C++集成开发环境

1.1 程序调试

1.1.1 程序调试的概念

程序调试（Debug）是将编制的程序投入实际运行前，用手工或编译程序、调试工具等方法进行测试，以修正程序错误（bug）的过程。

从编写程序到程序的运行，需要经过如下步骤：

1）编辑：依据语法规则编写源程序，编写完成后保存文件，C++语言程序文件的扩展名为“.cpp”。

2）编译：用专门的软件——编译程序对源程序进行语法检查并将符合语法规则的源程序语句翻译成计算机能识别和执行的二进制机器指令，生成由机器指令组成的目标程序，扩展名为“.obj”。

3）链接：编译后的程序可能存在一个文件引用了另一个文件中的符号（变量或函数调用等）或者在程序中调用了某个库函数等问题。因此，必须把这些符号、库函数的处理过程（“代码”）链接（“插入”）到经过编译生成的目标程序中，最终生成可执行文件。Windows 平台下的可执行文件扩展名为“.exe”，而 Linux 平台对可执行文件的后缀名没有特别要求。

4）运行：运行链接后生成的可执行程序。

在程序设计、编写代码的过程中，不可避免地会发生各种各样的错误，尤其是代码规模比较大的程序，出错的可能性更高、错误的种类也更多。通过调试来查找和修改错误是软件设计开发过程中非常重要的环节。调试也是程序员必须掌握的技能，可以说不会调试就无法开发软件。

1.1.2 程序错误的种类

本节将通过下面的例子详细介绍程序错误的种类。

【例 1.1】以下程序预期实现“计算数组 *a* 中所有元素和”的功能。

```
1 #include <iostream>
2 using namespace std;
3 class SUM
4 {
```

```
 5  public:
 6      SUM();
 7      static const int N = 5;
 8      int Add();
 9  private:
10      int n;
11      int a[N];
12 };
13 SUM::SUM()
14 {
15      cout << "Input n:";
16      cin >> n;
17      for (int i=0; i<n; i++)
18      {
19          a[i] = i + 1;
20      }
21 }
22 int SUM::Add()
23 {
24      int sum;
25      for (int i=0; i<n; )
26      {
27          sum += a[i];
28      }
29      return sum;
30 }
31 int main(void)
32 {
33      SUM s;
```

```
34          int sum = Add();
35          cout << "sum = " << sum << endl;
36          return 0;
37 }
```

在分析程序错误之前，需要了解程序会遇到哪些错误。一般来说，一个程序会遇到以下几种常见错误。

（1）编译错误

编译错误是指在编译阶段，编译器根据 C++语言的语法规则就能发现的错误，如保留字输入错误、括号不匹配、语句少分号等。编译器一般能指出编译错误的位置（错误所在的代码行号）、错误的内容，所以这类错误能方便地根据编译器给出的出错提示进行修改。例 1.1 中第 34 行调用 SUM 类的方法 Add()时没有写对象名，这会导致编译错误。

（2）链接错误

通常，由于缺少包含函数定义/类模板定义的头文件或者包含头文件的路径错误等原因会导致链接时出错。

（3）运行时错误

程序通过了编译、链接之后，虽然能够运行，但在运行过程中也可能出现错误，如无法停止执行、访问冲突等，这类错误称为运行时错误。

例 1.1 中第 25 行 for 循环内没有对循环变量进行更新处理，在语法层面没有错误，所以编译时不会报错，但会使 for 语句的循环无法停止，导致死循环，这个错误就属于运行时错误。

与编译错误相比，运行时错误难发现、难修改。运行时错误产生的原因有以下几种：

- 程序采用的算法本身存在错误，注定不会产生正确结果，例如计算时使用了不正确的计算公式。
- 代码实现（编程）层面的错误，如数组越界访问、堆栈溢出等。
- 缺少对错误输入的容错考虑，程序产生了不正常的计算结果。

所以，在编写程序时，我们需要仔细检查程序的算法、逻辑和执行顺序等是否正确。利用编程环境的调试工具跟踪程序的执行，了解程序在运行过程中的状态变化情况，如关键变量的数值等，可以帮助我们快速定位并修改错误。

1.1.3 常用调试方法

调试程序通常需要借助工具软件（即调试器）来实现。集成开发环境中一般都带有调试功能，常用的调试方法如下。

1. 设置断点

所谓断点（Breakpoint），就是在程序运行过程中使其暂停运行的位置（代码行）。暂停时，断点所在代码行尚未执行。程序暂停后，可以方便我们观察程序运行过程中变量的数值、函数调用情况等，分析程序是否运行正常、程序运行异常的原因等。调试工具一般都有在程序中设置断点的功能。一个程序可以设置多个断点。每次运行到断点所在的代码行，程序就会暂停。

条件断点是指给断点设置条件，该条件满足时，这个断点才会生效，暂停程序的运行。条件断点在调试循环程序时非常有用。试想一个循环 1000 次的程序，如果每次循环都中断，将是无法接受的。而通过观察循环程序的特点，用可能导致程序异常的变量数值、边界数值等对断点设置一定的条件，仅在该条件为“真”（true）时才暂停程序，调试将变得更高效、更直接。一般的调试工具都支持条件断点功能。

2. 单步跟踪

当程序在设置的某断点处暂停时，调试工具会提供单步运行并暂停的功能，即单步跟踪。通过单步跟踪，可以逐个语句或逐个函数地执行程序，每执行完一个语句或函数，程序就暂停，从而可以逐个语句或逐个函数地检查它们的执行结果。

断点所在行的代码是下一次单步执行要被执行的代码，称为当前代码行。此时对程序的单步跟踪执行有 6 个选择。

1）单步执行（Step over）：执行一行代码，然后暂停。当存在函数调用语句时，使用单步执行会把整个函数视为一次执行（即不会在该函数中的任何语句处暂停），直接得到函数调用结果。该方式常用在多模块调试时期，可以直

接跳过已测试完毕的模块，或者直接通过函数执行后的值来确定该测试模块中是否存在错误。

2）单步进入（Step into）：如果此行中有函数调用语句，则进入当前所调用的函数内部调试，在该函数的第一行代码处暂停；如果此行中没有函数调用，其作用等价于单步执行。该方式可以跟踪程序的每一步执行过程，优点是容易直接定位错误，缺点是调试速度较慢。所以一般在调试时，先划分模块，对模块进行调试，尽量缩小错误范围，找到错误模块后再使用单步执行和断点来快速跳过没有出现错误的部分，最后才使用该方式来逐步跟踪找出错误。

单步进入一般只能进入用户自己编写的函数。有的编译器提供了库函数的代码，可以跟踪到库函数中执行。如果库函数没有源代码，就不能跟踪进入，此时有的调试器会以汇编代码的方式单步执行函数，有的调试器则会忽略函数调用。

3）运行出函数（Step out）：继续运行程序，当遇到断点或返回函数调用者处暂停。当只想调试函数中的一部分代码，调试完想快速跳出该函数的时候，可以使用这个命令。

4）运行到光标所在行（Run to cursor）：将光标定位在某行代码上并调用这个命令，程序会执行到抵达断点或光标定位的那一行代码处暂停。如果想重点观察某一行（或多行）代码，而且不想从第一行启动，也不想设置断点，则可以采用这种方式。这种方式比较灵活，可以一次执行一行代码，也可以一次执行多行代码；可以直接跳过函数，也可以进入函数内部。

5）继续运行（Continue）：继续运行程序，当遇到下一个断点时暂停。

6）停止调试（Stop）：程序运行终止，停止调试，回到编辑状态。

3. 监视窗

当程序暂停时，需要通过监视窗来查看某个变量的值，以便确定语句是否错误。每运行到需要观察变量所在的语句处，就可以观察程序执行了哪些操作以及产生了哪些结果。因此，通过调试和观察变量能方便地找出程序的错误。

如果程序比较大，调试工作会变得耗时、费力。此时可以将程序划分为模块，先对单个或多个模块分别进行调试，最后再将模块组合在一起进行整体调试。组合的时候需要注意模块和模块之间接口的一致性。在单个模块中的调试

将变得简单，模块能缩小错误的范围，如果一个模块正确，则可以直接排除该模块，继续测试下一个模块。对于出错的模块，可逐条仔细检查各语句，再结合一些调试方法，便能找出错误所在的位置。

1.2 经典集成开发环境

使用高效、便捷的集成开发环境，有利于降低程序编写和调试的难度，提高工作效率。本节介绍 Windows 平台下三种常用的 C++语言集成开发环境：Microsoft Visual Studio、Code::Blocks 和 Dev-C++。

1.2.1 Visual Studio 集成开发环境的使用和调试方法

Microsoft Visual Studio（简称 VS）是美国微软公司开发的产品，是目前流行的 Windows 平台应用程序的集成开发环境，它不仅支持 C++语言，也兼容 C 语言。VS 功能强、软件规模庞大、价格昂贵。但微软从 2013 版开始推出了免费的社区版，即 Visual Studio Community 2013，其在功能上与专业版相同。目前已经推出了 Visual Studio Community 2019，可以从微软官网下载其安装程序。本节将介绍如何在 Visual Studio Community 2019（以下简称为 VS2019）下开发和调试 C++语言程序。

1. 创建项目

在 VS 中以解决方案（Solution）为管理单位，一个解决方案可以包括多个项目（Project）。下面创建一个名为 VS2019Demo 的解决方案，包含一个名为 Test1 的项目。

第一步：启动 VS2019，首次启动的界面如图 1-1 所示。

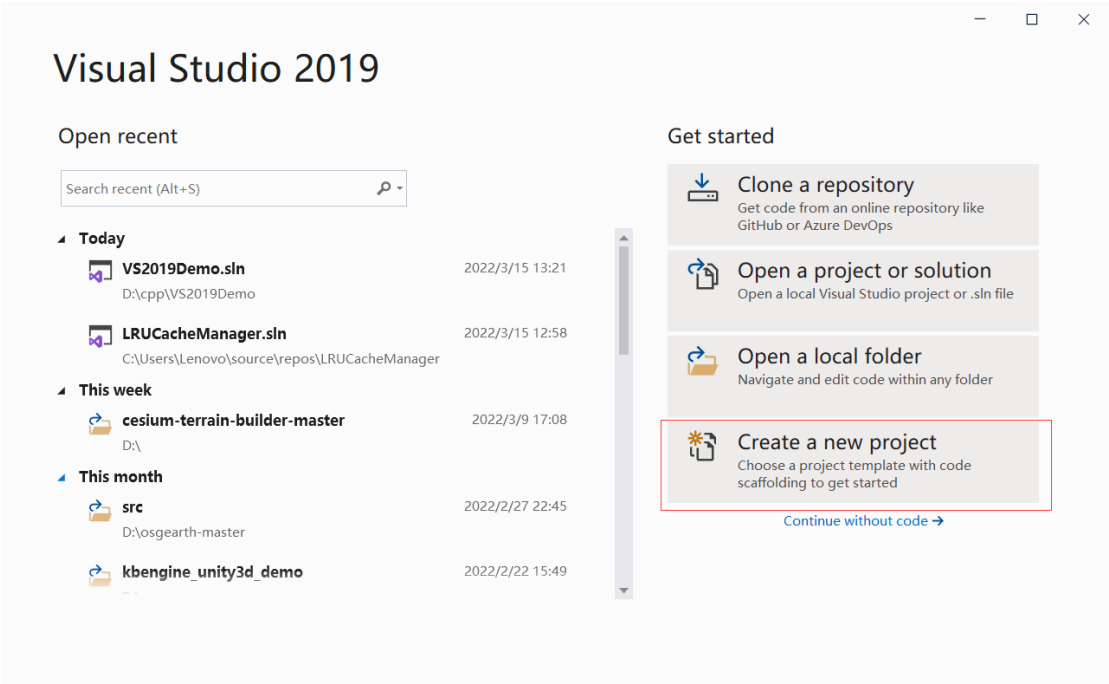

图 1-1 VS2019 启动界面

第二步：创建 C++程序的项目。在图 1-1 所示的界面中，单击“创建新项目”。也可以单击图 1-1 中窗口右下角的“继续但无需代码”，并在如图 1-2 所示的后续窗口界面中选择“文件”/“新建”/“项目”菜单。

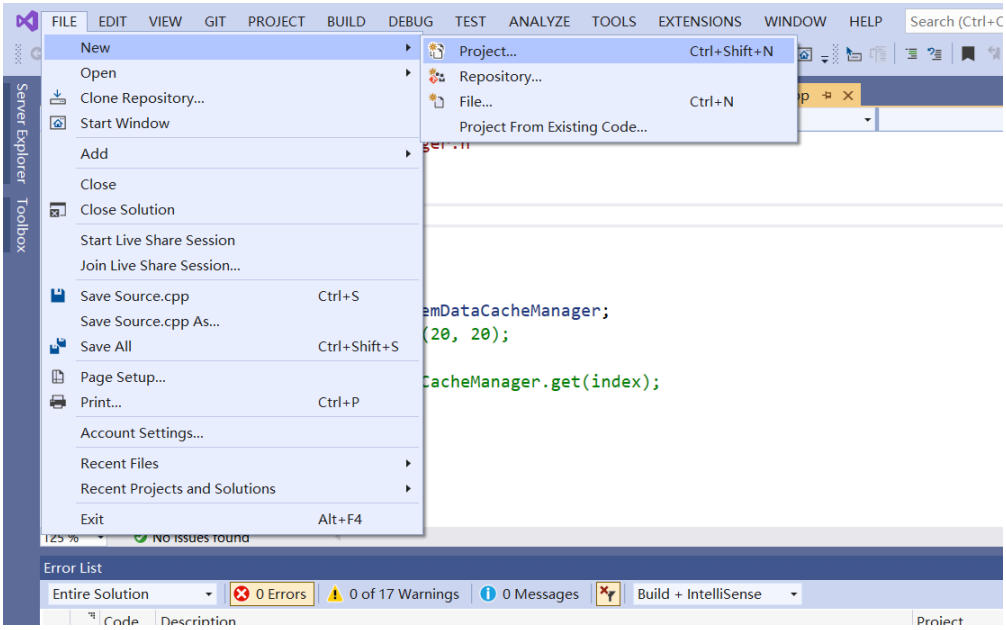

图 1-2 VS2019 中建立 C++语言程序项目

第三步：在如图 1-3 所示的窗口中，选择“空项目”，并单击“下一步”按钮。

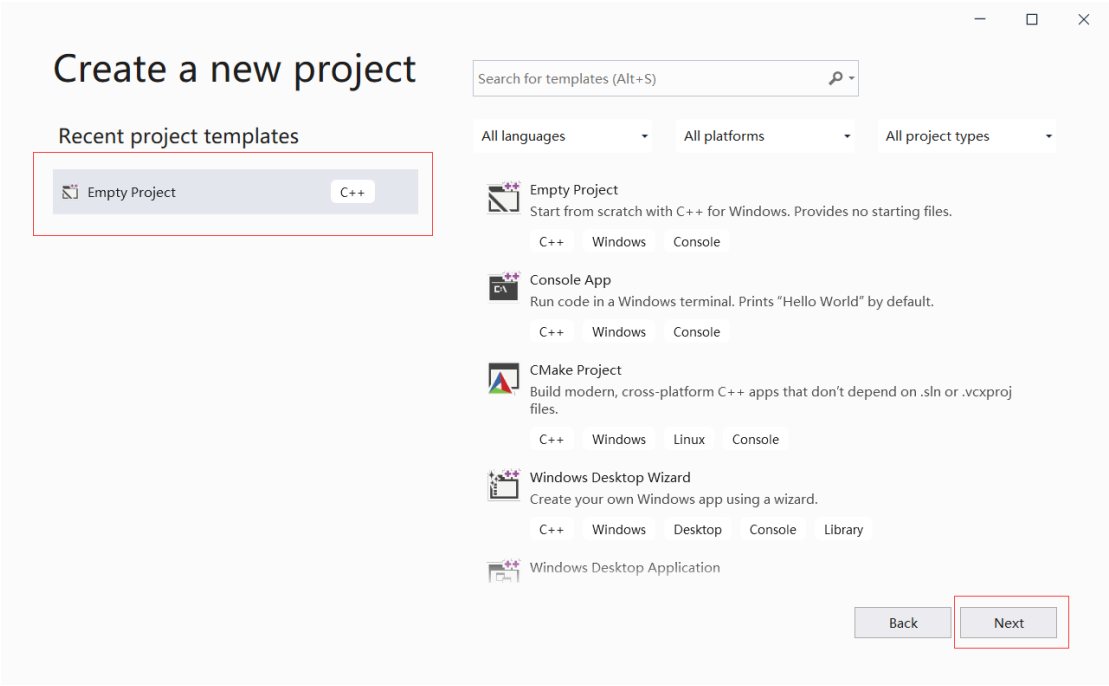

图 1-3 在 VS2019 中选择“空项目”

第四步：配置新项目。在如图 1-4 所示的窗口中，设定项目名称、保存位置和解决方案名称。在 VS 中，解决方案是最大的管理单位，一个解决方案可以包含多个项目，每个项目可以包含多个代码文件。在本例中，项目名称为 Test1，解决方案名称为 VS2019Demo，保存位置为 D:\cpp\，然后单击“创建”按钮，完成项目创建。

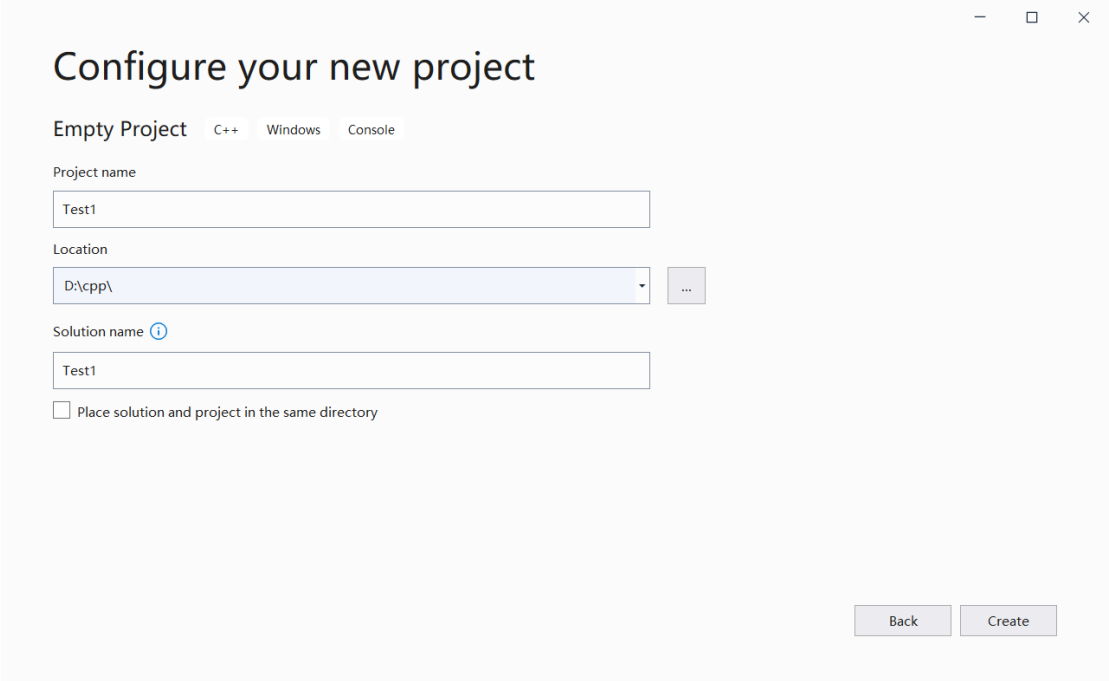

图 1-4 VS2019 创建项目的设置界面

第五步：完成项目创建后，进入如图 1-5 所示的界面。由于之前选择的是创建“空项目”，因此没有任何代码文件。但在 D:\cpp\下会有新建的与解决方案同名的文件夹 VS2019Demo，在该文件夹下有 VS2019 的解决方案文件 VS2019Demo.sln 和项目 Test1 对应的子文件夹 Test1。

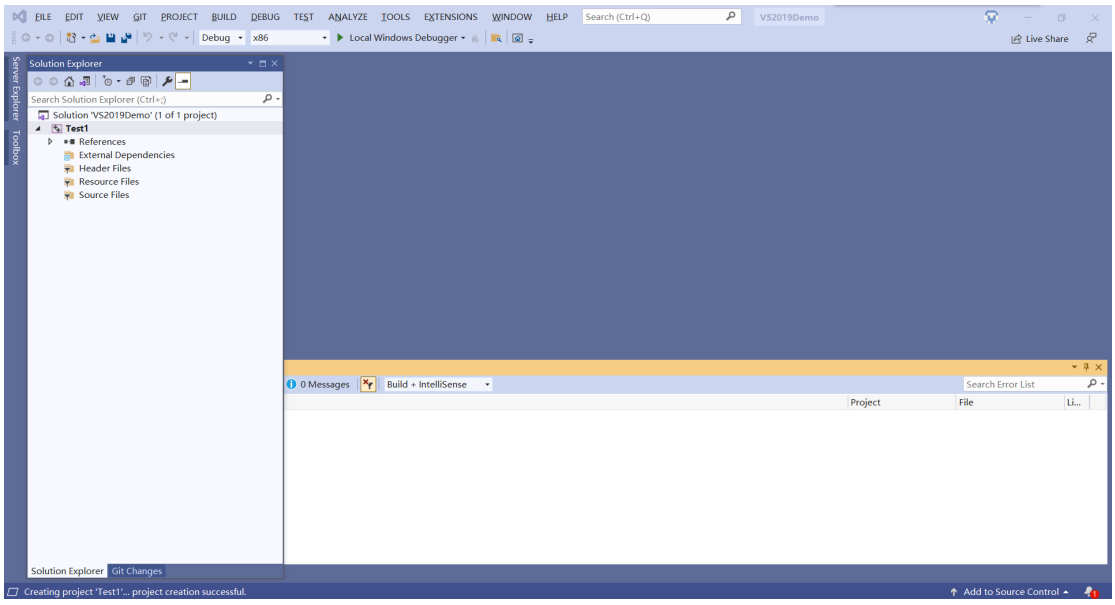

图 1-5 VS2019 创建项目后的界面

第六步：添加代码文件。在图 1-5 中的“解决方案资源管理器”窗口内的工程名 Test1 下面的“源文件”上单击鼠标右键，选择“添加”/“新建项”，界面如图 1-6 所示。也可以通过在工程名字 Test1 上单击鼠标右键，选择“添加”/“新建项”进行操作。

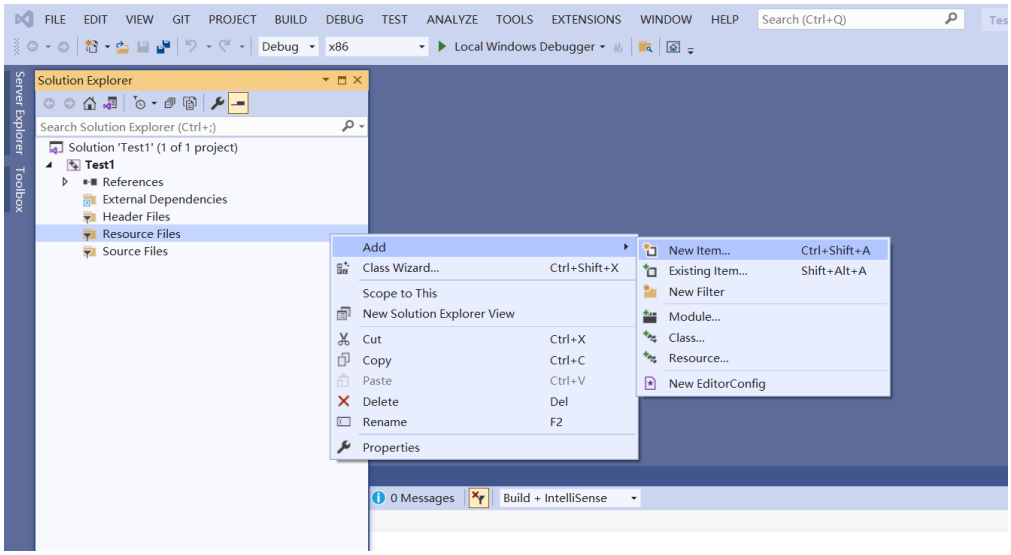

图 1-6 为项目添加新建项

第七步：设定添加项的类型和名称。在如图 1-7 所示的界面中，选择“C++文件(.cpp)”，并在下方的输入框中输入源文件名称 test.cpp，然后单击“添加”按钮，完成添加新代码文件的操作。

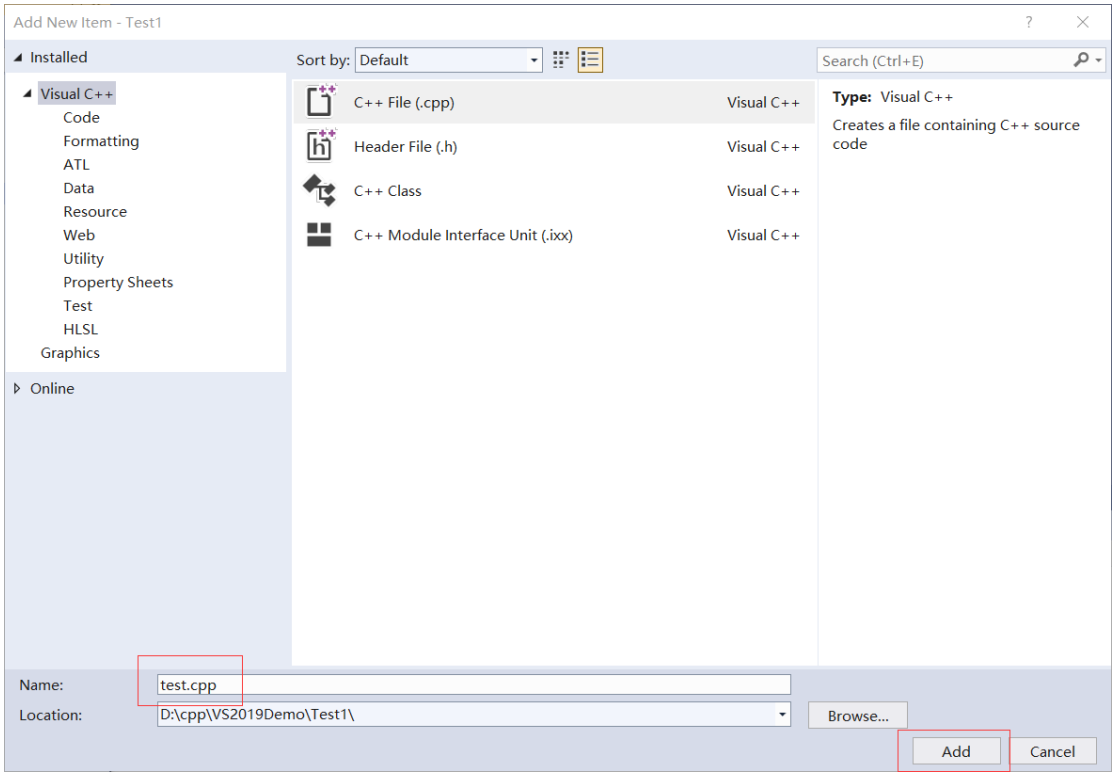

图 1-7 设定新建项为 C++语言源代码文件

添加文件 test.cpp 后，界面如图 1-8 所示。新添加的 test.cpp 是一个空文件，此时就可以在 VS2019 中编写程序了。

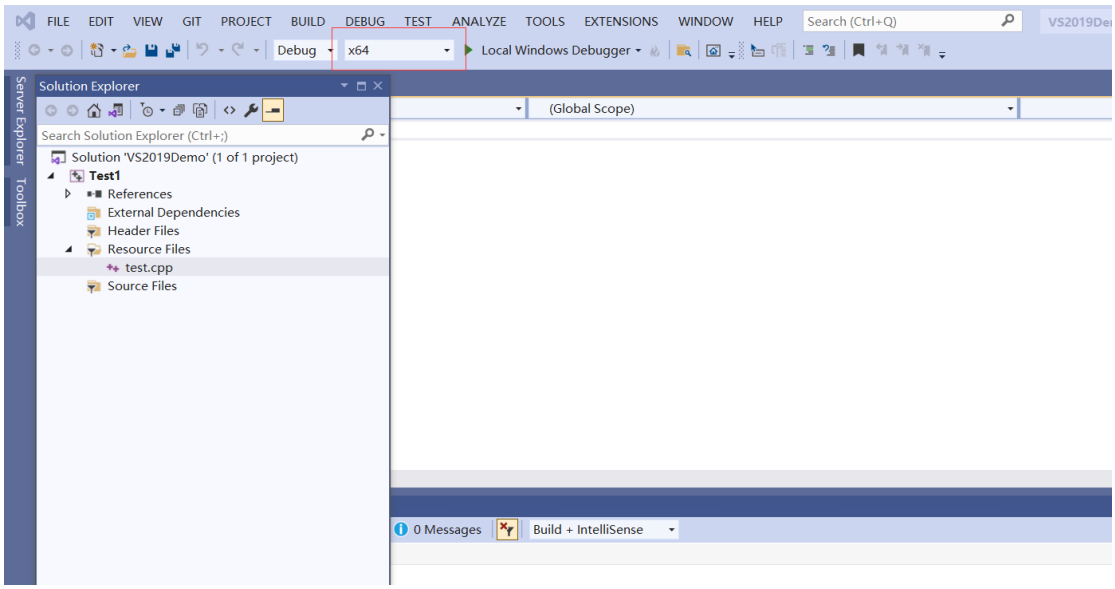

图 1-8 VS2019Demo 中添加代码文件 test.cpp 后的界面

在 VS2019 的编辑器中编辑该文件，写入完整的代码。VS2019 编辑器除了文本编辑功能以外，还有许多专门为编写代码而开发的功能，比如：

- 关键字高亮显示。

- 代码提示。
- 智能缩进。
- 按 Ctrl +]键自动寻找配套的括号。
- 按 Ctrl + K + C 键快捷注释选中代码。
- 按 Ctrl + K + U 键取消注释选中代码。

如果事先已经创建并编辑了代码文件 test.cpp，则可通过“添加”/“现有项”将源代码文件或头文件添加到工程中。这个方法可以用于将各种类型的文件添加到项目中。

2. 编译和运行

将 1.1 节的示例代码编辑到 test.cpp 中，然后单击菜单栏中的“生成”/“编译”或按 Ctrl + F7 键，就开始进行编译，如图 1-9 所示。“重新生成解决方案”将把整个项目的所有源代码重新编译，生成可执行程序。注意：在图 1-8 中圈出的“x64”表示将程序编译成 64 位程序，这是 VS 的默认值。如果需要编译成 32 位的可执行程序，可在编译前从下拉列表中选择“x86”。

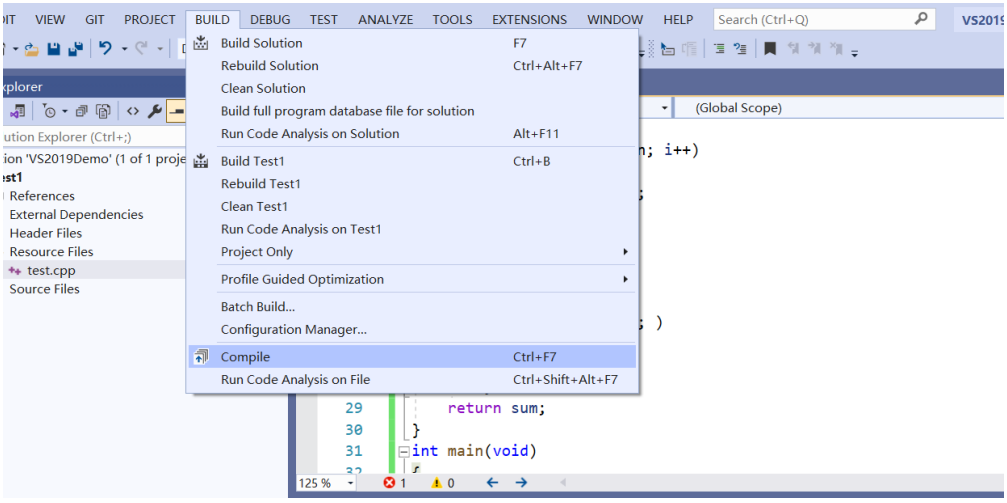

图 1-9 在 VS2019 中编译程序

VS 完成编译后，在“输出”窗口内显示生成失败的信息，如图 1-10 所示。这说明出现了编译错误。

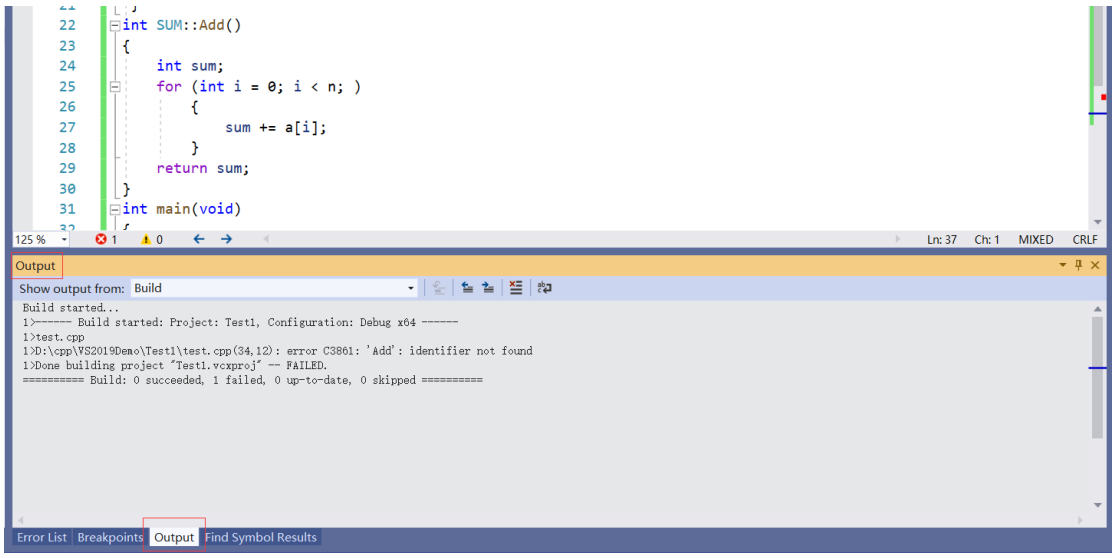

图 1-10 编译结束后的输出窗口

为了方便查看程序的编译错误，可通过菜单“视图”/“错误列表”打开错误列表窗口，如图 1-11 所示。

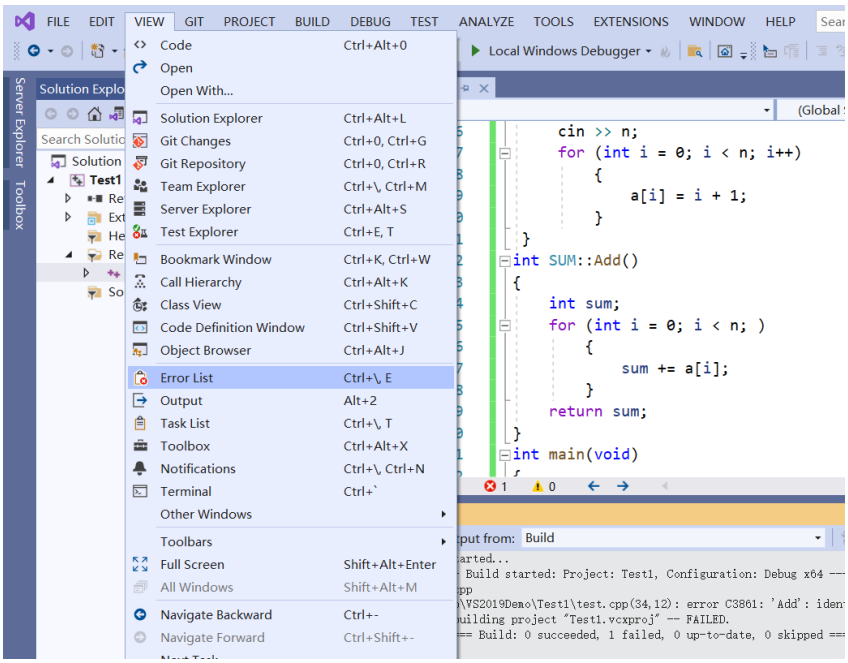

图 1-11 打开错误列表

随后，单击“错误列表”标签，查看错误列表窗口，如图 1-12 所示。

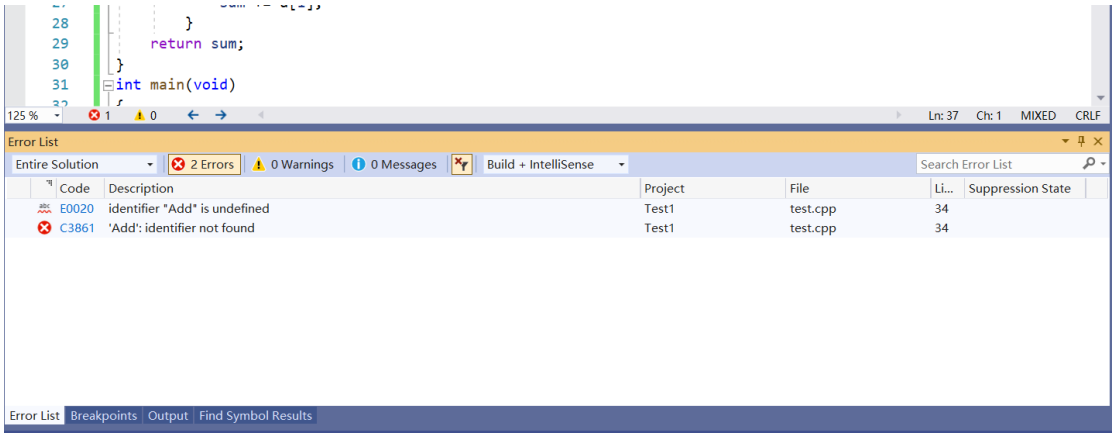

图 1-12 VS 中显示编译错误的提示信息

窗口中的消息区内列出了所有错误和警告及其发生的位置与可能原因。双击错误提示信息，光标会立刻跳转到发生错误的代码行。提示：如果警告数量太多，影响查看，单击图 1-12 中的“警告 0”按钮，则可关闭或打开警告信息。类似地，也可以单击“错误 2”按钮来关闭或打开错误信息。

根据错误列表中第 1 行的错误提示信息可知，第 34 行的方法名 Add 前缺少对象名“s.”，即第 34 行代码应修改为：

```
int sum = s.Add();
```

此时，重新编译程序，VS 的输出窗口会显示如图 1-13 所示的错误信息“使用了未初始化的局部变量 sum”，双击该错误，会在第 27 行代码前面出现一个小横线，审查第 27 行语句发现该行语句在实现累加之前，sum 变量未被初始化。

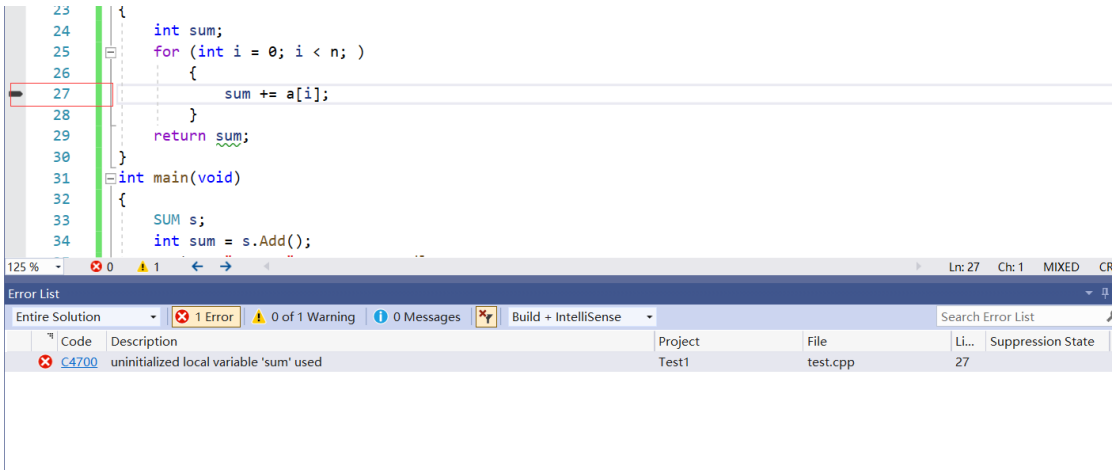

图 1-13 修改错误后重新编译程序时显示的编译错误提示信息

将第 24 行语句修改为：

```
int sum = 0;
```

此时，重新编译程序，VS 的输出窗口会显示如图 1-14 所示的成功信息。

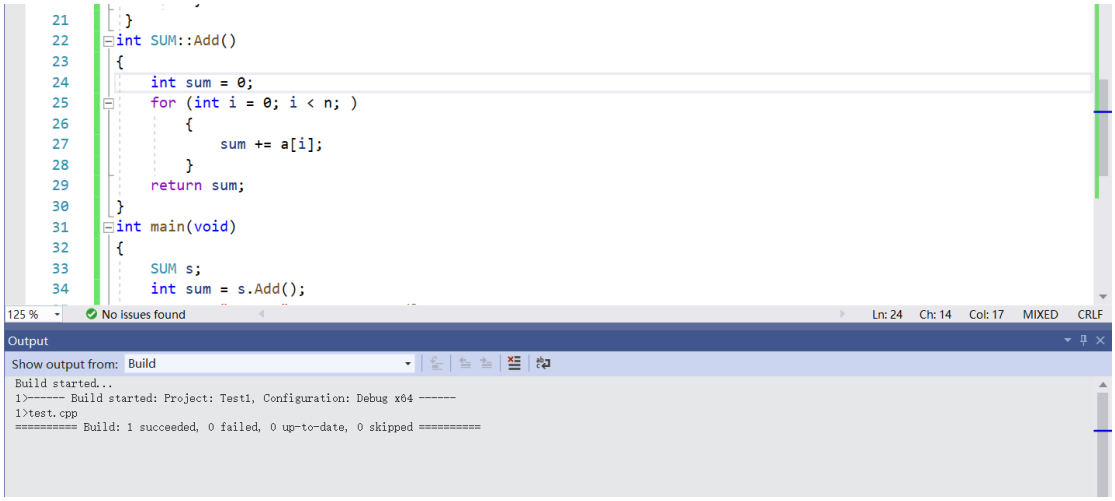

图 1-14 编译成功时 VS 输出窗口的内容

单击工具栏中的“本地 Windows 调试器”按钮或按 Ctrl + F5 键，可以运行程序。程序运行后，会弹出一个黑色的控制台窗口，输入 *n* 的值为 5 后，光标闪烁，没有任何输出，且程序执行一直不结束，如图 1-15 所示，这说明程序出现了死循环。下面，我们将介绍如何用单步调试的方法来定位这个错误。

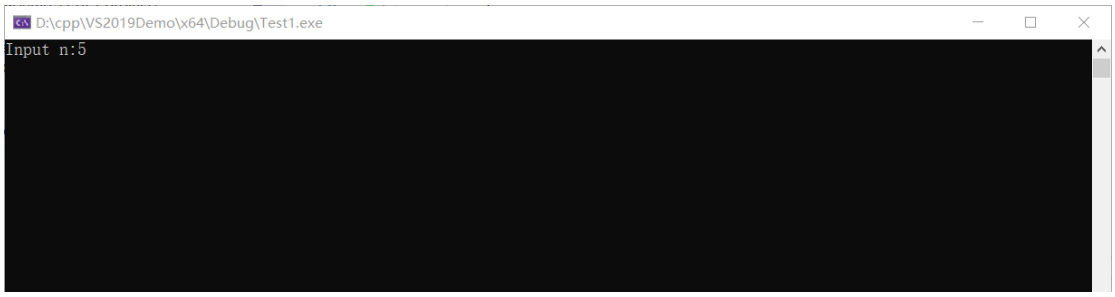

图 1-15 VS2019 中 C++程序的运行结果

3. 调试程序

若要定位程序中的错误所在行，即找到错误根源，就要用到之前提及的调试方法。在默认情况下，VS 中的程序都是采用调试模式进行编译的。如图 1-16 所示，单击 Debug 右侧的下三角按钮图标，在弹出的下拉框中有 Debug 和 Release 两个“解决方案配置类型”选项。每个选项代表 VS 工作模式的一系列参

数设定，这些设定是 VS 默认的。用户可以根据需要通过“菜单”/“项目”/“属性”进行设定的修改和调整。Debug 表示将 VS 设定为调试程序的工作模式，该模式下生成的编译结果包含调试信息，可以方便地调试程序，但程序运行速度慢。Release 模式会在程序编译过程中对代码进行速度或者大小的优化。程序优化后生成的可执行文件尽管在功能和计算结果上不会改变，但与源程序的代码往往不一致，也没有调试信息，不适合调试程序。因此，在程序开发阶段，需要频繁调试的时候，通常使用 Debug 模式编译。完成调试工作后，需要将软件交付给用户时，则采用 Release 模式编译。

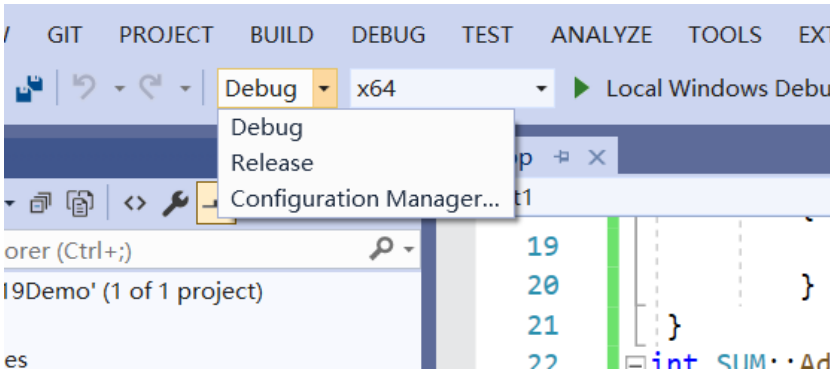

图 1-16 设置编译模式

（1）VS 中的基本调试操作

VS 中有如下几种基本调试操作。

- 按钮表示开始或继续调试，快捷键为 F5。
- 按钮表示单步进入或逐语句执行，可以跟进函数内部调试，快捷键为 F11。
- 按钮表示单步执行或逐过程执行，可以直接得到函数结果，快捷键为 F10。
- 按钮表示停止调试，快捷键为 Shift + F5。
- 按钮表示重新开始，快捷键为 Ctrl + Shift + F5。
- 按钮表示跳出函数。

（2）设置断点

若希望程序执行完第一条可执行语句（即例 1.1 中的第 33 行）后暂停，则可采用以下任意一种方法。

1）将代码的光标移至第 34 行，按 F9 键，该行代码左侧会出现一个红色的圆点，标志着设置断点成功。直接在左侧深灰色一栏第 34 行的位置单击鼠标左键也可设置断点，设置后如图 1-17 所示。

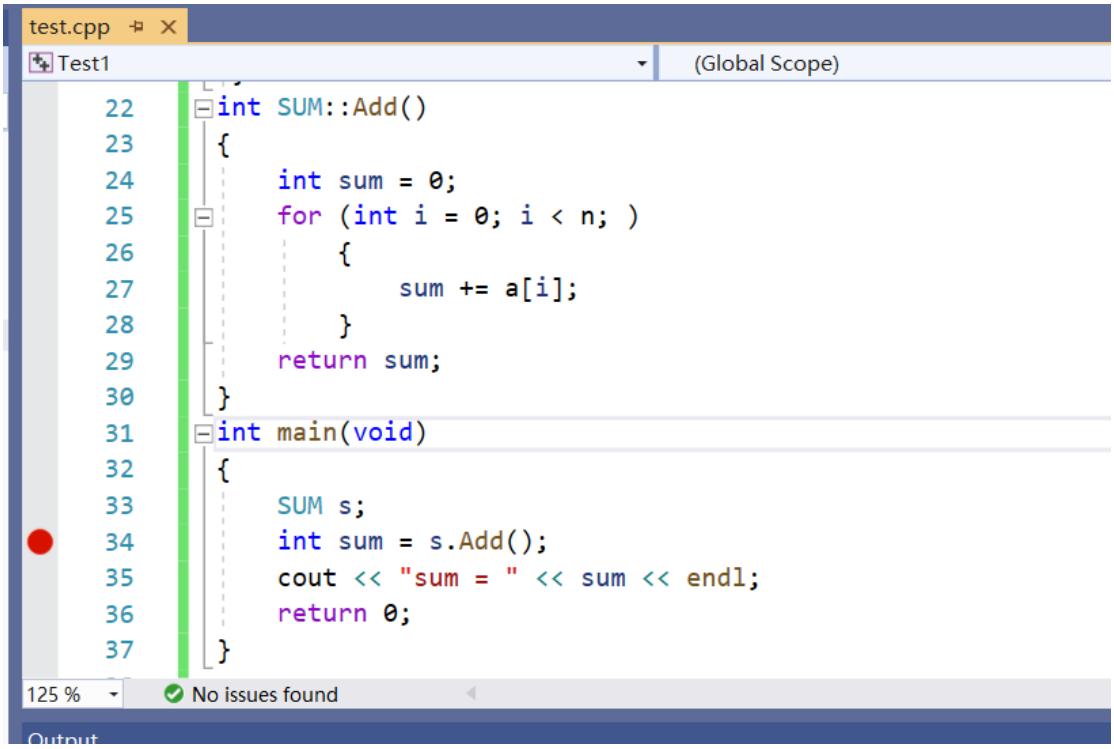

图 1-17 在需暂停的行上设置断点

2）按 F5 键开始调试该程序，程序遇到断点就暂停，进入跟踪状态，如图 1-18 所示。需要注意的是，当程序在断点处暂停时，断点所在的代码行并未执行。

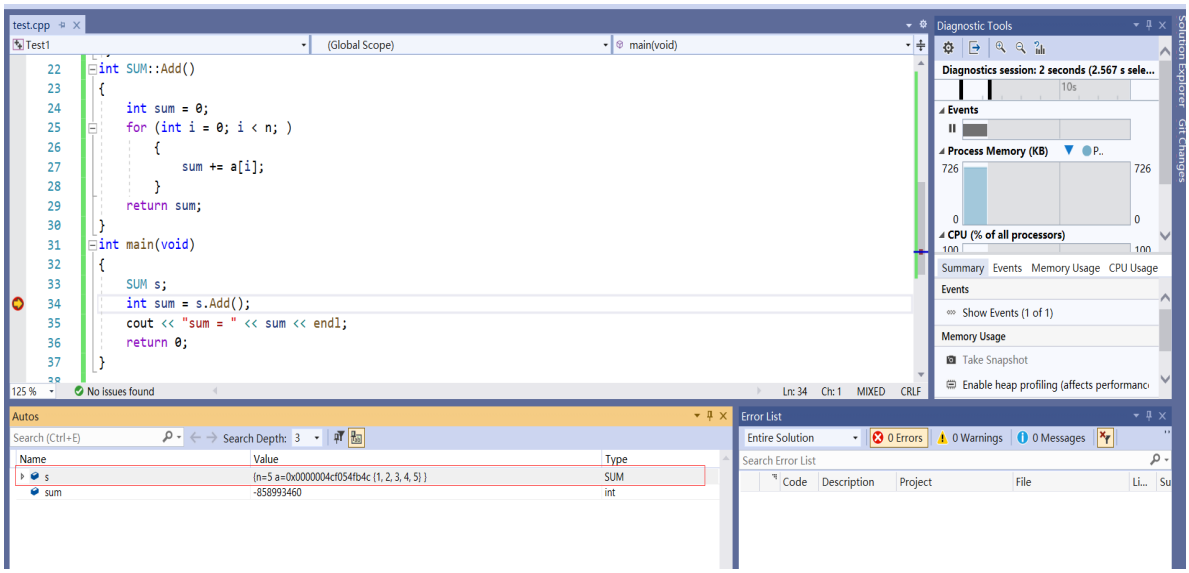

图 1-18 在需要暂停的行上设置断点后调试运行的状态

（3）在监视窗中观察变量值

程序在断点位置暂停时，该行的语句尚未执行，它是程序下一条要执行的语句。从图 1-17 可见，程序在第 10 行暂停，但是下面局部变量窗口中的 sum 值仍是-858993460。原因是第 10 行的程序尚未执行，sum 还未被赋值。程序中尚未被赋值的变量的数值可能是一个和编译器有关的随机值，在这里是-858993460。

如图 1-17 所示，在中断程序后，VS 中有如下监视窗口。

- 自动窗口：显示在当前代码行和前面代码行中使用的变量，以及函数返回值（如果有的话）。
- 局部变量窗口：显示对于当前上下文（通常是当前正在执行的函数）来说位于本地的变量。
- 监视窗口：可以添加需要观察的变量。通过这几个窗口，可以在程序中断时手工修改变量的数值。在源代码窗口中，在需要监视数值的变量名上，通过单击鼠标右键选择“添加监视”，即可将该变量添加到监视窗口。

此外，还有调用堆栈、断点、异常设置、输出等窗口。

（4）单步进入，跟踪函数调用

s.Add()是方法调用语句，由于程序只有这一个调用方法，因此若要进一步分析为何执行完该方法得到的结果是错误的，就需要进入 Add 方法内部跟踪运行情况，方法如下。

1）在第 34 行设置断点，按 F5 键开始调试，如图 1-17 所示。

2）按 F11 键——单步进入（如图 1-18 所示），黄色箭头暂停在 Add 方法上。

调试程序时，首先分析出可疑函数，然后跟踪至函数内部，最后在函数内部调试。本例中，进入 Add 方法后，如图 1-19 所示，按 F10 键单步跟踪，跟踪到函数内部窗口如图 1-20 所示。观察变量监视窗口中各个变量值的变化情况，发现循环变量 *i* 的值始终为 0，经人工检查发现第 25 行的 for 语句没有在每次执行完循环后对 *i* 值进行更新，从而导致了死循环。

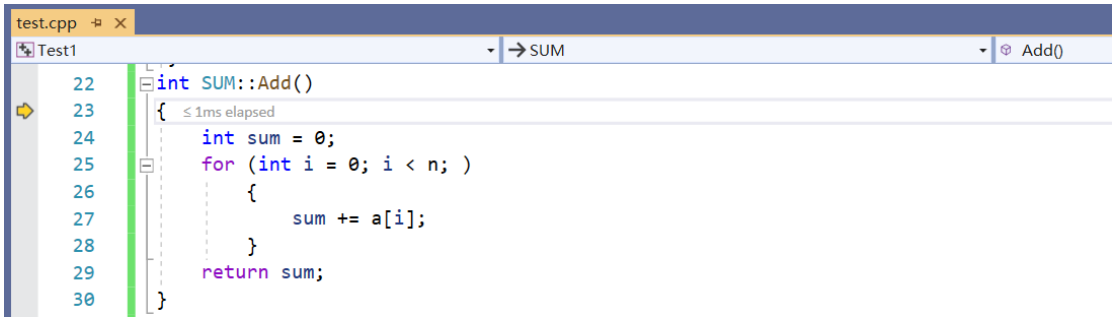

图 1-19 VS 中单步进入跟踪函数调用

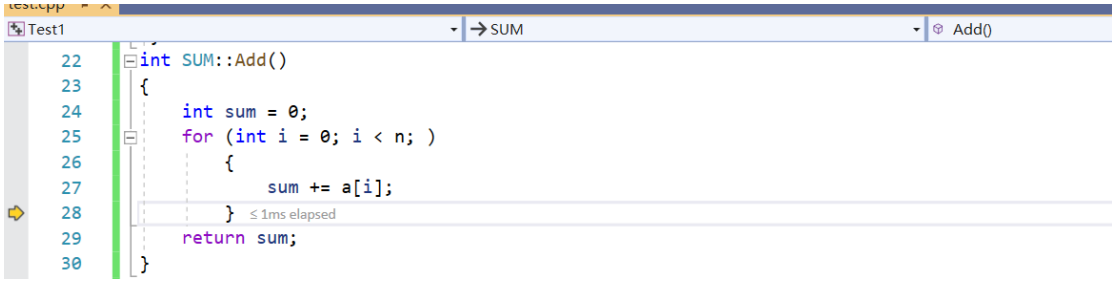

图 1-20 VS 中跟踪到函数内部调试

修改第 25 行的 for 语句为：

for (int i = 0; i < n; i++)

重新运行例 1.1，如图 1-21 所示，经确认，修改后的结果正确。

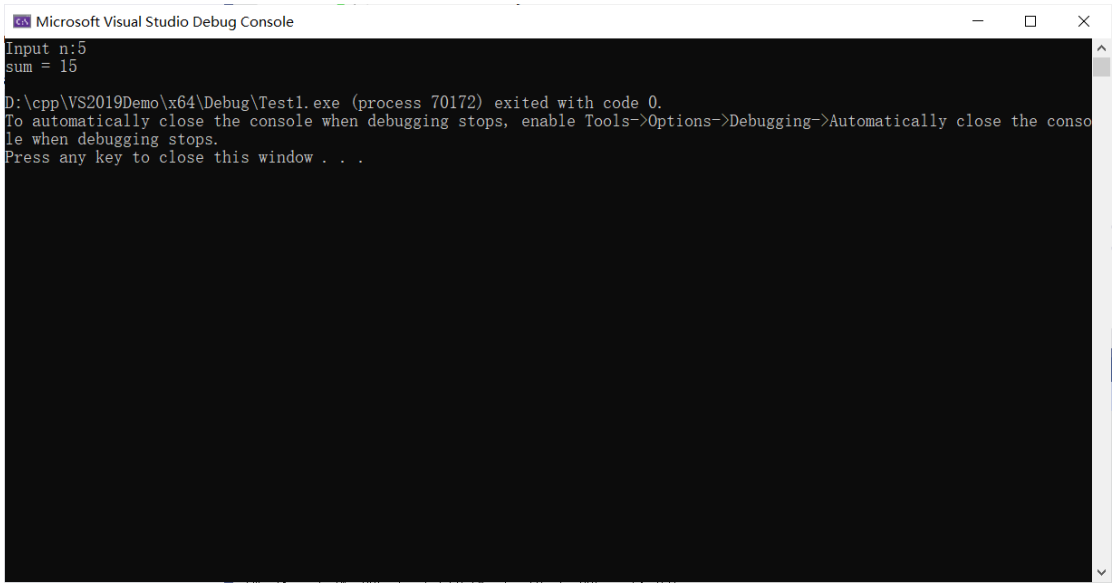

图 1-21 修改后的程序结果

（5）控制调试的步伐、条件断点

本例中 for 循环内的语句只执行 5 次，我们可以通过手动方式观察每次循环，如果有成百上千甚至上万次循环，那么逐步的执行将给调试工作带来很大困扰。幸运的是，我们可以根据需要来控制调试的进程：

1）如果希望调试工作继续运行，按 F5 键，程序会一直运行到结束或再次遇到断点。

2）如果只是想完成这个循环，那么把光标移动到循环语句之后的 return sum;这一行，按 Ctrl + F10 键即执行命令“运行到光标所在的行”，于是黄色箭头停在 return sum;语句标号处，直接得到循环后的结果。

3）如果不想逐条语句跟踪，按 Shift + F11 键就可以“运行出函数”，直接回到调用者。

4）设置条件断点。例如在第 20 行设置断点，并期望在 $i = 4$ 时断点生效，暂停程序运行。首先，将光标移动到第 20 行，按 F9 键设置断点。然后，鼠标右键单击第 20 行左端的实心圆点形的断点图标。在如图 1-22 所示的弹出菜单中有多种操作和属性设置，我们选中“条件”复选框，进入条件断点设置界面；或者，将鼠标移动到断点的红色圆点图标后，单击窗口出现的齿轮形浮动图标，进入条件断点设置界面。

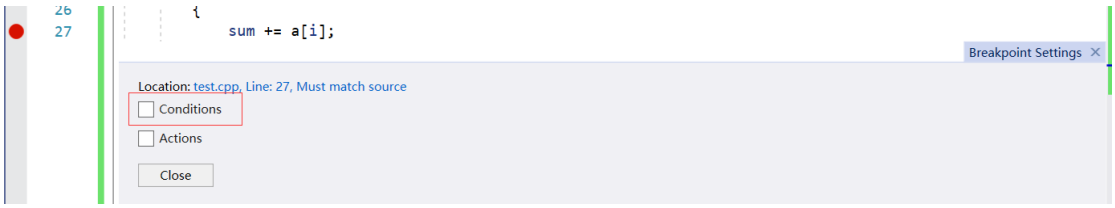

图 1-22 设置断点属性

条件断点设置界面如图 1-23 所示，在窗口中输入条件“i == 4”，条件值设定为默认值 true，单击“关闭”按钮完成条件设定。此时，断点图标从红色的实心圆点变成带有白色加号的红色实心圆点。

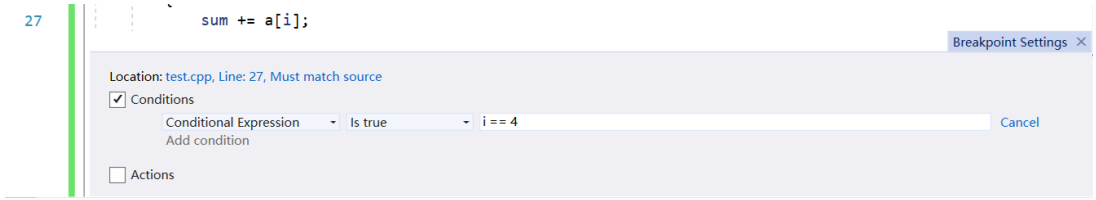

图 1-23 设置条件断点

然后，重新调试运行，程序在第 27 行的条件断点处暂停时的情况如图 1-24 所示：$i=4$ 时 sum 为 10。

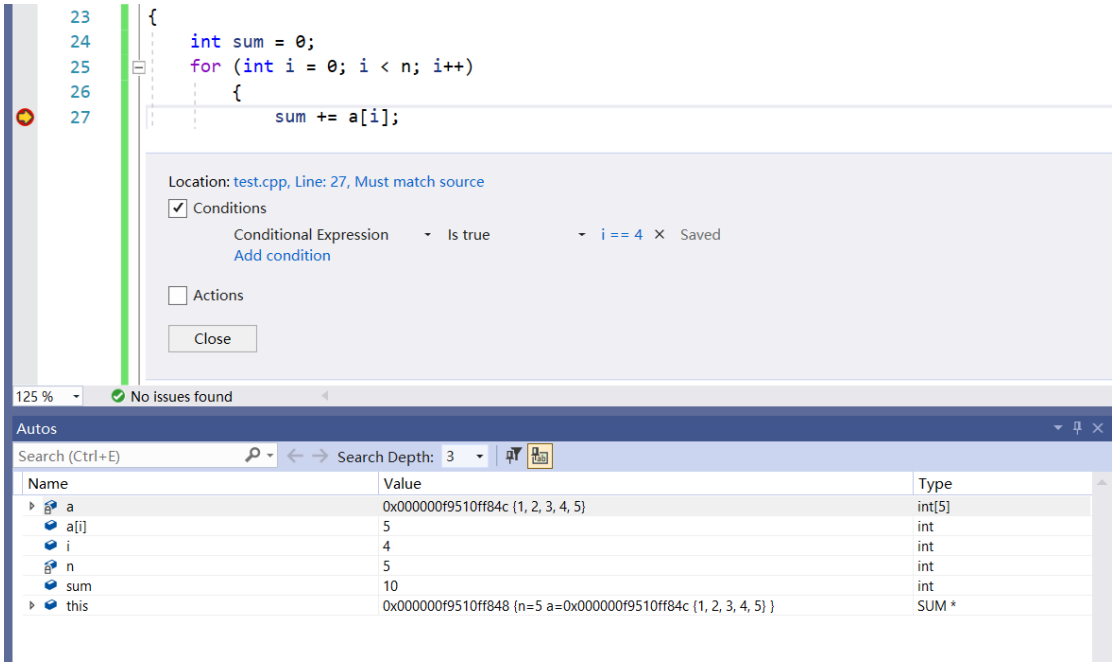

图 1-24 在条件断点处暂停时的结果

4. main 函数带参数的 C++程序的调试

在以上调试的程序中，main 函数是无参的，下面通过例 1.2 介绍如何在 VS 中调试 main 函数带参数的程序。

【例 1.2】带参数的程序实例。

```
1 #include <iostream>
2 #include <cstdlib>
3 using namespace std;
4 int main(int argc, char* argv[])
5 {
6       cout << argc << endl;
7       cout << argv[1] << endl;
8       cout << argv[2] << endl;
9       cout << argv[3] << endl;
10      return 0;
```

11 }

在 main 函数的参数列表中，argc 表示命令行参数的个数，argv 表示传入的参数，命令行参数以字符串的形式存储在 argv 中。在调试时，需要设计 Debug 参数将参数传入 main 函数。

首先，在现有解决方案 VS2019Demo 中添加新的项目 Test2：右键单击“解决方案 VS2019Demo”/“添加”/“新建项目”，如图 1-25 所示。

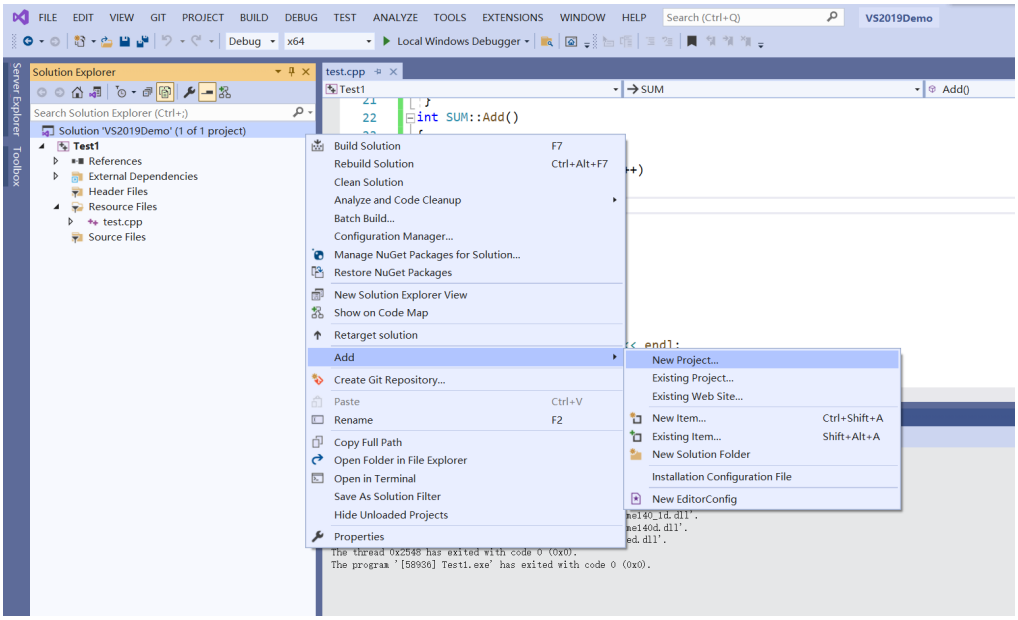

图 1-25 VS 中在原有解决方案上添加新项目

后续步骤和图 1-3、图 1-4 所示相似，不用设定解决方案名称和存储位置（之前已经设定过了），只需要选择“空项目”并指定项目名称为 Test2 即可。添加完工程 Test2 后，我们在操作系统的资源管理器中可以看到，解决方案 VS2019Demo 的文件夹中有两个子文件夹 Test1 和 Test2。

在 VS 的解决方案管理窗口内，在工程名 Test2 下的“源文件”上单击鼠标右键后，选择“添加”/“新建项”，用与图 1-6 类似的方法添加源代码文件 test2.cpp，并将例 1.2 的源代码完整输入。

若想调试运行解决方案 VS2019Demo 中的某一个项目，需要在解决方案资源管理器中，在该项目名称上单击鼠标右键，在弹出菜单中选择“设为启动项目”，否则运行的是之前的项目。现在，我们将 Test2 设定为启动项。

在 VS 中调试时，首先选择“项目”/“属性”菜单项，打开属性页，如图 1-26 所示。

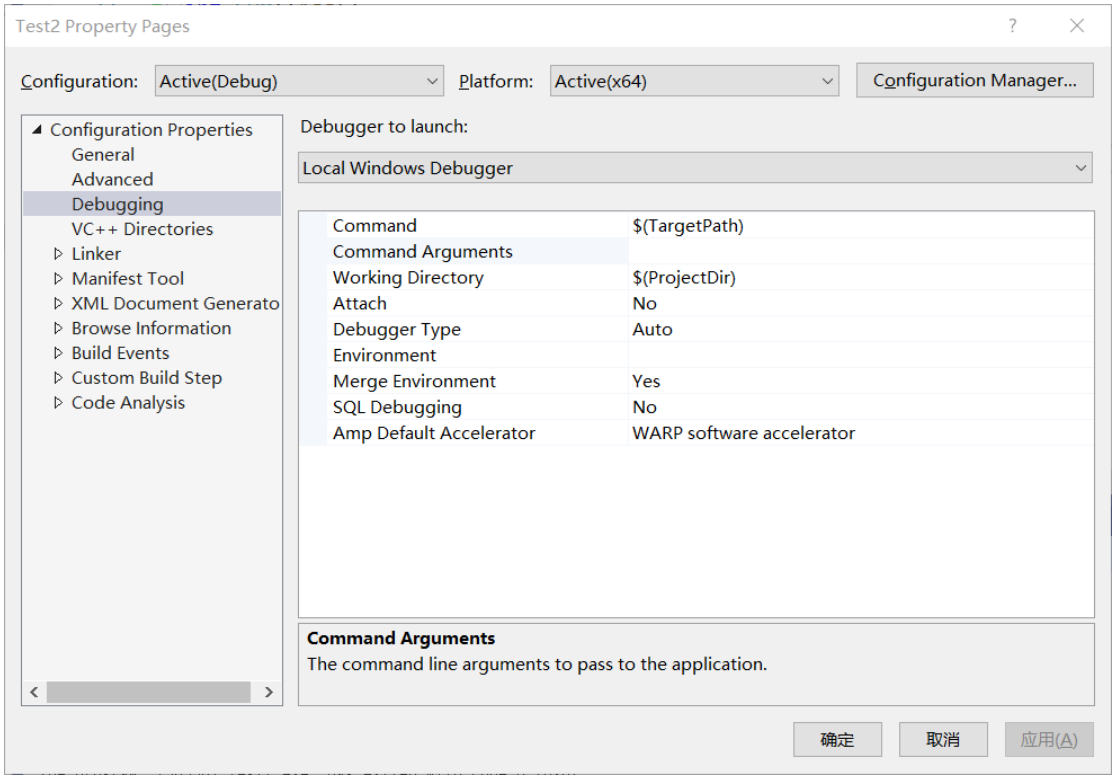

图 1-26 在 VS 中调试 main 函数带参数的程序

然后，在弹出的窗口中，在“调试”/“命令参数”对话框中输入各个参数，如图 1-27 所示，此处输入了三个用空格分开的参数值 1、2、3。单击“确定”按钮之后，调试就和普通程序一样，命令行参数会随着程序的调试运行而被自动输入。

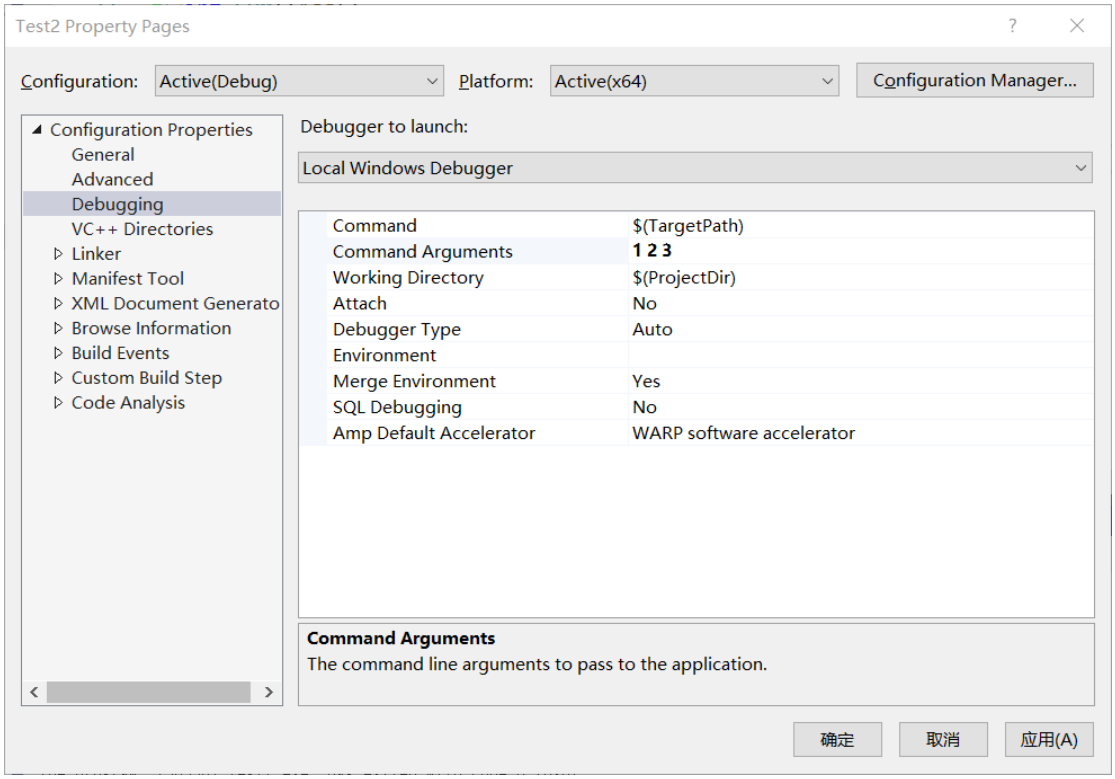

图 1-27 在 VS 中输入 main 函数命令行参数的值

在调试过程中，如果想终止程序的调试，操作方法为：单击“菜单”/“调试”/“停止调试”或按 Shift + F5 键。

1.3 多文件项目的开发

在例 1.1 的代码中，类的定义和实现以及主函数 main 都在一个.cpp 文件中。当程序规模比较大时，往往需要将类的定义和实现、源代码分别保存在不同类型的多个文件中。本节介绍多文件项目的开发和调试方法。

1.3.1 Visual Studio 下的多文件项目开发

1. 创建项目并添加文件

使用在 1.2.1 节中图 1-25 所介绍的方法，在解决方案 VS2019Demo 中增加第三个项目 Test3，并在该项目中添加如下不同类型的文件。

（1）添加.h 文件：sum.h

首先，在解决方案资源管理器中的项目 Test3 上单击鼠标右键，如图 1-69 所示，在弹出的菜单中选择：“添加”/“新建项”。之后会显示如图 1-70 所示的界面，在其中选择“头文件(.h)”，输入名称 sum.h 即可完成添加 sum.h 的操作。

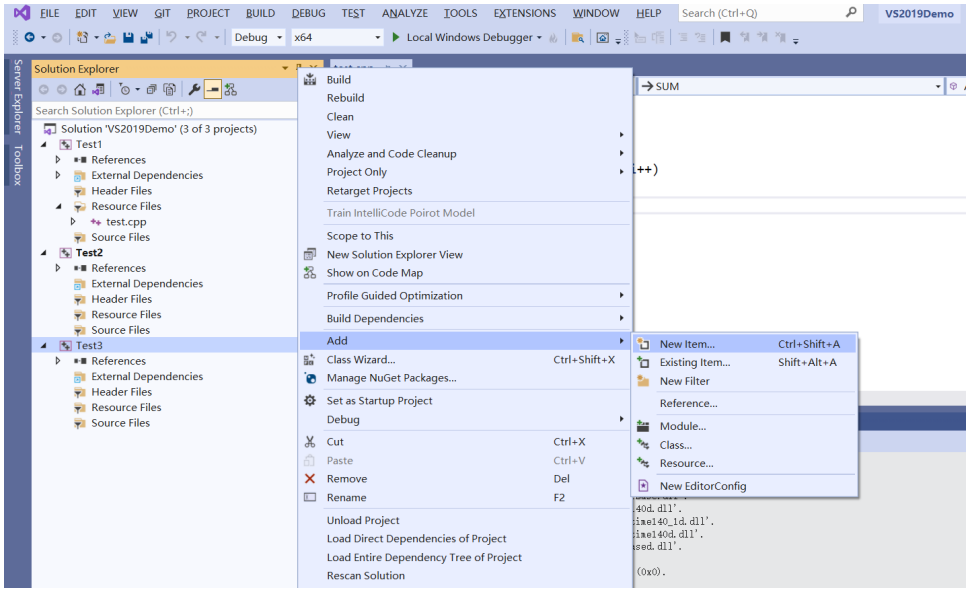

图 1-69 在项目中添加新建项的界面

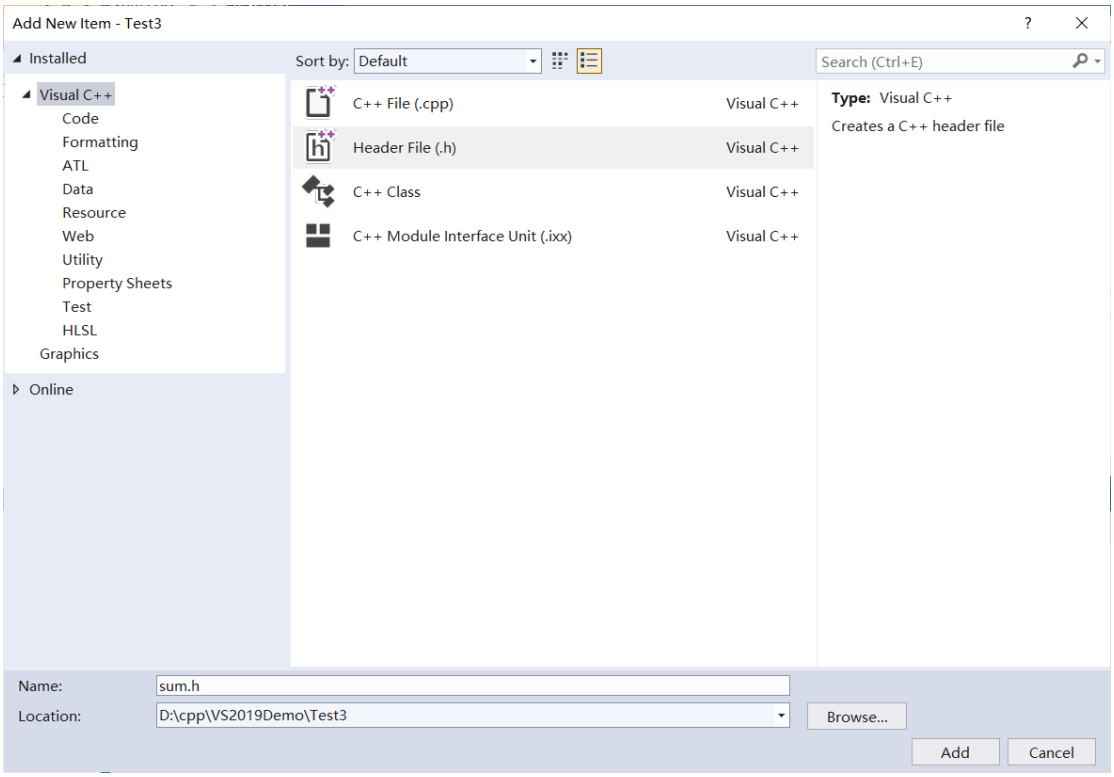

图 1-70 添加.h 头文件的界面

编辑 sum.h 文件，该文件内容如下：

```
class SUM
{
    public:
        SUM();
        static const int N = 5;
        int Add();
    private:
        int n;
        int a[N];
};
```

这里尽可能将所有类的定义以及宏定义放到一个文件中，在需要使用类定义和宏定义的代码文件（.h、.cpp）中，只要增加该头文件的包含语句即可，形式如下：

#include "sum.h"

在大型项目中，不同的文件之间的包含关系复杂，最终可能导致一个代码文件直接或间接包含了某个.h 文件很多次（超过 1 次），进而导致编译错误：类或宏的重复定义。为了避免同一个头文件被包含多次，C/C++中规定了两种宏实现方式：一种是#ifndef 方式，另一种是#pragma once 方式。#ifndef 得到 C/C++语言标准的支持，不受编译器的任何限制。但一些较老版本的编译器可能不支持#pragma once 方式，因为其兼容性可能不够好。第一种方式是使用条件编译语句 ifndef… #else… #endif 将.h 文件的全部内容括起来：

```
#ifndef    HIT_C_EXAMPLE_SUM_H
#define    HIT_C_EXAMPLE_SUM_H
class SUM
{
  public:
    SUM();
    static const int N = 5;
    int Add();
  private:
    int n;
    int a[N];
};
#endif
```

添加了条件编译语句后，相当于告知编译器：如果没有定义过宏 HIT_C_EXAMPLE_SUM_H，则先定义该宏，再定义类。如果已经定义了 HIT_C_EXAMPLE_SUM_H，#ifndef 条件不成立，则该文件的内容相当于空文件，也就不会导致重复定义类了。这里，要保证 HIT_C_EXAMPLE_SUM_H 不会和别的代码文件中的宏名字重复。因此，通常采用这种较长的命名形式，甚至采用在末尾添加随机数串的形式，以确保名称的唯一性。

（2）添加.cpp 文件：sum.cpp

sum.cpp 内容如下：

```
#include <iostream>
#include "sum.h"
using namespace std;
SUM::SUM()
{
    cout << "Input n:";
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        a[i] = i + 1;
    }
}
int SUM::Add()
{
    int sum = 0;
    for (int i = 0; i < n; i++)
    {
        sum += a[i];
    }
    return sum;
}
```

该源代码文件负责实现 sum.h 中声明的类中定义的方法。

在 C++语言中使用#include 命令的两种方式如下：

- 一种是在包含指令#include 后面用<>将头文件名括起来。这种方式用于将 C 标准库函数所需的头文件包含到当前文件中，这样系统会自动到保存标准库函数头文件的文件夹下查找该头文件。

- 另一种是在包含指令#include 后用双引号""将头文件括起来。使用这种格式时，编译器先查找当前目录是否有指定名称的头文件，再从标准头文件目录中查找。这种方式常用于程序员自己定义的头文件。

（3）添加.cpp 文件：test.cpp

test.cpp 文件为主程序源代码文件，内容如下：

```
#include <iostream>
#include "sum.h"
using namespace std;
int main(void)
{
    SUM s;
    int sum = s.Add();
    cout << "sum = " << sum << endl;
    return 0;
}
```

主程序中调用了 SUM 类中的方法 Add()，因此需要在 test.cpp 中包含它们的定义.h 文件。完成文件添加后，在项目 Test3 的文件夹内可看到刚刚添加的.h 文件和.cpp 文件，如图 1-71 所示。

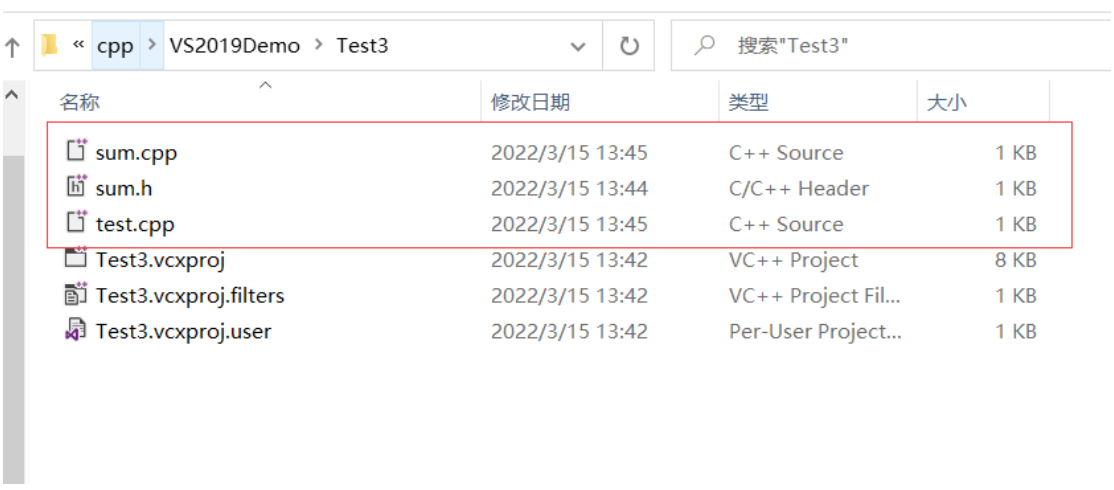

图 1-71 Test3 的程序文件（高亮显示的文件是代码文件）

除上述方法外，也可以用文本编辑器逐个创建并编辑这些代码文件，将其保存成纯文本格式的文件。最后，在 VS 中可用如下方法将这些文件添加到项目中：鼠标右键单击项目名称 Test3/“添加”/“现有项”，在之后弹出的窗口中用 Ctrl+鼠标左键点选要添加的文件，之后单击“添加”按钮即可，一次可以添加一个或多个文件。

2. 编译和调试

由于项目包含多个文件，因此在编程和调试过程中会涉及多个文件的修改。在程序编译时，仅对修改的文件进行编译，可以节省编译时间。但在项目规模不算庞大（例如代码行数不到一万行）时，完全重新编译的速度也比较快。如果仅编译修改的文件，在调试运行时，VS 有时会提醒项目过期或者出现断点无法停止的问题，此时需要重新完整编译。因此，在多文件项目编译时，推荐采用“重新生成”的方法对整个项目进行完整的编译。

在解决方案资源管理器中的项目 Test3 上单击鼠标右键，选择“重新生成”，如图 1-72 所示；或者如图 1-73 所示，先在解决方案资源管理器中将预编译的项目设定为启动项目：在项目名称 Test3 上单击鼠标右键，在弹出的菜单上选择“设为启动项目”，然后在 VS 的主菜单中单击“生成”/“重新生成解决方案”。

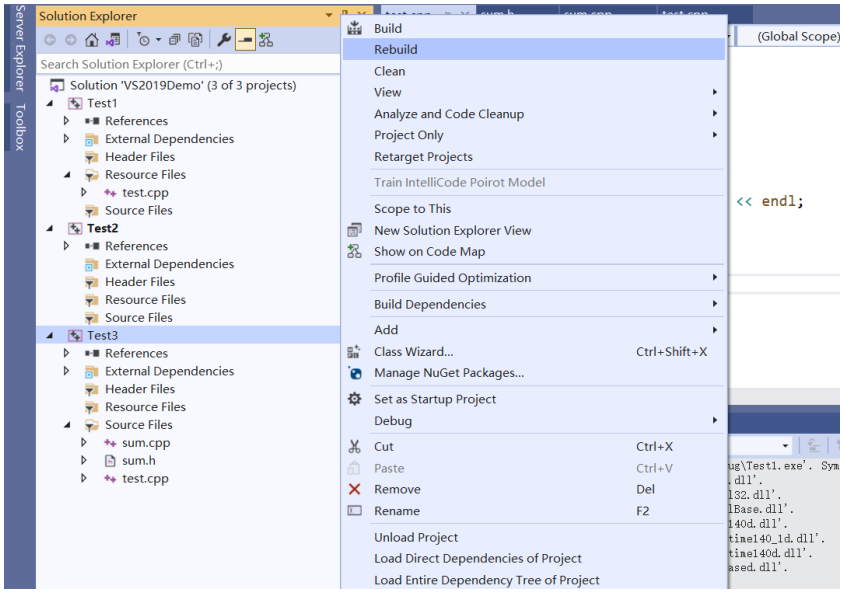

图 1-72 编译项目的方法：在项目名称上单击鼠标右键菜单

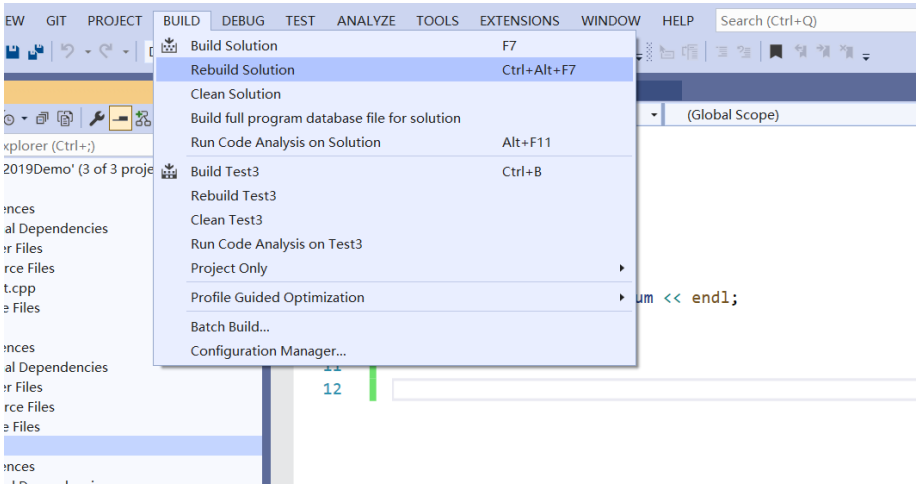

图 1-73 编译项目的方法：主菜单“生成”

多文件项目的调试过程和基本方法与普通单文件项目并无区别，只是在调试过程中，需要格外注意变量、函数、类或宏的重复定义、外部声明问题。