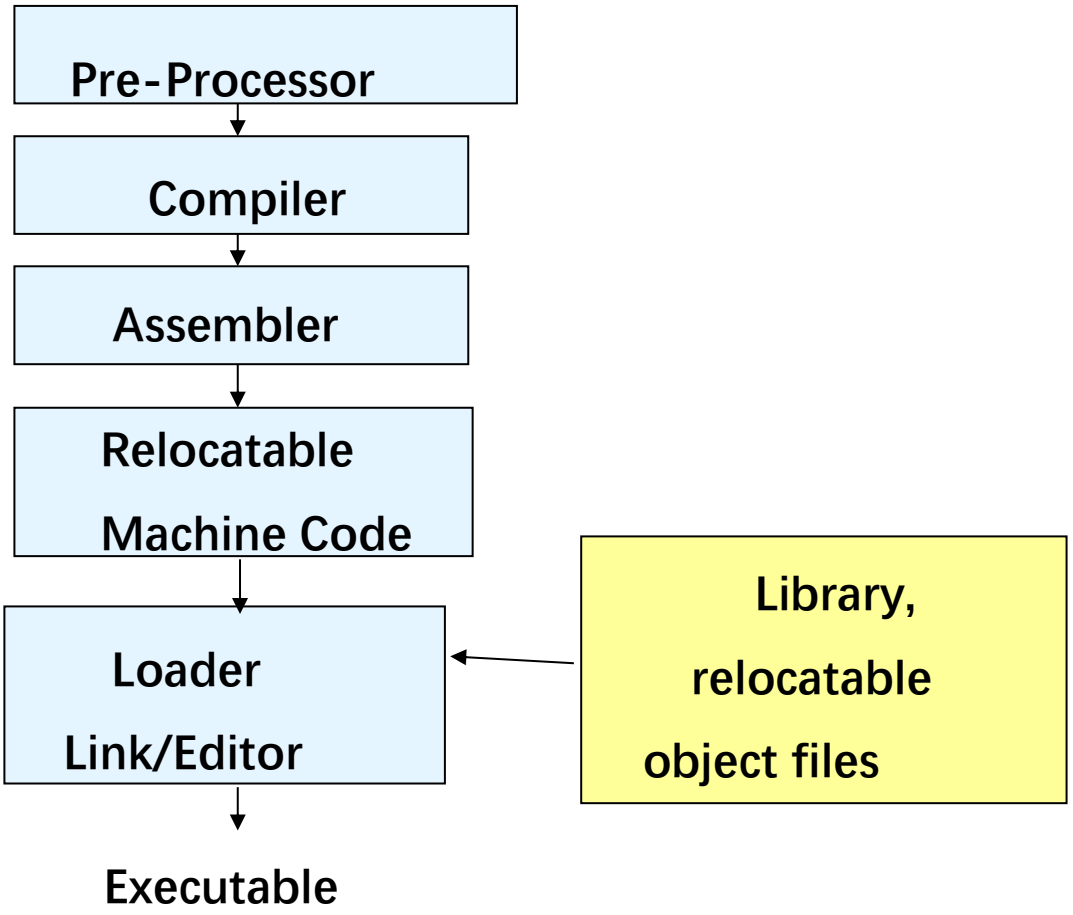
预备工作 1——了解你的编译器（4 分）

以你熟悉的编译器，如 VC、GCC（TDM-GCC）为研究对象，更深入地探究语言处理系统的完整工作过程：

1. 预处理器做了什么？
2. 编译器做了什么？
3. 汇编器做了什么？
4. 链接器做了什么？

大家可以结合编译原理课程上的第一个 PPT 上较为靠后的内容体会语言处理的具体过程，此处只列举一张概括图：



下面简要回答仅作参考（需要同学们在实验过程中亲自动手体验，细化各个阶段的作用）：

1. 预处理器：处理源代码中以 # 开始的预编译指令，例如展开所有宏定义、插入 #include 指向的文件等，以获得经过预处理的源程序。

2. 编译器：将预处理器处理过的源程序文件翻译成为标准的汇编语言以供计算机阅读。

3. 汇编器：将汇编语言指令翻译成机器语言指令，并将汇编语言程序打包成可重定位目标程序。

4. 链接器：将可重定位的机器代码和相应的一些目标文件以及库文件连接在一起，形成真正能在机器上运行的目标机器代码。

方法：

以一个简单的 C（C++）源程序为例（如下面的阶乘程序、斐波那契程序）利用编译器的程序选项获得各阶段的输出，研究它们与源程序的关系。

具体操作说明（以 VC 编译器为例，以下为示例代码）：

```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main()
{
    //声明两个新的变量
    int a, b;
    //输入其中的两个变量的值
    cin >> a >> b;
    //输出变量之和
    cout << a + b << endl;
    system("pause");
    return 0;
}
```

(一) 预处理器

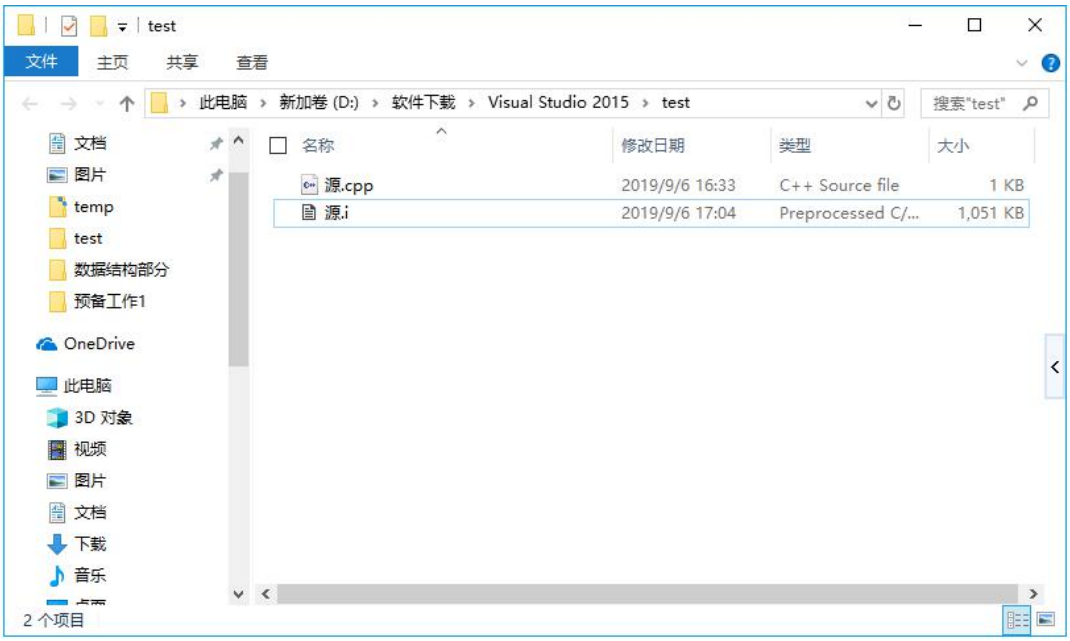
处理阶段会处理源代码中所有的预编译指令，预编译指令全部以“#”头，如 #include、#define、#if 等，其中 #include 命令中包含的头文件会被替换成相应的文件内容，#define 定义的宏命令，会直接进行相应内容的替换。处理之后的输出文件就完全是个纯 C++ 源文件，不再包含预处理命令。

打开 VS2015 开发人员命令提示框，在命令行中输入指令 cl /P 源.cpp，并运行。其中“/P”表示“将预处理器输出写入文件”。 如图下所示：

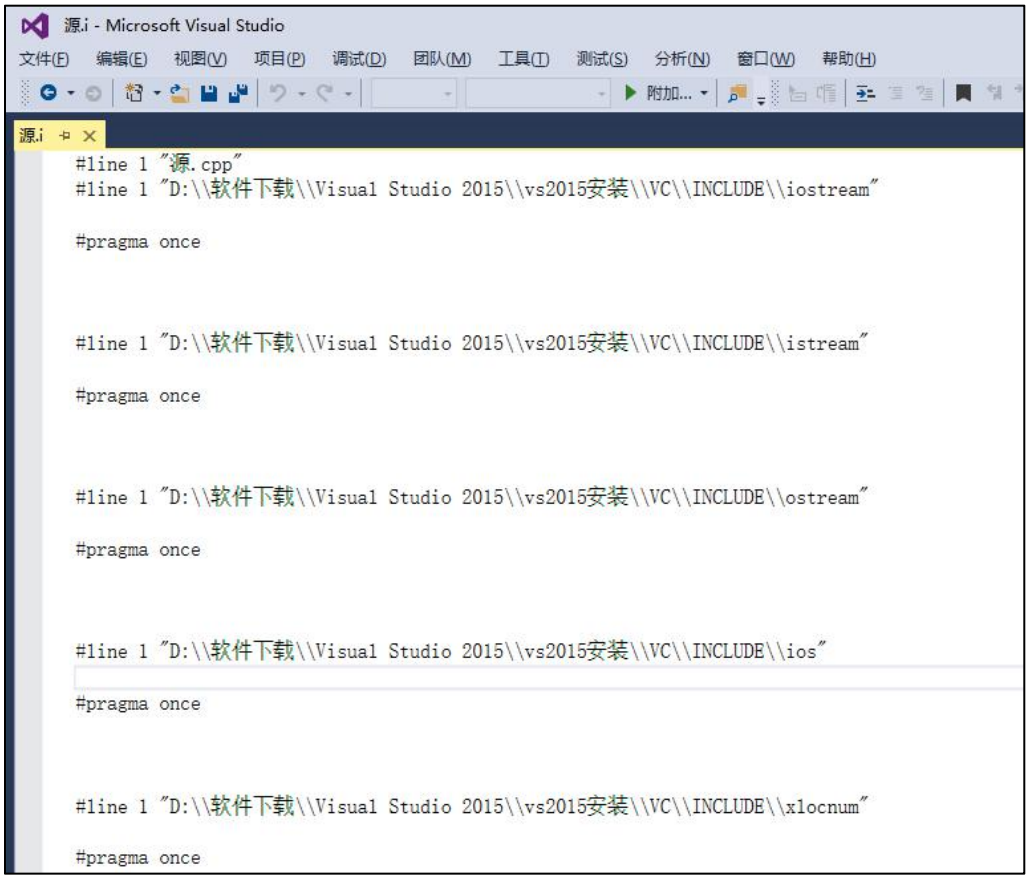
```
D:\软件下载\Visual Studio 2015\test>cl /P 源.cpp
用于 x64 的 Microsoft (R) C/C++ 优化编译器 19.00.23506 版
版权所有(C) Microsoft Corporation。保留所有权利。

源.cpp

D:\软件下载\Visual Studio 2015\test>
```

预处理结果

1. 打开源程序所在的目录，可以看到目录中添加了预处理产生的源.i，如图所示。



2. 源.i 的部分内容如图所示。我们可以发现生成的源.i 的文件大小远大于源.cpp. 这是由于预处理程序是将源程序的诸多头文件、宏定义进行了替代。经过此种“替代”，生成一个没有宏定义、编译指令、没有特殊符号的预处理文件。



3. 下面进行分析，可以看到，头文件中的内容被完整替换到 #include 命令所在的位置。进一步观察我们可以发现，源.cpp 中的注释语段“返回阶乘的数值”在预处理生成的源.i 文件中被删除了。

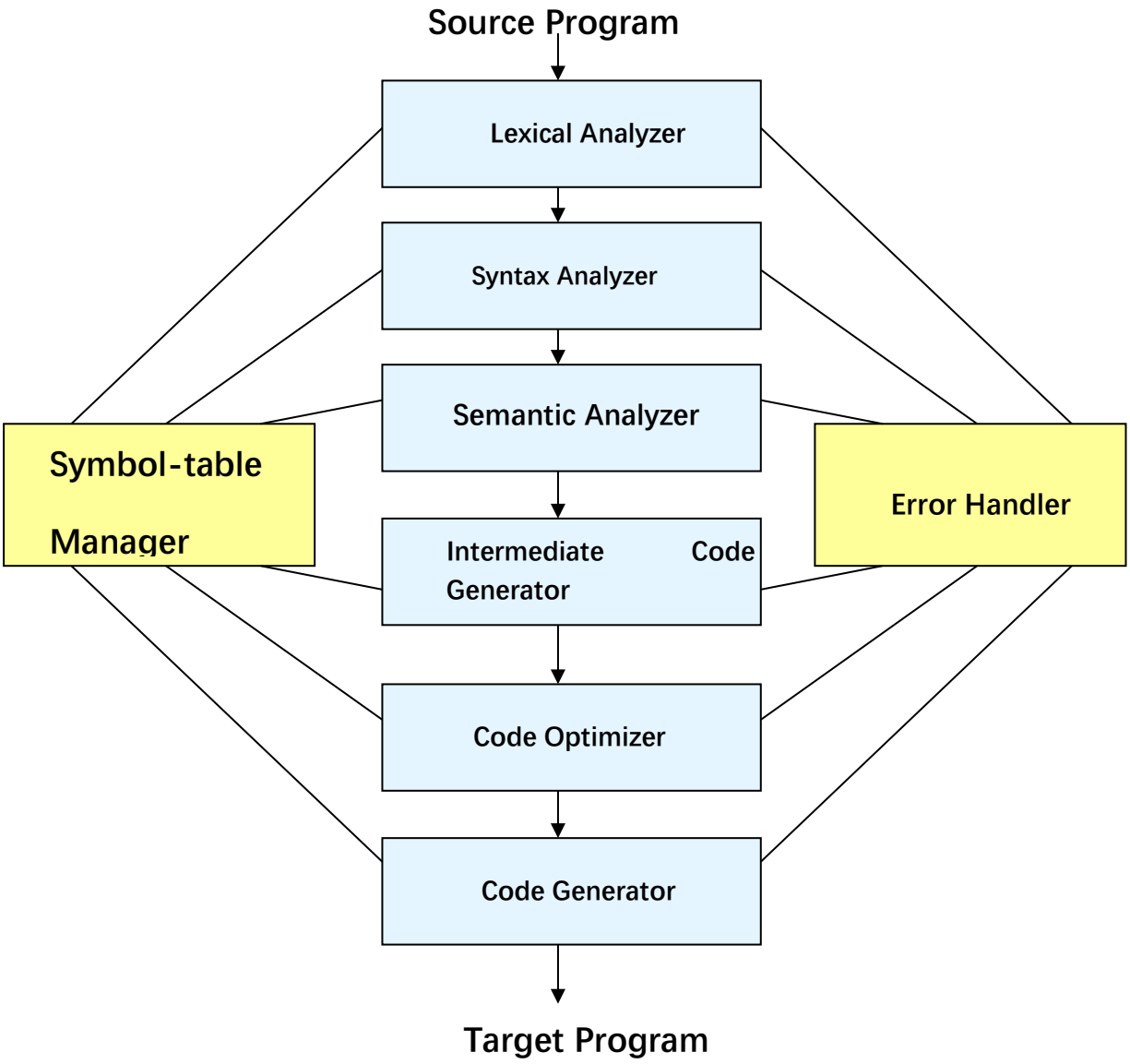
```
#line 2 "源.cpp"

using namespace std;
int main()
{

    int a, b;

    cin >> a >> b;

    cout << a + b << endl;
    system("pause");
    return 0;
}
```

希望同学们在报告中以此为基础详细分析并阐述预处理的结果以及功能。

(二) 编译器

编译过程就是把预处理阶段得到的文件进行一些分析优化后生成相应的汇编代码。具体分为 6 个步骤（**以下的 6 个步骤在课程的第一个 PPT 中阐述的十分详细，存在问题的同学可以对 PPT 进行查看，以便更加深刻地理解编译过程，其实也是编译原理课程中较为核心的内容，在日后的课程和作业中会有清晰明确的体现**）：

1. 词法分析: 将字符序列转换为单词序列（token）的过程。
2. 语法分析: 使用由词法分析器生成的各个词法单元的第一个分量来创建树形的中间表示。
3. 语义分析: 使用语法树和符号表中的信息来检查源程序是否和语言定义的语义一样; 收集类型信息，把这些信息存放在语法树或符号表中; 类型检查;
自动类型转换。
4. 中间代码生成: 源程序的语法分析和语义分析完成之后, 很多编译器生成一个明确的低级的或类机器语言的中间表示。
5. 代码优化: 机器无关的代码优化步骤试图改进中间代码，以便生成更好的目标代码。
6. 代码生成: 以源程序的中间表示形式作为输入，并把它映射到目标语言。



在命令行输入指令 cl/c/Fa/Tp/actoriali，并运行。则开始进行编译过程。其中各参数如下：
/c：编译但不链接。
/Fa：创建程序集列表文件。
/Tp：指定 C++ 源文件。
运行之后得到警告，如下图第一条指令。

```
D:\软件下载\Visual Studio 2015\test>cl /c /Fa /Tp 源.i
用于 x64 的 Microsoft (R) C/C++ 优化编译器 19.00.23506 版
版权所有(C) Microsoft Corporation。保留所有权利。

源.i
D:\软件下载\Visual Studio 2015\vs2015 安装\VC\INCLUDE\xlocale(341): warning C4530: 使
```

用了 C++ 异常处理程序，但未启用展开 语义。请指定 /EHsc
D:\软件下载\Visual Studio 2015\vs2015 安装\VC\INCLUDE\exception(358): warning C4577: 在未指定异常处理模式的情况下使用了 "noexcept"；不一定会在异常时终止。指定 /EHsc

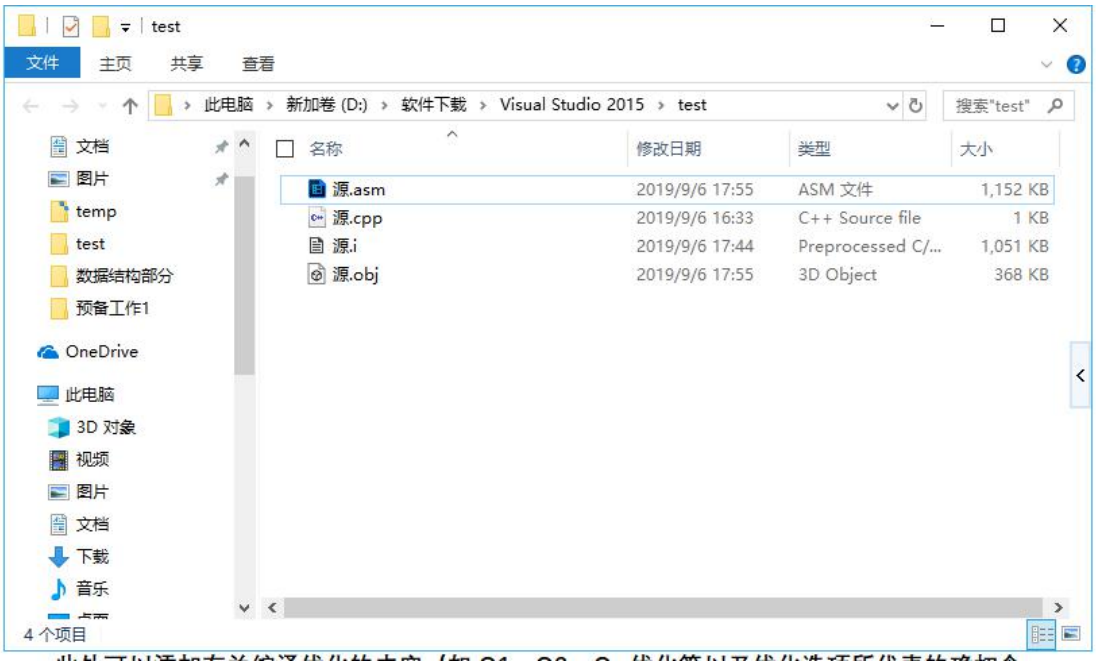
D:\软件下载\Visual Studio 2015\test>cl /EHsc /c /Fa /Tp 源.i
用于 x64 的 Microsoft (R) C/C++ 优化编译器 19.00.23506 版
版权所有(C) Microsoft Corporation。保留所有权利。

源.i

经查阅相关资料可知：指定当编译器使用的异常处理类型、何时优化掉异常检查以及是否销毁由于异常而超出范围的 C++ 对象。如果未指定/EH，则编译器将同时捕获异步结构化异常和 C++ 异常，但不会销毁由于异步异常超出范围的 C++ 对象。如果与 sc(/EHsc) 一起使用，则仅捕获 C++ 异常并通知编译器假定声明为 extern "C"的函数从未引发 C++ 异常
添加参数之后重新执行指令，如上图第二条指令。

编译器处理结果：

通过观察可知，在执行完上述指令之后，生成了源.asm 和源.obj 两个文件，如图所示。



在此我们只讨论生成的源.asm 文件（源.obj 文件将主要在下一节讨论）。打开源.asm，发现编译器生成了一个汇编程序。

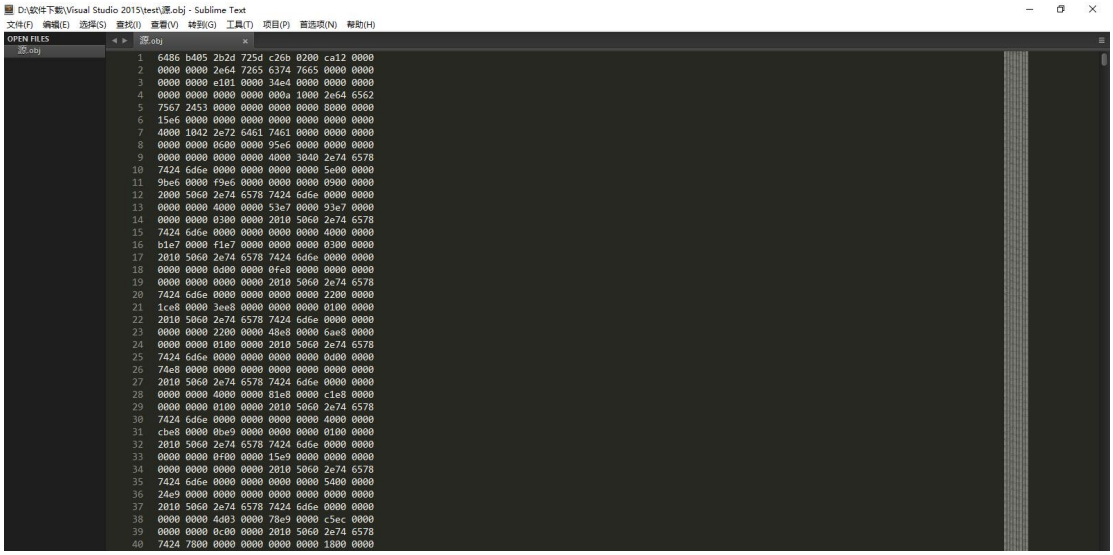
希望同学们在报告中详细分析并阐述编译器处理的结果以及编译器的功能。
此处可以添加有关编译优化的内容（如 O1，O2，Ox 优化等以及优化选项所代表的确切含义，也可以通过具体实例来比较不同优化或者优化选项所带来的不同优化效果）

(三) 汇编器

汇编过程实际上把汇编语言程序代码翻译成目标机器指令的过程。其最终生成的是可重定位的机器代码。由于在本文中的语言处理过程中，在执行了编译命令后，既生成了.asm 为后缀的汇编程序代码，又生成了.obj 为后缀的可重定位文件。故在此阶段不需要进行其他操作。

汇编器处理结果:

汇编器处理后产生以.oj 为后缀的可重定位目标文件，本质上是与源程序等效的机器语言代码，如图所示。



希望同学们在报告中详细分析并阐述汇编器处理的结果以及汇编器的具体功能分析。

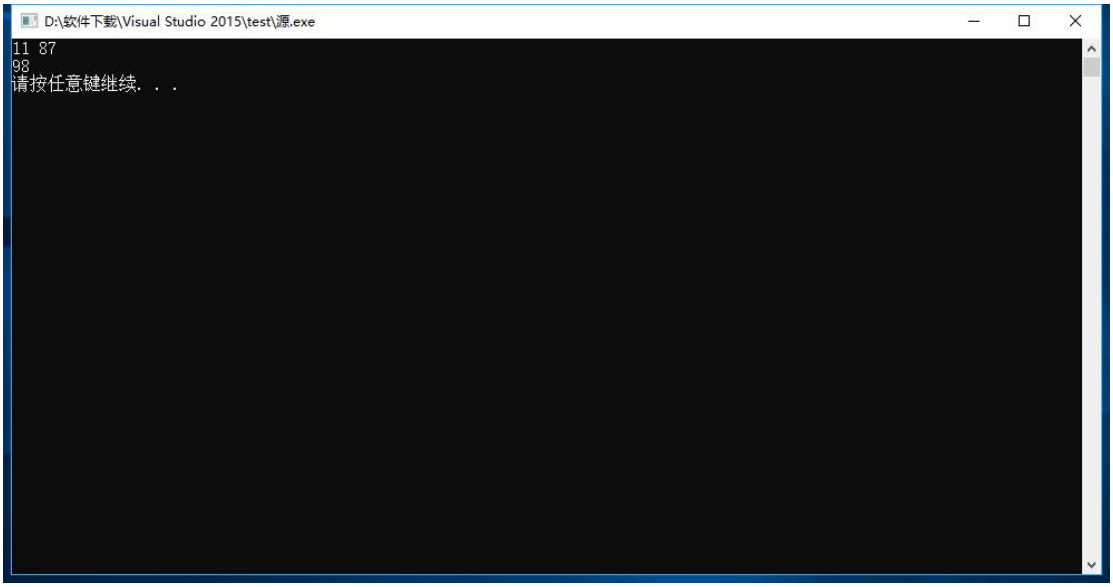
(四) 链接器 加载器

由汇编程序生成的目标文件不能够直接执行，由于大型程序经常被分成多个部分进行编译，因此，可重定位的机器代码有必要和其他可重定位的目标文件以及库文件连接到一起，最终形成真正在机器上运行的代码。进而由加载器对该机器代码进行执行生成可执行文件。输入指令 link 源.obj，并运行，执行链接操作，如图所示。

```
D:\软件下载\Visual Studio 2015\test>link 源.obj
Microsoft (R) Incremental Linker Version 14.00.23506.0
Copyright (C) Microsoft Corporation.  All rights reserved.
```

链接器处理结果：

通过链接最终生成可执行程序源.exe，双击运行测试，可见其实现了计算两个整数求和的功能，如图所示。



希望同学们在报告中详细分析并验证链接器、加载器处理的结果以及具体功能分析。

注意：建议使用 latex 进行书写