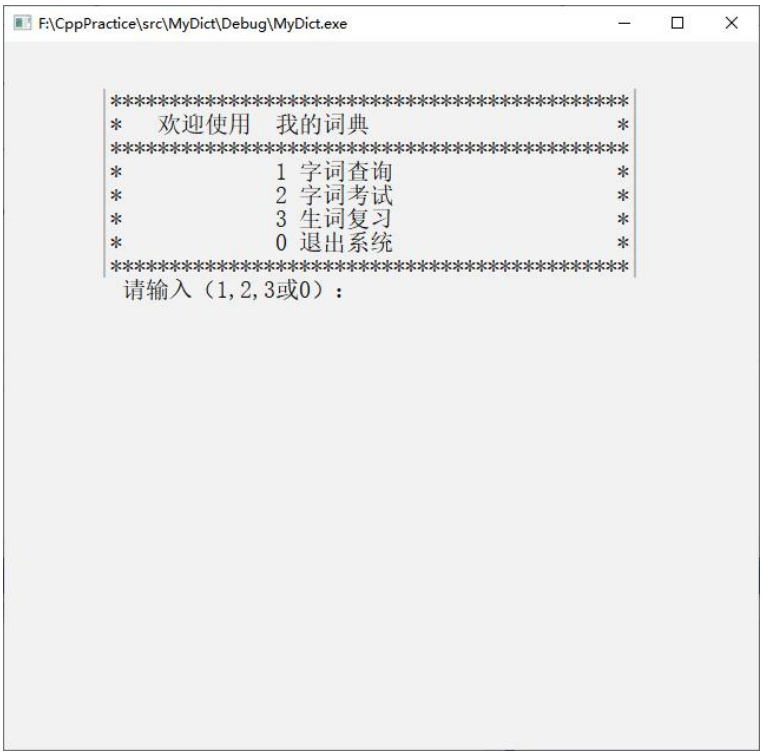
```
11
12 const string main_ui_prompt =
13 "\n\n"
14 "\t|*********************************************|\n"
15 "\t|*   欢迎使用  我的词典                        *|\n"
16 "\t|*********************************************|\n"
17 "\t|*            1 字词查询                     *|\n"
18 "\t|*            2 字词考试                     *|\n"
19 "\t|*            3 生词复习                     *|\n"
20 "\t|*            0 退出系统                     *|\n"
21 "\t|*********************************************|\n"
22 "\t  请输入（1,2,3或0）：  "
23 ;
```

```
24
25  void main_ui() {
26      CLEAR();
27      cout << main_ui_prompt;
28      string op;
29      std::getline(cin, op);
30      if (op == "1") {
31      }
32      else if (op == "2") {
33      }
34      else if (op == "3") {
35      }
36      else if (op == "0") {
37          return;
38      }
39
40      main_ui();
41  }
```

修改 MyDict.cpp:

```
1   #include <iostream>
2   #include "ui.h"
3
4   int main(int argc, char** argv)
5   {
6       init_console();
7       main_ui();   <-- 添加了这一句
8       PAUSE();
9       return EXIT_SUCCESS;
10  }
11
```

编译运行：



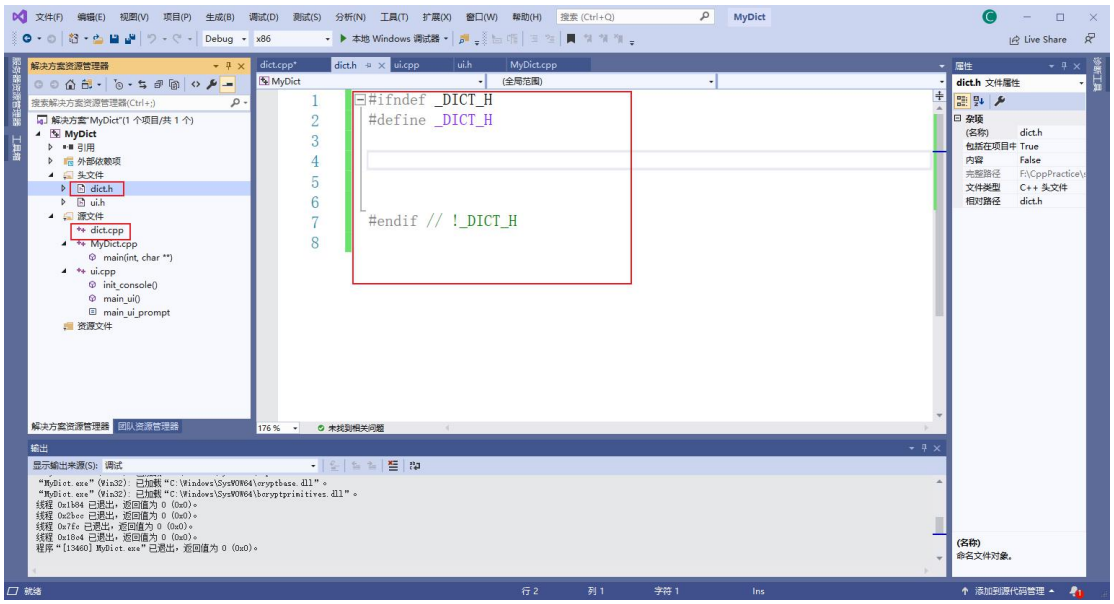
试用一下，除了输入 0 之外，其他的所有输入都好像没有反应一样，其实这是符合我们现在的程序逻辑的。不急，我们慢慢来改。如果输入 0 然后回车，这个界面会退出来，程序终止。

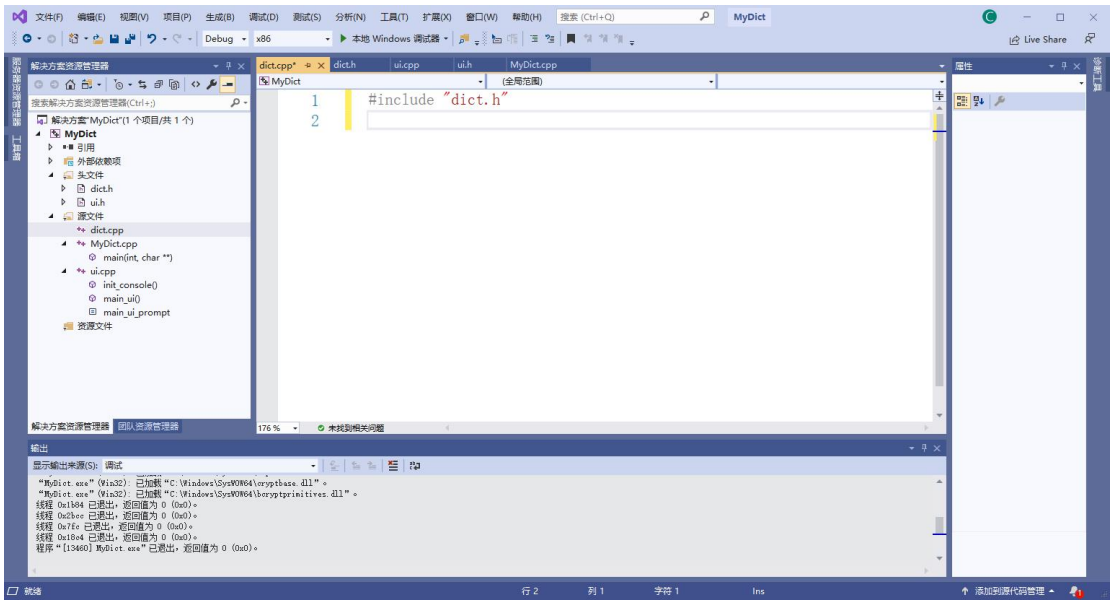
这个地方的代码要注意输入时采用的是字符串，不是整数，不是字符。

1.4 跳转到查字典界面

所有关于查字典的代码我们都放置在 dict.h 和 dict.cpp 中。

为项目添加两个文件 dict.h 和 dict.cpp，步骤和前面的 ui.h、ui.cpp 一样：





修改 dict.cpp 的代码：

```
#include "dict.h"
#include <string>
#include <iostream>
#include "ui.h"

using namespace std;
const string dict_ui_prompt =
"\n\n"
"\t|*********************************************|\n"
"\t|*   请选择（字）词典                        *|\n"
"\t|*********************************************|\n"
"\t|*              1 英汉词典                   *|\n"
"\t|*              2 汉英词典                   *|\n"
"\t|*              3 成语词典                   *|\n"
"\t|*              4 新华字典                   *|\n"
"\t|*              0 返回上级                   *|\n"
"\t|*********************************************|\n"
"\t  请输入（1,2,3,4或0）：  "
;
```

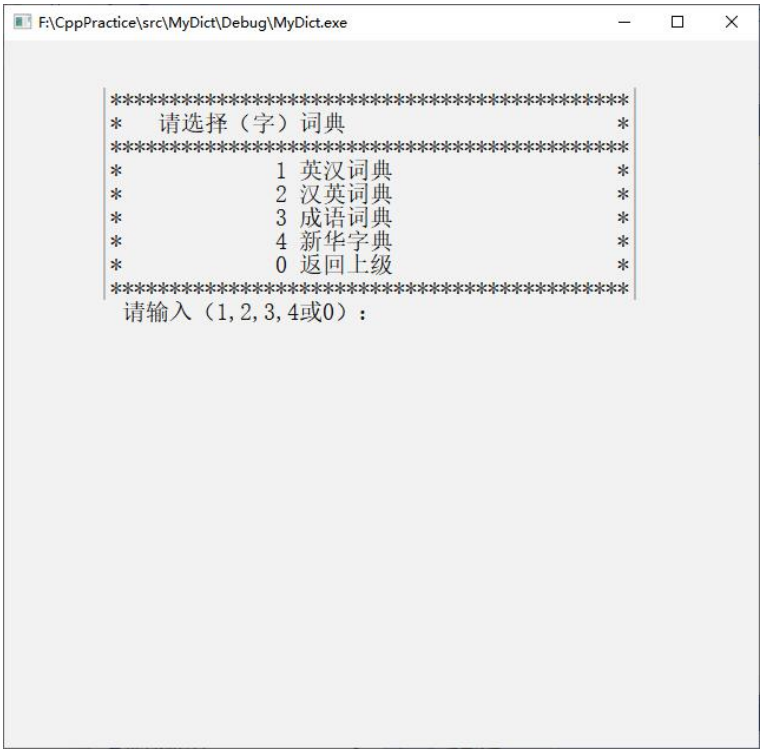
```
21  void dict_ui() {
22      CLEAR();
23      cout << dict_ui_prompt;
24      string op;
25      std::getline(cin, op);
26      if (op == "1") {
27
28      }
29      else if (op == "2") {
30
31      }
32      else if (op == "3") {
33
34
35      }
36      else if (op == "4") {
37
38      }
39      else if (op == "0") {
40          return;
41      }
42      dict_ui();
43  }
```

修改 ui.cpp 中的 main_ui 函数：

```
void main_ui() {
    CLEAR();
    cout << main_ui_prompt;
    string op;
    std::getline(cin, op);
    if (op == "1") {
        dict_ui();
    }
    else if (op == "2") {
    }
    else if (op == "3") {
    }
    else if (op == "0") {
        return;
    }

    main_ui();
}
```

编译运行，当主界面出现后，输入 1，将跳转到查字典界面，要求你选择何种词典：



输入 0 后，又可以回到主界面。程序现在已经可以在这两个界面之间自如地跳转了。

1.5 跳转到英汉词典（英译汉）界面

在 dict.h 中添加函数原型：

```
void dict_ui_english_to_chinese();
```

在 dict.cpp 中添加函数实现：

```
void dict_ui_english_to_chinese() {
    CLEAR();
    const string prompt =
        "\n\n"
        "\t|*********************************************|\n"
        "\t|*       英汉词典                            *|\n"
        "\t|*********************************************|\n"
        "\t  请输入英文："
        ;
    cout << prompt;
    string word = "";
    std::getline(cin, word);
    if (word == "") {
        dict_ui_english_to_chinese();
        return;
    }
    cout << "\t  查询结果是：" << endl;
    if (word == "book") {
        cout << "\t    书、书籍；订购" << endl;
    }


    cout << "\t|*********************************************|\n";
    cout << "\t  请选择(1 加入生词本,0 返回上级,其他 继续):";
    string op;
    std::getline(cin, op);
    if (op == "1") {
    }
    else if (op == "0") {
        return;
    }
    dict_ui_english_to_chinese();
}
```

仔细阅读一下程序，理解它的运行逻辑。

现在已经实现一个只能够查询一个单词 book 的中文含义的词典了。

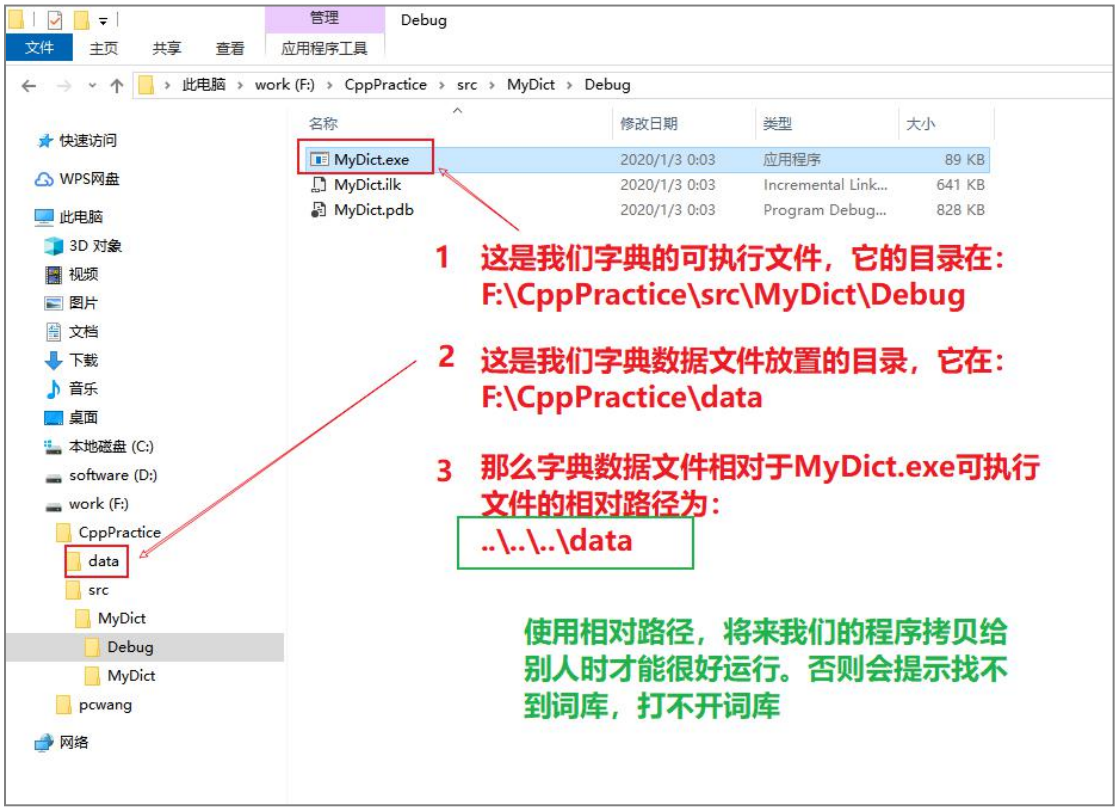
我们紧接着来弄一个复杂的。

2 字典读取及查询

要实现字典的读取，首先得了解一下字典格式。我专门写了一个文档介绍了我们的字典格式，请阅读一下它再到这里来。等你 10 分钟.....

.....

哦，首先定位一下字典所在的目录。打开 MyDict 所在目录看看：



2.1 字典读取函数

下面为 dict.h 一次增加的代码有点多哦：

```
#ifndef _DICT_H
#define _DICT_H

#include <string>
#include <vector>

// 下面的宏定义了字（词）库所对应的文件
#define DICT_PATH "..\\..\\..\\data\\"
#define DICT_ENGLISH_CHINESE "english_chinese.18030.txt"
#define DICT_CHINESE_ENGLISH "chinese_english.18030.txt"
#define DICT_CHENGYUDACIDIAN "chengyudacidian.18030.txt"
```

```
#define DICT_XINHUAZIDIAN     "xhzd.18030.txt"

using namespace std;

// 字典结构体
typedef struct {
  string filename;
  bool loaded ;
  vector<string> words;                 // 一个词典很多单词，是一个数组
  vector< vector<string> > meanings; // 单词的释义，和单词是一一对应的
}Dictionary ;

// 我们总共有 4 个字词典：英汉、汉英、成语词典、新华字典
// 同时我们也有四个生词本
// extern 表示这是在别处定义的一个变量，我们只是想提前用一用它
extern Dictionary dictionaries[4];
extern Dictionary unfamiliarDictionaries[4];
// 下面的这些宏表示字典序号 1 英汉、2 汉英、3 成语词典、4 新华字典
// 定义宏是为了容易记忆且不易出错
#define ENGLISH_CHINESE 0
#define CHINESE_ENGLISH 1
#define CHENGYUDACIDIAN 2
#define XINGHUAZIDIAN    3

//装载词库
void dict_load(Dictionary* dict, const string& filename);
//查字典
vector<string> dict_lookup(Dictionary* dict, const string& word, bool& find);
vector<string> dict_lookup(Dictionary* dict, const string& word);
//输出单词/词语 释义
void meaning_display(const vector<string>& meaning);
void dict_ui();
void dict_ui_english_to_chinese();

#endif // !_DICT_H
```

简易

在 dict.cpp 最开始的地方添加两个变量：

```
#include "dict.h"
#include <string>
#include <iostream>
#include "ui.h"

using namespace std;

Dictionary dictionaries[4];
Dictionary unfamiliarDictionaries[4];
```

dict.cpp 中实现 dict_load：

```
void dict_load(Dictionary* dict, const string& filename) {
    // 如果装载过了，就不要再读文件了
    if (dict->loaded) {
        return;
    }


    fstream in(filename, ios::in);
    if (!in) {
        cerr << "打不开词典文件:" << filename << endl;
        PAUSE();
        exit(1);
    }

    dict->filename = filename;
    //得到单词/短语个数
    int wordscount;
    string line;
    std::getline(in, line);
    //读取出来的是字符串，要将他转成整数才行，atoi 函数
    wordscount = atoi(line.c_str());
    //有这么多单词，当然要都这么多次了
    for (int i = 0; i < wordscount; i++) {
        //读取单词
        std::getline(in, line);
        //单词保存到内存中的字典数据结构里
        dict->words.push_back(line);
        vector<string> meaning;
        // 一个单词的解释有很多行，读取行数
        std::getline(in, line);
        int linecount = atoi(line.c_str());
        // 多行释义的读取
        for (int j = 0; j < linecount; j++) {
            std::getline(in, line);
            // 每读一行，都将它存到单词释义的数组里
            meaning.push_back(line);
        }
        // 每一个单词有一个释义，释义是多行的，放进内存里面去
        dict->meanings.push_back(meaning);
    }

    in.close();
    // 装载成功了，更新一下装载成功的标志
```

```
        dict->loaded = true;
    }
```

2.2 字典查询

dict.cpp 中添加剂啊 dict_lookup 函数，它有两个原型：

```
    vector<string> dict_lookup(Dictionary* dict, const string& word, bool& find) {
        vector<string> result;
        result.push_back("运气不好，没有找到！");

        //这个查询就是按顺序来，一个个比较，
        //如果匹配成功，就表示查到了这个单词，返回相应的解释
        for (unsigned int i = 0; i < dict->words.size(); i++) {
            if (word == dict->words[i]) {
                result = dict->meanings[i];
                find = true;
                return result;
            }
        }
        // 如果没查到，程序就到这里来了
        find = false;
        return result;
    }

    // 下面就是一个偷懒的写法，调用了上面的那个带三个参数的同名函数
    vector<string> dict_lookup(Dictionary* dict, const string& word) {
        bool find = false;
        return dict_lookup(dict, word, find);
    }
```

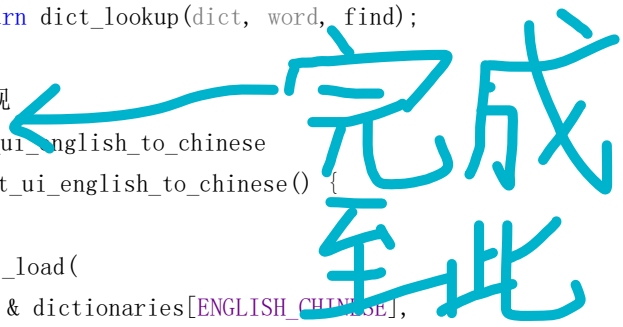
2.3 英译汉功能实现

还要改一下 dict_ui_english_to_chinese

```
    void dict_ui_english_to_chinese() {

        dict_load(
            & dictionaries[ENGLISH_CHINESE],
            DICT_PATH DICT_ENGLISH_CHINESE
        );

        CLEAR();
        const string prompt =
            "\n\n"
            "\t|*****************************************|\n"
            "\t|*       英汉词典                        *|\n"
            "\t|*****************************************|\n"
            "\t  请输入英文："
```

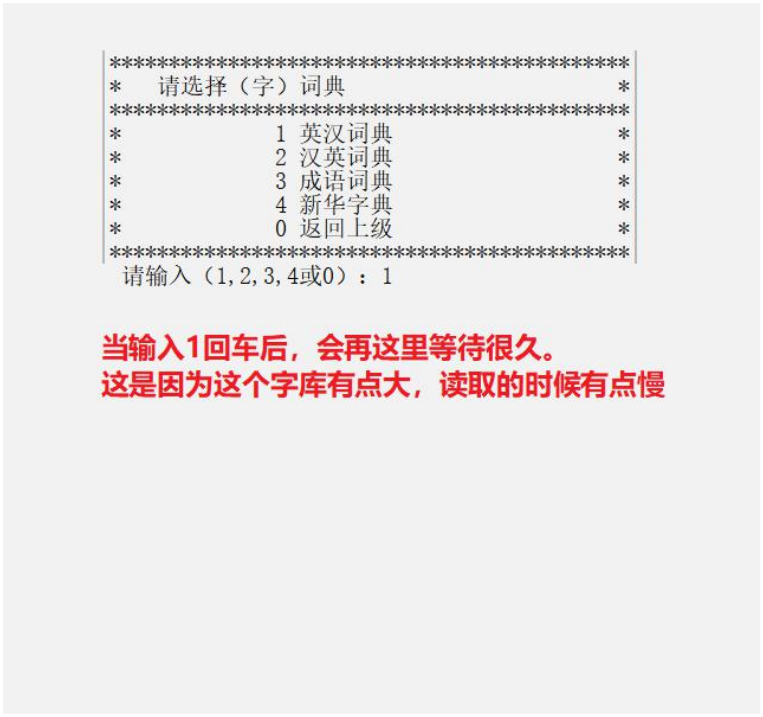


```
        ;
    cout << prompt;
    string word = "";
    std::getline(cin, word);
    if (word == "") {
        dict_ui_english_to_chinese();
        return;
    }

    bool find = false;
    vector<string> meaning = dict_lookup(
        &dictionaries[ENGLISH_CHINESE], word, find);
    if (find) {
        cout << "\t  查询结果是：" << endl;
        meaning_display(meaning);
        cout << "\t|*****************************************|\n";
        cout << "\t  请选择(1 加入生词本,0 返回上级,其他 继续):";
    }
    else {
        meaning_display(meaning);
        cout << "\t|*****************************************|\n";
        cout << "\t  请选择(0 返回上级,其他 继续):";
    }


    string op;
    std::getline(cin, op);
    if (op == "1") {
        //只有查询得到的单词/词语才会需要加入生词本
        if (find) {
            //加入生词本
        }
    }
    else if (op == "0") {
        return;
    }
    dict_ui_english_to_chinese();
}
```

运行一下，你会发现在输入 1 后好像没有反应，要等待很久。那就先等一会呗。



等吧，直到出现：

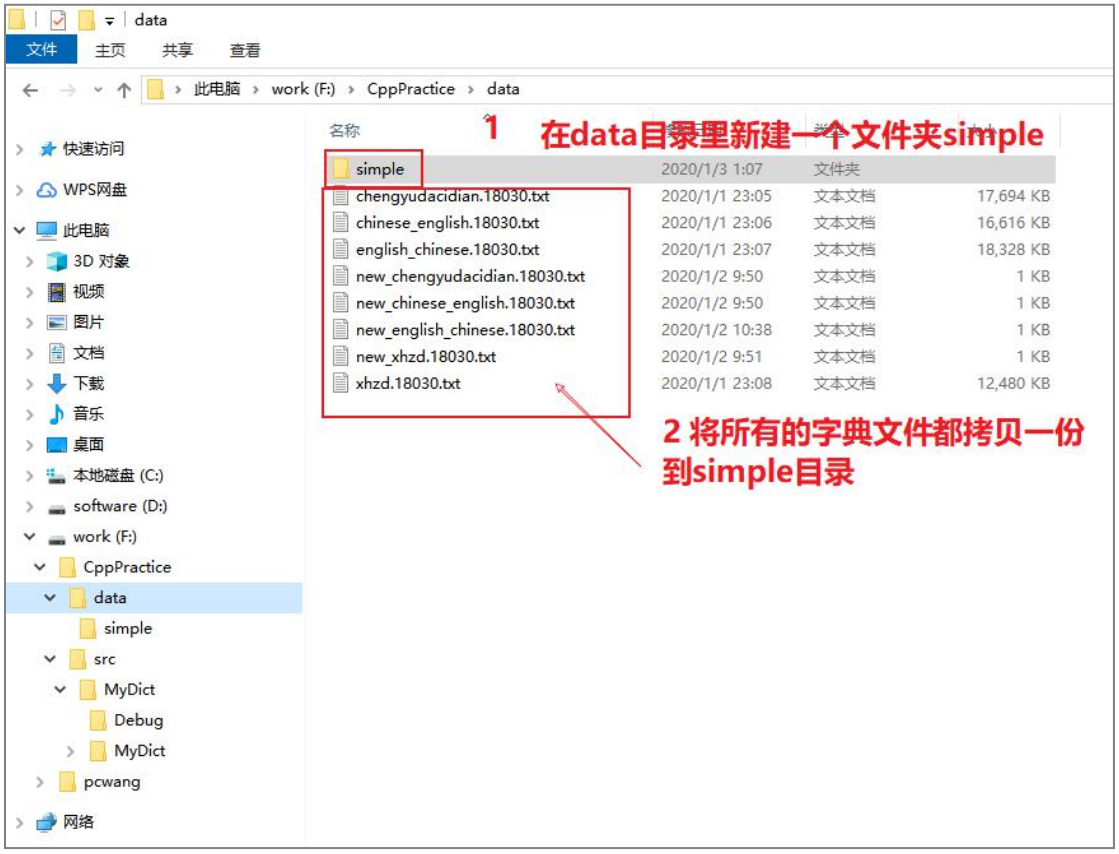
输入单词，试试我们的字典是否可用：

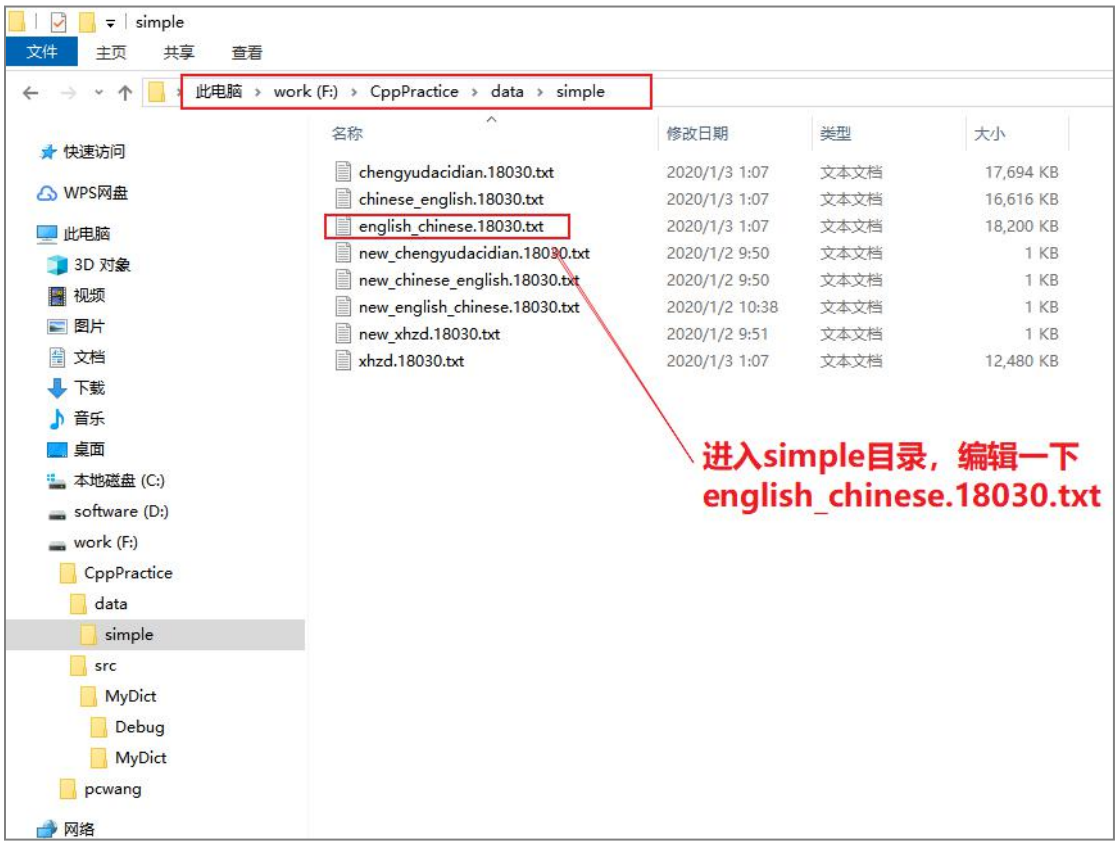
```
*********************************************
*       英汉词典                            *
*********************************************
 请输入英文：book
 查询结果是：

  *[buk]
  n. 书，书籍，帐簿，名册，工作簿
  vt. 登记，预订
  vi. 登记，预订
  【计】 工作簿
  【经】 帐本，注册，挂号；登记入册，预定
  相关词组:
    suit sb's book
    throw the book at...
    without book
    throw the book at someone
    book sth up
    take a leaf out of someone's book
    bring sb to book for sth
    by the book
    know like a book
    make book
*********************************************
 请选择(1 加入生词本,0 返回上级,其他 继续):
```

2.4 解决调试时的时间过长问题

如果我们每次调试，试运行的时候总是要等待这么就，会烦死的。想个办法：




simple
文件
主页
共享
查看
此电脑 › work (F:) › CppPractice › data › simple
名称
修改日期
类型
大小
chengyudacidian.18030.txt 2020/1/3 1:07 文本文档 17,694 KB
chinese_english.18030.txt 2020/1/3 1:07 文本文档 16,616 KB
english_chinese.18030.txt 2020/1/3 1:07 文本文档 18,200 KB
new_chengyudacidian.18030.txt 2020/1/2 9:50 文本文档 1 KB
new_chinese_english.18030.txt 2020/1/2 9:50 文本文档 1 KB
new_english_chinese.18030.txt 2020/1/2 10:38 文本文档 1 KB
new_xhzd.18030.txt 2020/1/2 9:51 文本文档 1 KB
xhzd.18030.txt 2020/1/3 1:07 文本文档 12,480 KB
快速访问
WPS网盘
此电脑
3D 对象
视频
图片
文档
下载
音乐
桌面
本地磁盘 (C:)
software (D:)
work (F:)
CppPractice
data
simple
src
MyDict
Debug
MyDict
pcwang
网络
进入simple目录，编辑一下
english_chinese.18030.txt


english_chinese.18030.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
435468
这个数有点大，我们先改小点，譬如改成10000
a
4
*[ei]
art. —
【计】 累加器, 加法器, 地址, 振幅, 模拟, 区域, 面积, 汇编, 组件, 异步
【医】 热力效能
A and B agglutinogens
1
【医】 A和B凝集原
A AND NOT B gate
1
【计】 禁止门
A as well as B
1
不仅有(乙)而且有(甲)
A B. S. pill
1
【医】 ABS丸, 芦荟素颠茄士的宁丸
a back number
1
过期的刊物, 过时的人
a bad actor
1
Unix (LF)
第 1 行，第 1 列
100%

改成 10000 表示我们只读取前 10000 个单词和短语。

再修改一下 dict.h：

2.5 汉译英

看你的咯

2.6 成语词典

看你的咯

2.7 新华字典

看你的咯

提示：汉译英、成语词典、新华字典的功能和英译汉都差不多。

2.8 生词入库

这个。。。，也看你的咯。

提示：每个字典、词典我们都对应有一个生词本的。再装载词典时，我们也将生字本装载进来。生词本本质上也时一个词典，只不过时对应词典的子集，小很多。我们可以使用和装载词典同样的函数先将生词本装载进内存。词典刚开始的时候，生词本里没有数据。生词入库的时候就是将查询到的单词和释义都加入生词本中，然后将生词本的内容写到文件里。下次打开的时候我们就有生词了。

3 字词考试功能

如何进行字词考试？我们的设想是从词库里随机抽取 n 个单词和它们的释义。在界面里显示释义，要求输入对应的单词。由程序来判定对错。

3.1 跳转到考试界面

增加两个文件 exam.h、exam.cpp

exam.h:

```
#ifndef _EXAM_H
#define _EXAM_H

#include <string>
#include <vector>
#include "dict.h"

using namespace std;

void exam_ui_display();
void exam_ui_chinese_english();

#endif // !_EXAM_H
```

examp.cpp:

```
#include "exam.h"
#include "ui.h"
#include <iostream>
#include <string>

using namespace std;

static const string exam_ui_prompt =
"\n\n"
"\t|*****************************************|\n"
"\t|*   请选择考试类别                        *|\n"
"\t|*****************************************|\n"
"\t|*            1 看中文写英文             *|\n"
"\t|*            2 看英文写中文             *|\n"
"\t|*            3 看解释写成语             *|\n"
"\t|*            0 返回上级                 *|\n"
"\t|*****************************************|\n"
"\t  请输入（1,2,3 或 0）："
;

void exam_ui() {
    CLEAR();
    cout << exam_ui_prompt;
```

```
        string op;
        std::getline(cin, op);
        if (op == "1") {
            exam_ui_english_for_chinese();
        }
        else if (op == "2") {

        }
        else if (op == "0") {
            return;
        }
        exam_ui();

    }


    void exam_ui_english_for_chinese(){

    }
```

编译，试运行。

3.2 Examination 数据结构及试卷生成

exam.h 代码：

```
#ifndef _EXAM_H
#define _EXAM_H

#include <string>
#include <vector>
#include "dict.h"

using namespace std;

typedef struct {
    vector<string> words;
    vector< vector<string> > meanings;
}Examination;

void exam_create(Examination* exam, Dictionary* dict);

void exam_ui();
void exam_ui_english_for_chinese();

#endif // !_EXAM_H
```

添加的代码

exam.cpp 代码：

包含头文件 stdlib.h,time.h。

添加代码：

```
/**
生成一个[min,max)之间的随机整数，
*/
static int random_index(int min, int max) {
    return (int)(min + (double)rand() / (double)RAND_MAX * (max - min));
}
```

```
/*
从字典中随机生成一个试卷
*/
void exam_create(Examination* exam, Dictionary* dict) {
    srand(time(NULL));
    int wordcount = dict->words.size();

    int itemcount = wordcount < 10 ? wordcount : 10;
    for (int i = 0; i < itemcount; i++) {
        int idx = random_index(0, wordcount);
        exam->words.push_back(dict->words[idx]);
        exam->meanings.push_back(dict->meanings[idx]);
    }

}
```

每次生成 10 个小题。

3.3 测验界面

在 ui.h、ui.cpp 中给输入弄一个辅助函数 get_input_string:

ui.h 中新增原型：

```
string get_input_string(const string& prompt);
```

ui.cpp 中实现它:

```
string get_input(const string& prompt) {
    cout << prompt;
    string word;
    std::getline(cin, word);
    return word;
}
```

修改 exam.cpp 中的 exam_ui_english_for_chinese 函数：

```
void exam_ui_english_for_chinese(){
    //装载词库
    dict_load(
        &dictionaries[ENGLISH_CHINESE],
        DICT_PATH DICT_ENGLISH_CHINESE
    );

    //生成考试题目
    Examination exam;
    exam_create(&exam, &dictionaries[ENGLISH_CHINESE]);
```

```
        for (int i = 0; i < exam.words.size(); i++) {
            CLEAR();
            cout << "\n\n";
            cout << "\t|*****************************************|\n";
            meaning_display(exam.meanings[i]);
            cout << "\n";
            cout << "\t|*****************************************|\n";
            string word = get_input_string("\t  请输入英文答案(0 退出) : ");
            //cout << "\t  请输入英文答案(0 退出) : ";
            //string word;
            //std::getline(cin, word);
            if (word == exam.words[i]) {
                cout << "\t  正确!!! " << endl;
                string op = get_input_string("\t  请输入( 0 退出,其他 下一题): ");
                if (op == "0") {
                    break;
                }
            }
            else if (word == "0") {
                break;
            }
            else {
                cout << "\t   错误,继续加油哦" << endl;
                //如果搞错了，就继续答这道题
                i--;
            }
        }
    }
```

3.4 计分功能

看你的咯。

提示：只要给 Examination 数据结构增加一个标记数组，标记某个题目是否答对。在所有的题目答完之后做一个成绩统计就 ok 了。不要太多的代码哦。

3.5 其他考试

上面只完成了看中文写英文，还有其他的两个共能没有写呢。完成它吧。

4 生词复习

找出一个个生词，显示它的释义，输入对应的生词。如果答对，提示是否从生词本移除，并作出相应的操作。

也看你的咯。