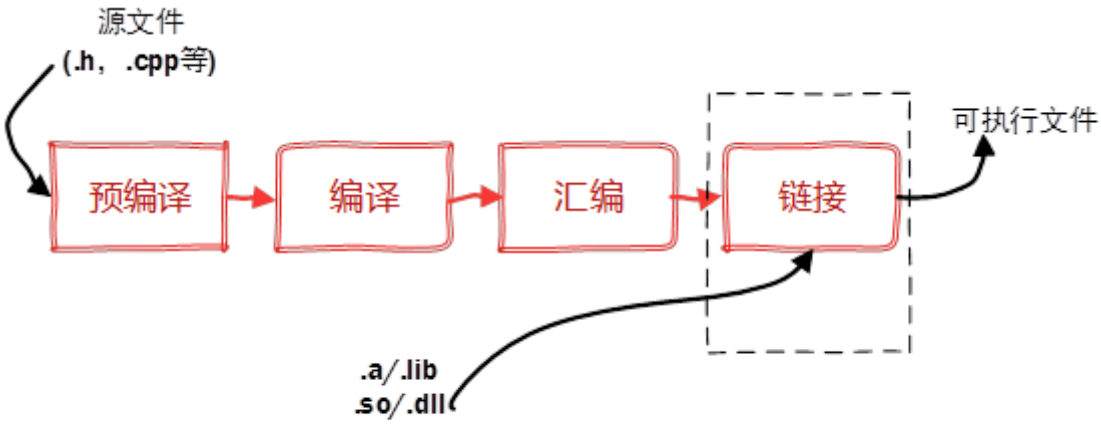
C语言——面向过程编程

单元七、承前启后

第一次课 组件化打包技术——库

一、“库”是什么

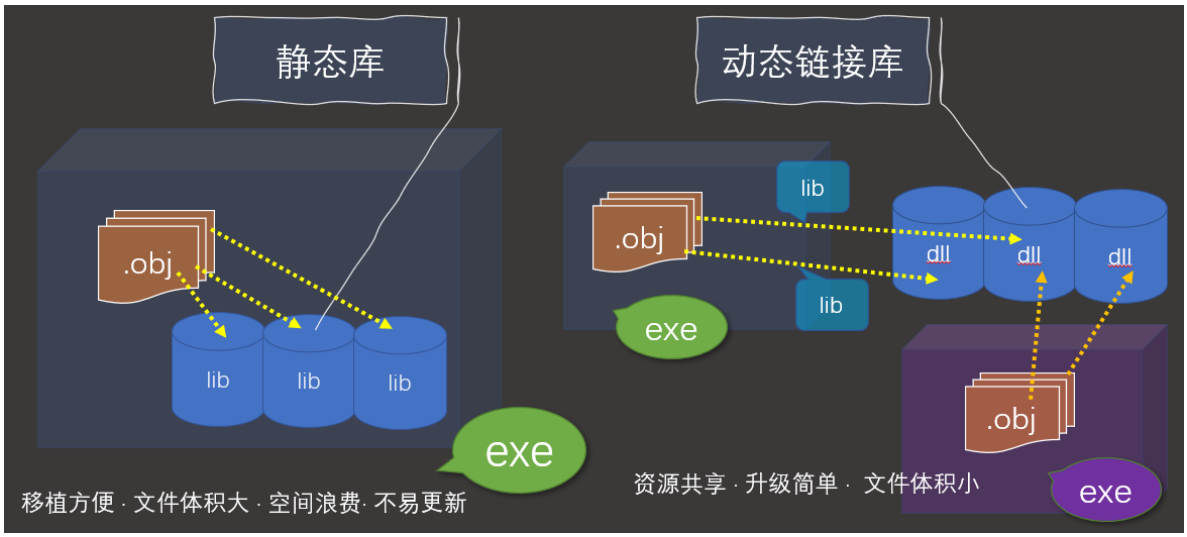
库是写好的，现有的，成熟的，可以复用的代码。**现实中每个程序都要依赖很多基础的底层库，不可能每个人的代码都从零开始，因此库的存在意义非同寻常。**本质上来说，库是一种可执行代码的二进制形式，可以被操作系统载入内存执行。库有两种：静态库（.a、.lib）和动态库（.so、.dll）。所谓静态、动态是指链接。回顾一下，将一个程序编译成可执行程序的步骤：



静态库、动态库区别来自【链接阶段】如何处理库，链接成可执行程序。分别称为静态链接方式、动态链接方式。

像我们平时使用的三方库：opencv、sdl、ffmeng、网络封装等，都属于库封装的方式，我们只能使用库里面函数的功能，但是看不到库里面函数的实现。在工作中，经常使用封装库的方式于其他部门一起进行软件开发。当然了，咱们小组一起开发项目就不用这样了。

二 、 静态库lib与动态链接库dll的区别

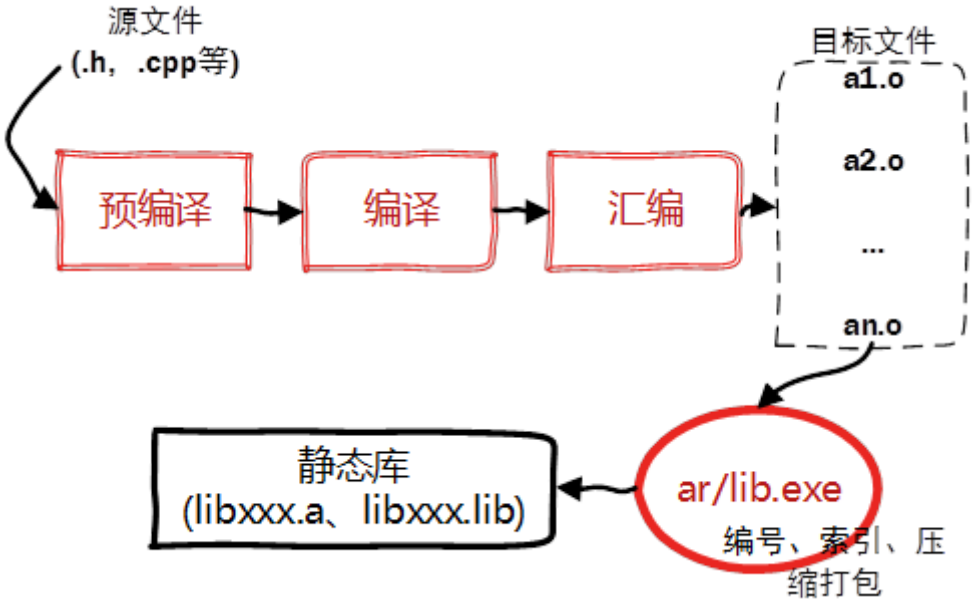


什么是**静态库**：之所以称为静态库，是因为在链接阶段，会将汇编生成的目标文件.o与引用到的库一起链接打包到可执行文件中。因此对应的链接方式称为静态链接。

试想一下，静态库与汇编生成的目标文件一起链接为可执行文件，那么静态库必定跟.o文件格式相似。其实一个静态库可以简单看成是**一组目标文件（.o/.obj文件）的集合**，即很多目标文件经过压缩打包后形成的一个文件。静态库特点总结如下：

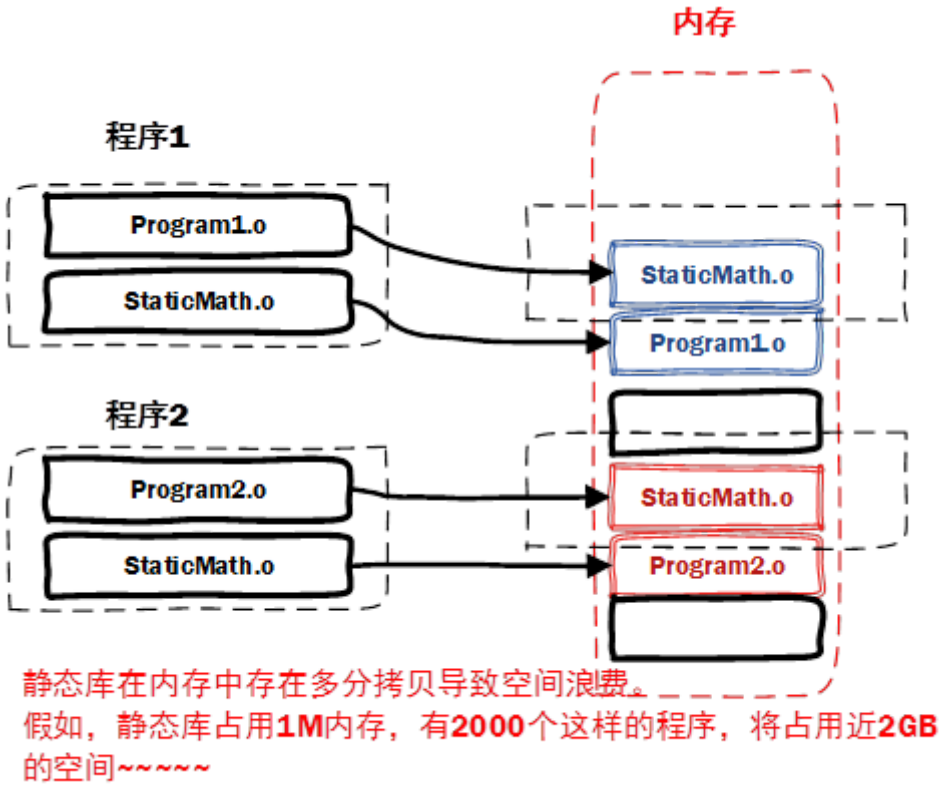
- 静态库对函数库的链接是放在编译时期完成的。
- 程序在运行时与函数库再无瓜葛，移植方便。
- 浪费空间和资源，因为所有相关的目标文件与牵涉到的函数库被链接合成一个可执行文件。

  Windows下vs使用lib.exe，将目标文件压缩到一起，并且对其进行编号和索引，以便于查找和检索。一般创建静态库的步骤如图所示：



为什么还需要**动态链接库**，其实也就是静态库的特点导致

- 空间浪费是静态库的一个问题。

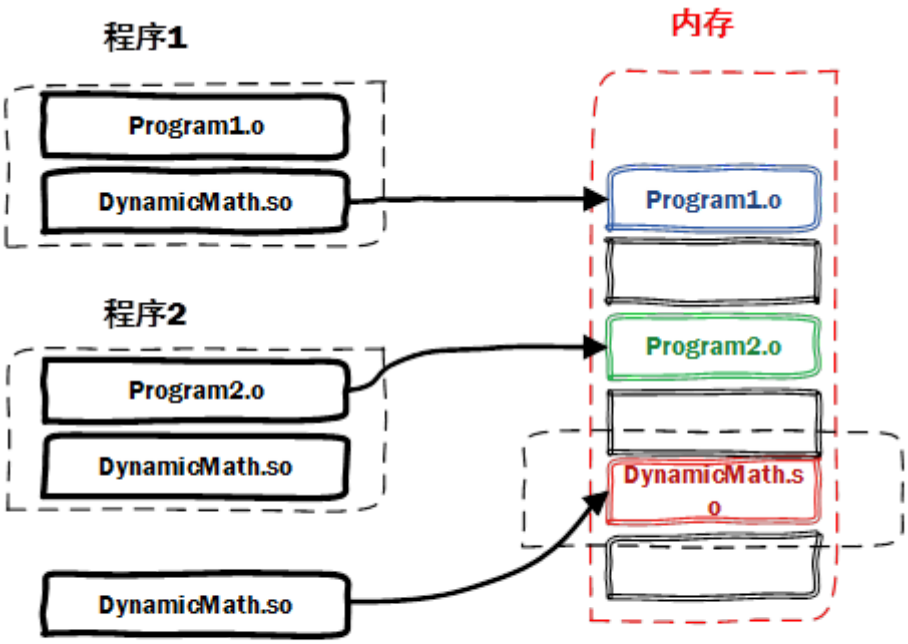


- 另一个问题是静态库对程序的更新、部署和发布页会带来麻烦。如果静态库libxx.lib更新了，所有使用它的应用程序都需要重新编译、发布给用户（对于玩家来说，只是一个很小的改动，却导致整个程序重新下载，全量更新）。

动态库在程序编译时并不会被连接到目标代码中，而是在程序运行是才被载入。不同的应用程序如果调用相同的库，那么在内存里只需要有一份该共享库的实现，规避了空间浪费问题。动态库在程序运行时才被载入，也解决了静态库对程序的更新、部署和发布页会带来麻烦。用户只需要更新动态库即可，增量更新。

动态库特点总结：

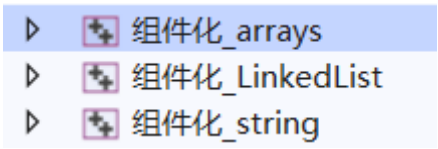
- 动态库把对一些库函数的链接载入推迟到程序运行的时期。
- 可以实现进程之间的资源共享。（因此动态库也称为共享库）
- 将一些程序升级变得简单。
- 甚至可以真正做到链接载入完全由程序员在程序代码中控制（**显示调用**）。



动态库在内存中只存在一份拷贝，避免了静态库浪费空间的问题。

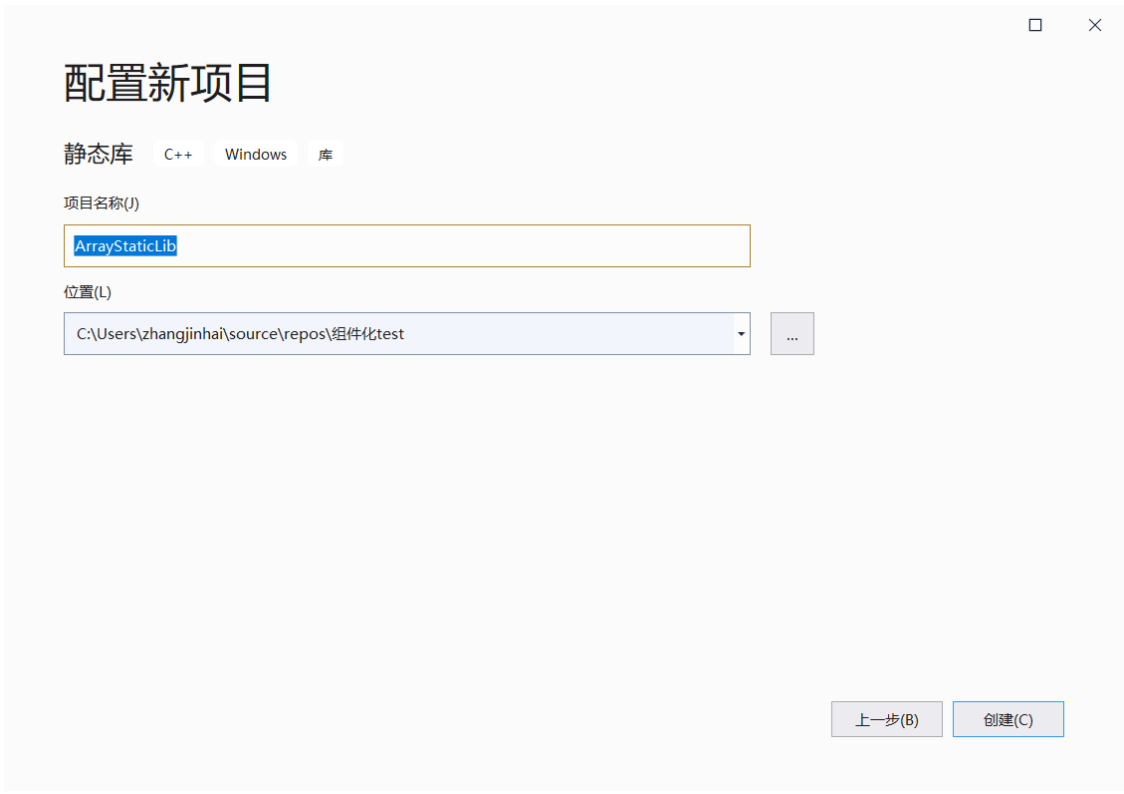
三 、 静态库lib创建过程

1. **把已经测试完成的组件化项目及代码（头文件及对应的源代码文件）准备好**

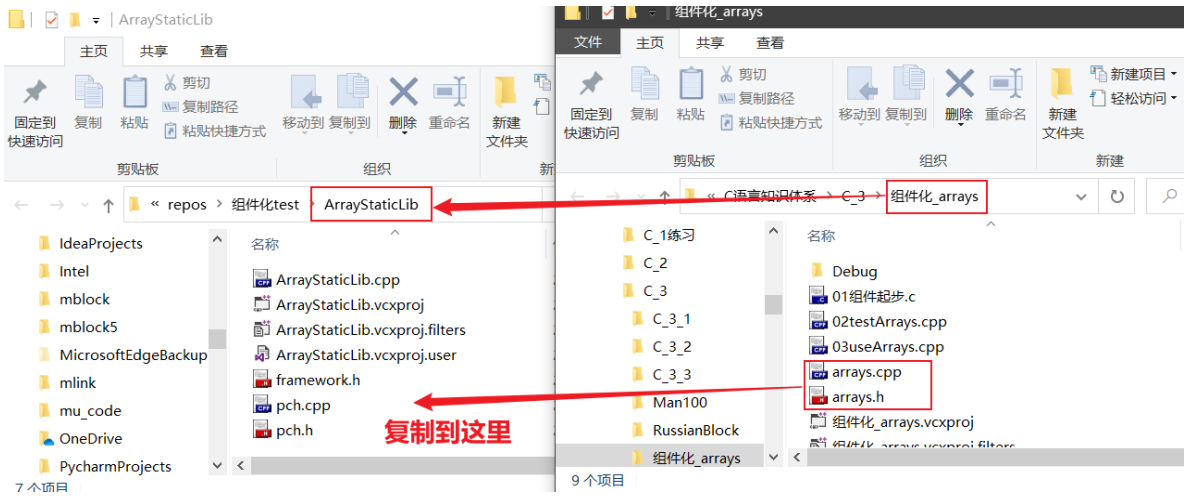


---

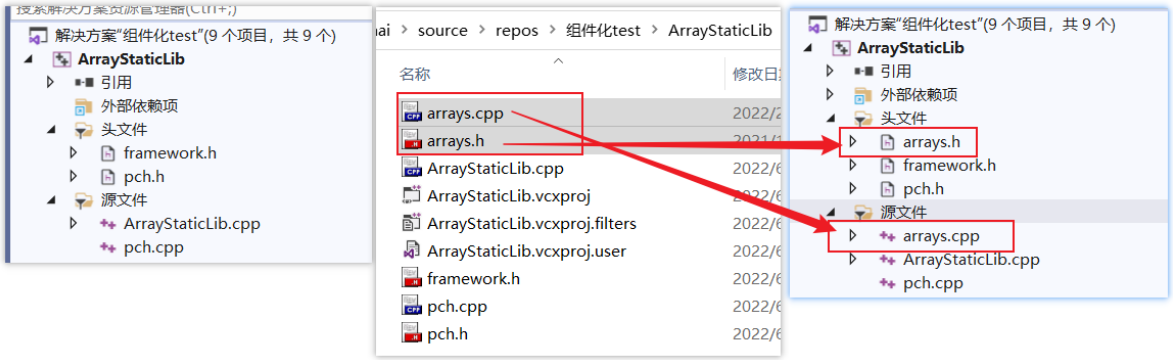
2. 在"**解决方案**上"新建项目——>选择：静态库



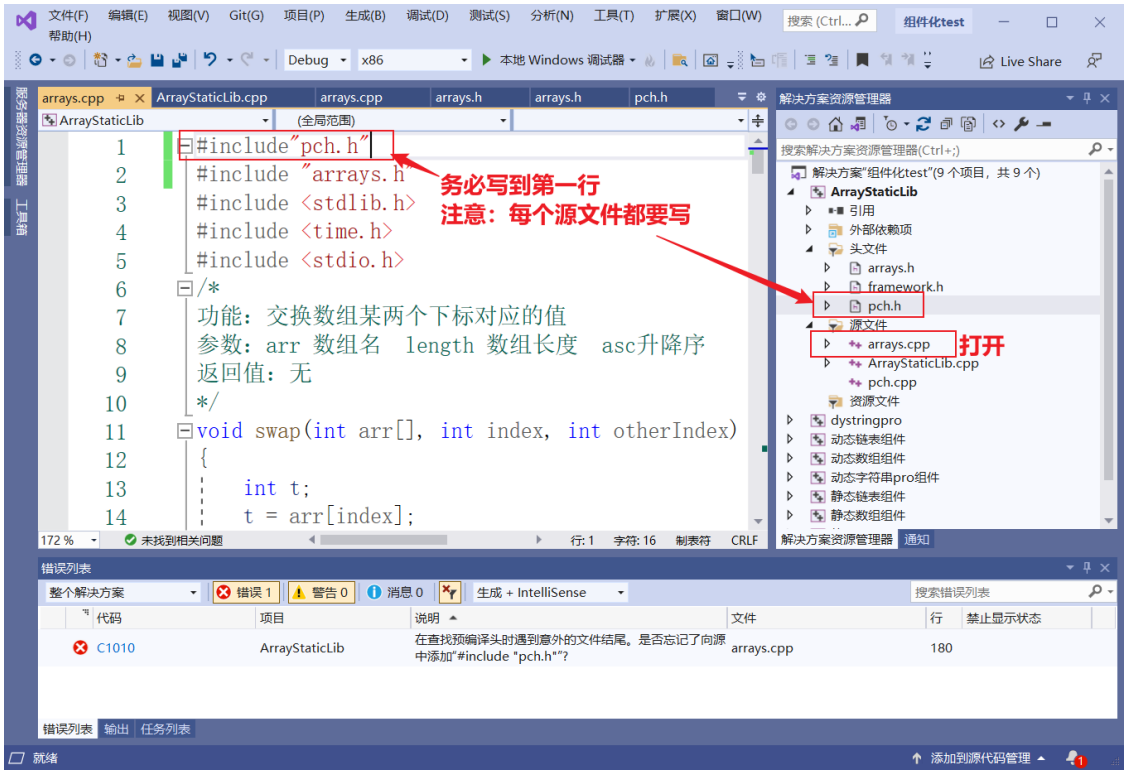
3. 把《组件化代码》复制到《静态库项目——ArrayStaticLib》所在文件夹



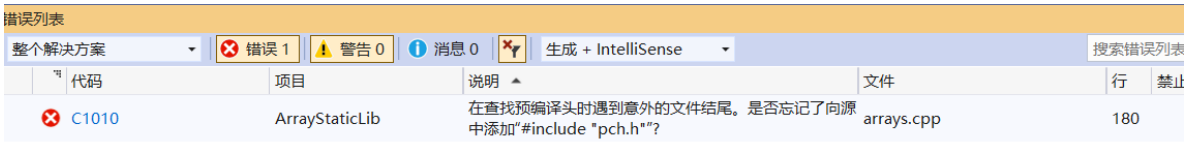
并把新复制进来的两个文件加入到《静态库项目——ArrayStaticLib》的指定位置



4. 把《组件化》的源代码文件第一行加入#include “pch.h”

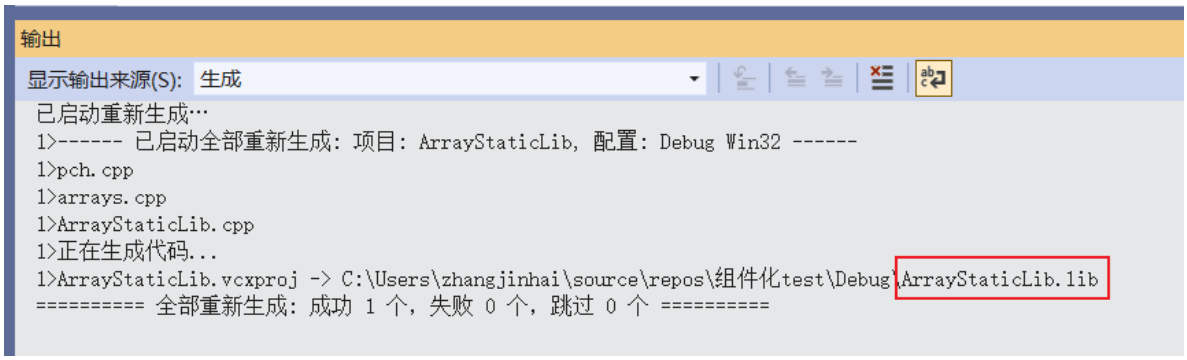


如果不加入在生成时会产生如下错误提示：

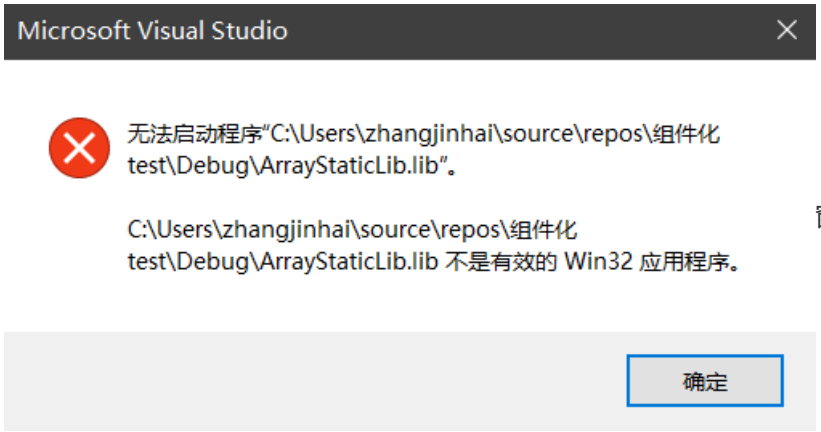


5. 生成静态库文件lib

如产生如下信息，则说明静态库生成成功。



如产生

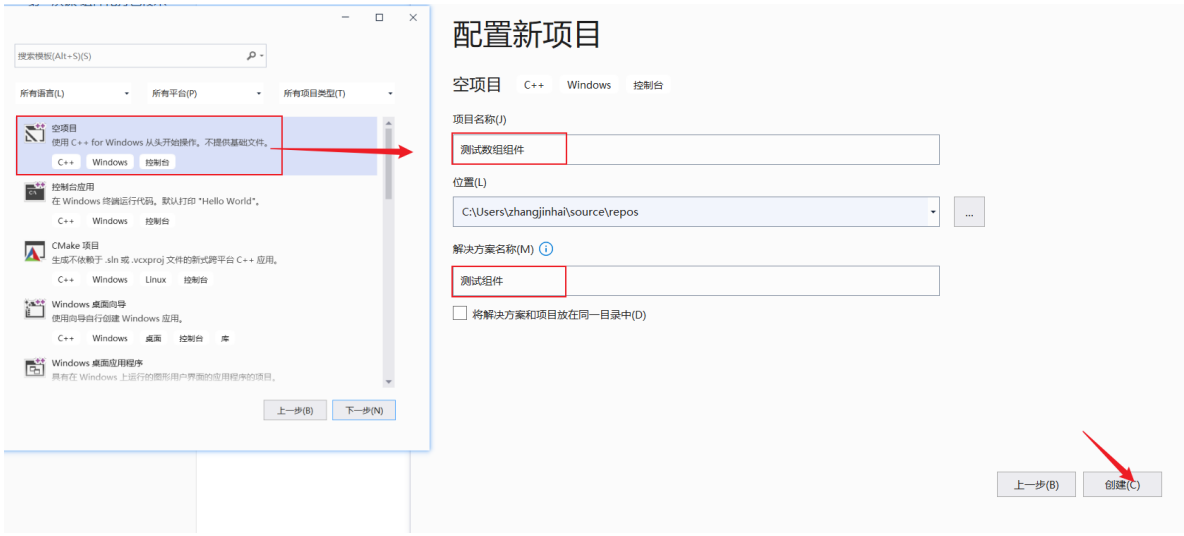


窗口则说明"试图执行此项目"。由于库项目不是生成可执行文件。所以有如上提示。

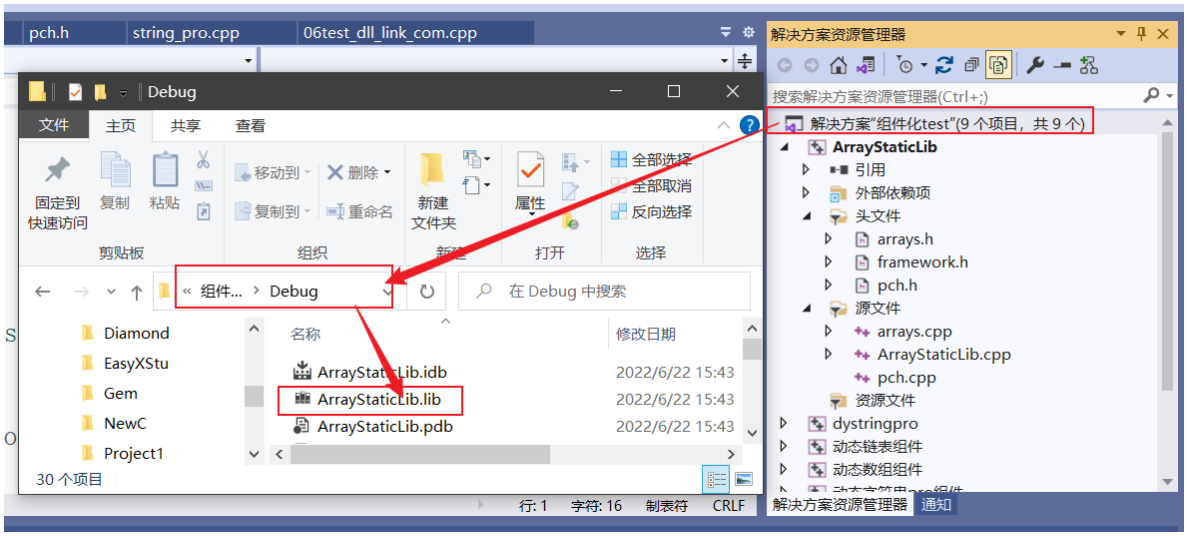
---

6. **验证静态库**

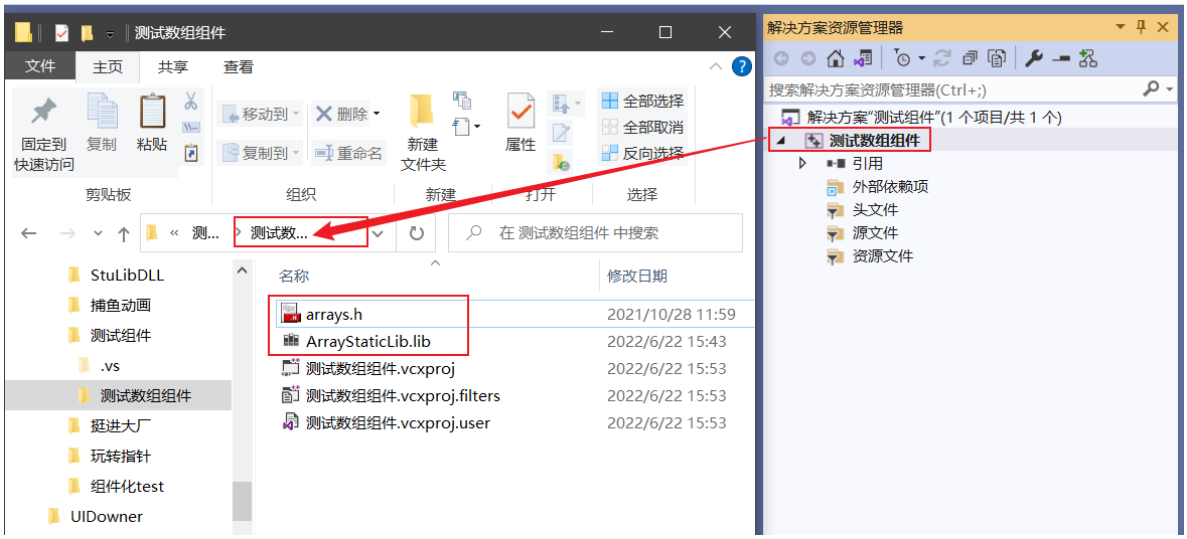
6.1 创建一个新的解决方案



6.2 到达 《组件化解决方案》debug文件夹找到生成的xxxx.lib文件(静态库)

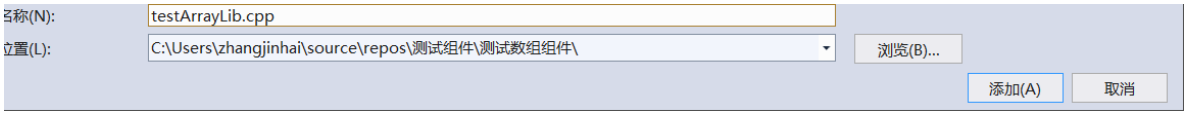


并复制到应用的项目里

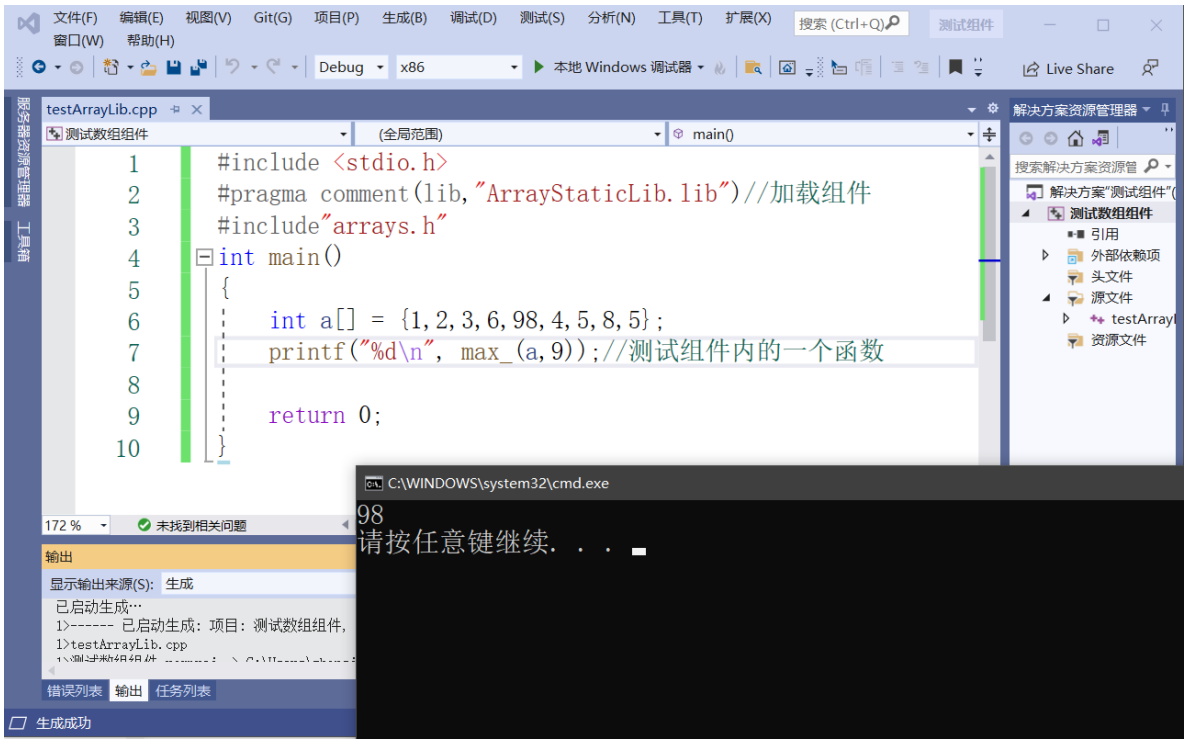


同时也需要把<头文件>一并复制进来。

6.3 写源代码使用《静态库组件》



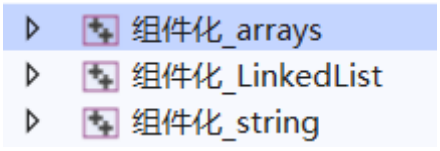
【注意】：测试文件的扩展名务必是：cpp 因为组件化的项目源代码也是cpp



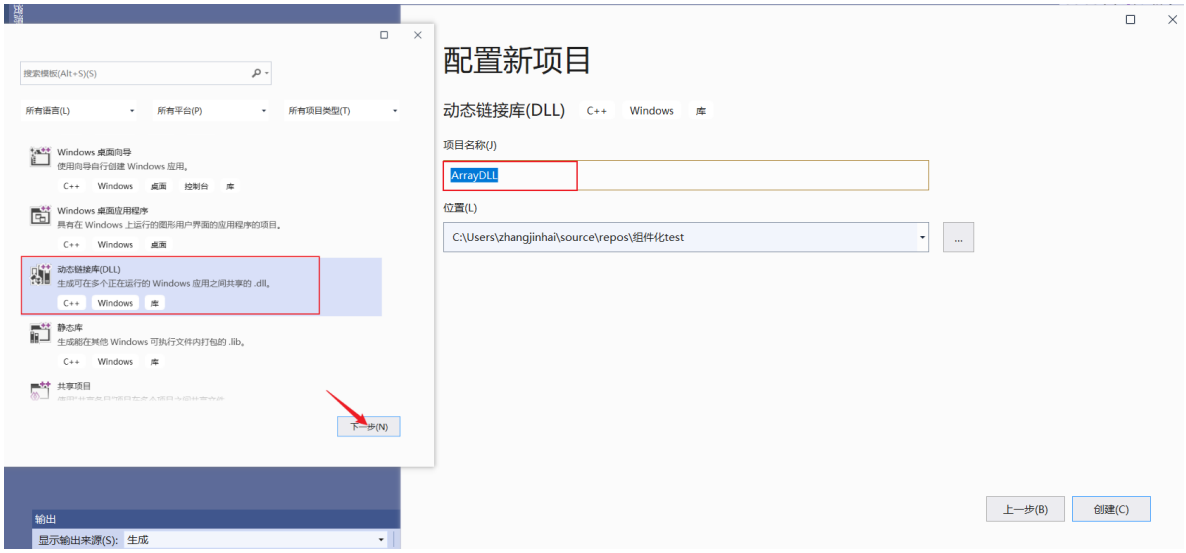

```c
#include <stdio.h>
#pragma comment(lib,"ArrayStaticLib.lib")//加载组件
#include"arrays.h"
int main()
{
    int a[] = {1,2,3,6,98,4,5,8,5};
    printf("%d\n", max_(a,9));//测试组件内的一个函数

    return 0;
}
```

四 、 动态链接库dll创建过程

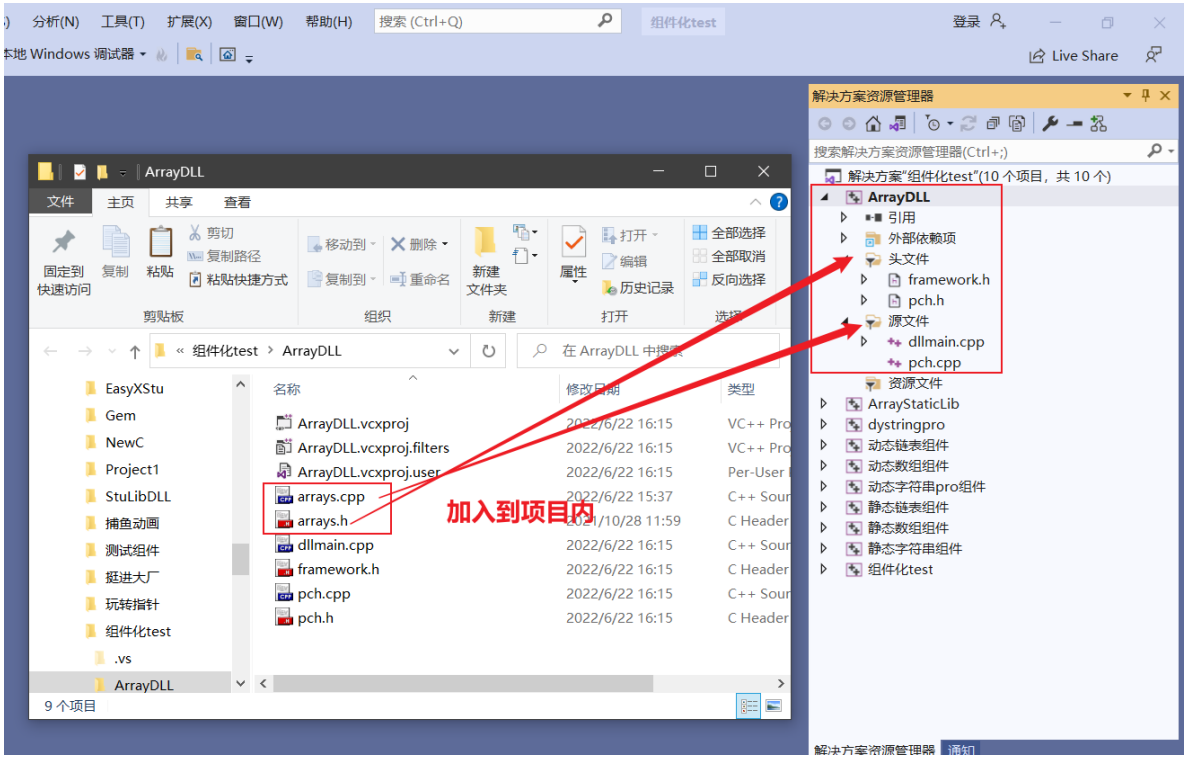
1、把已经测试完成的组件化项目及代码（头文件及对应的源代码文件）准备好



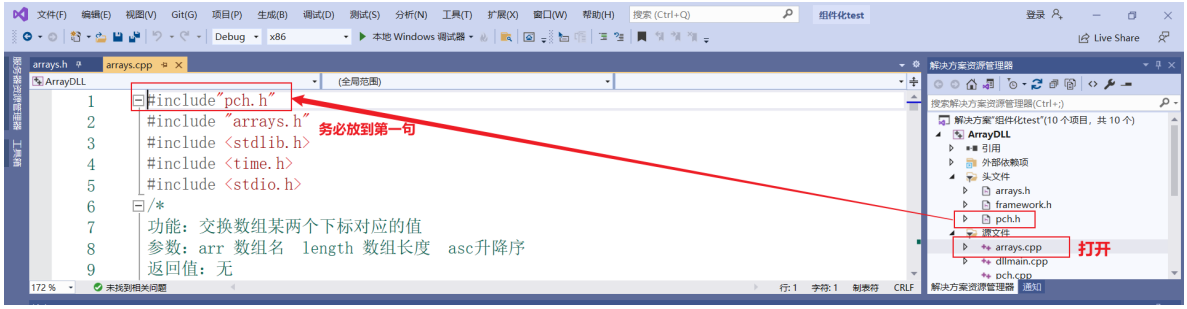
2、在"**解决方案**上"新建项目——>选择：动态链接库



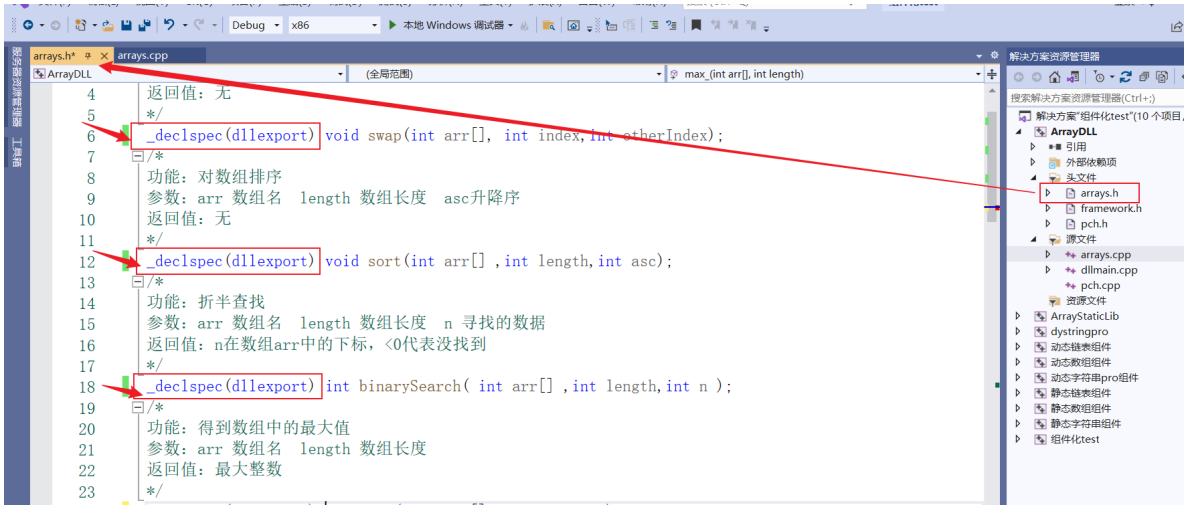
3 把组件化中开发好的头文件及源代码文件复制到DLL项目内并加入到指定位置



4 对源代码做一处改动：



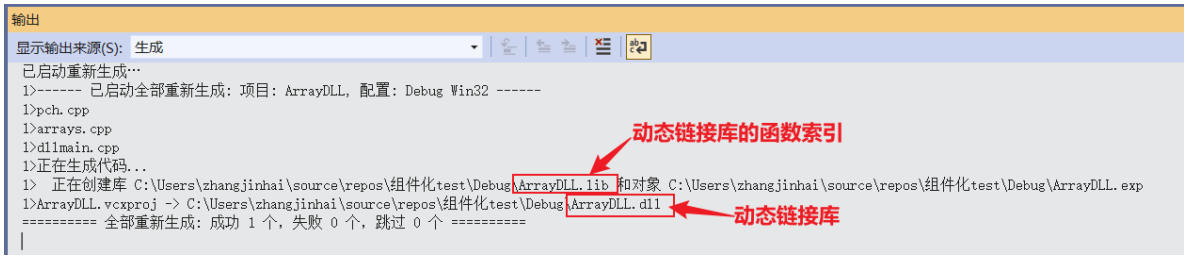
5 对头文件做如下改动：



_declspec(dllexport) 的含义是：让被修饰的函数 被导出 到DLL动态链接库中，也就是被外部所使用。

6 生成动态链接库

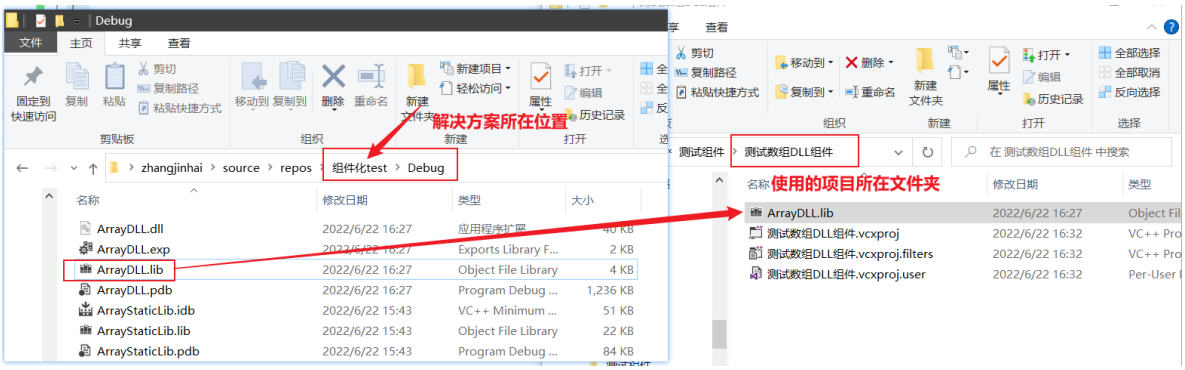
如果生成成功会产生如下提示：

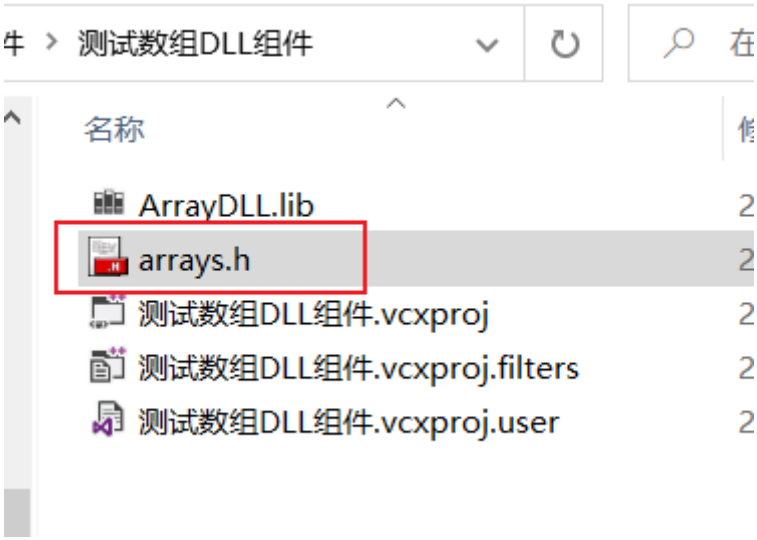


7 测试DLL文件

A：创建一个新的空项目准备使用DLL

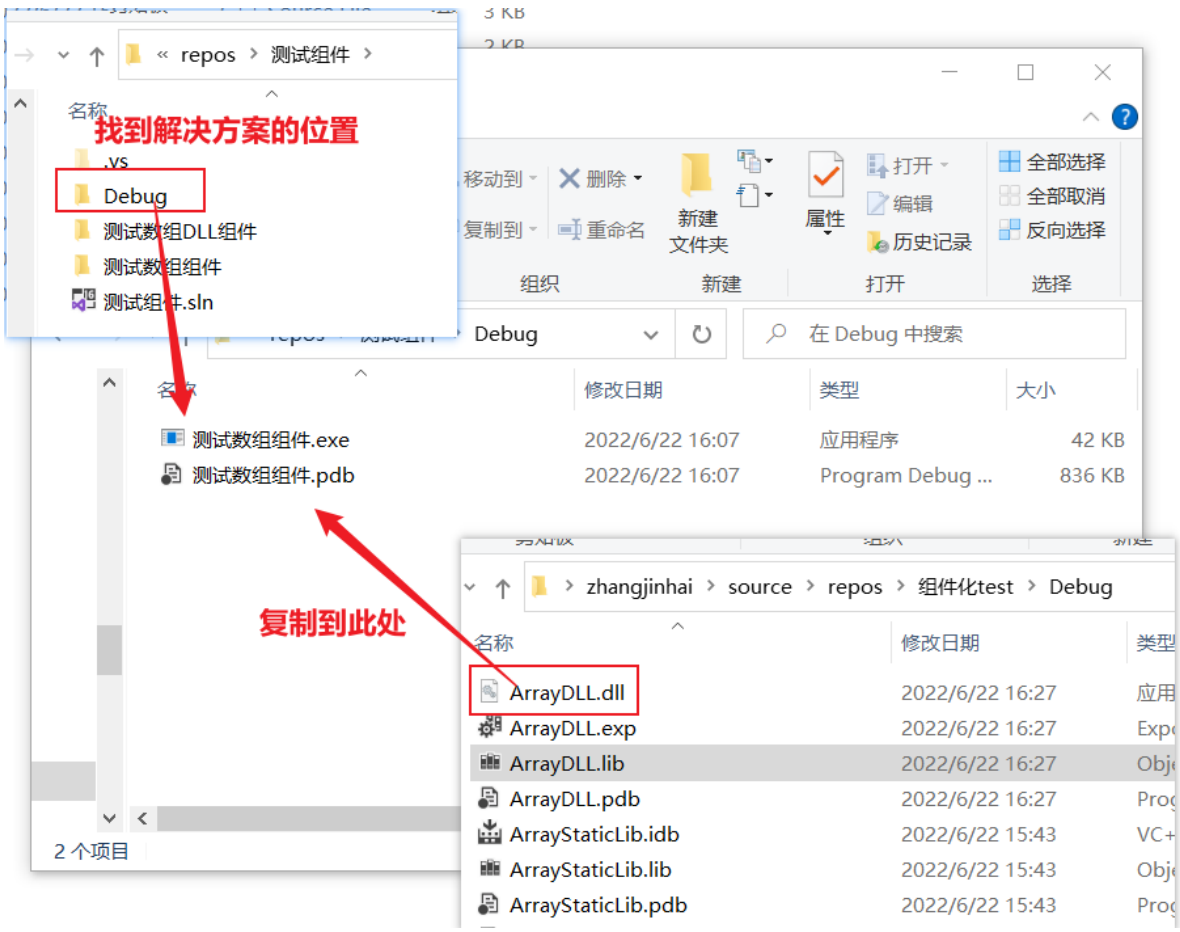
B：把xxx.lib文件（即动态链接库索引文件）和头文件复制到项目所在文件夹



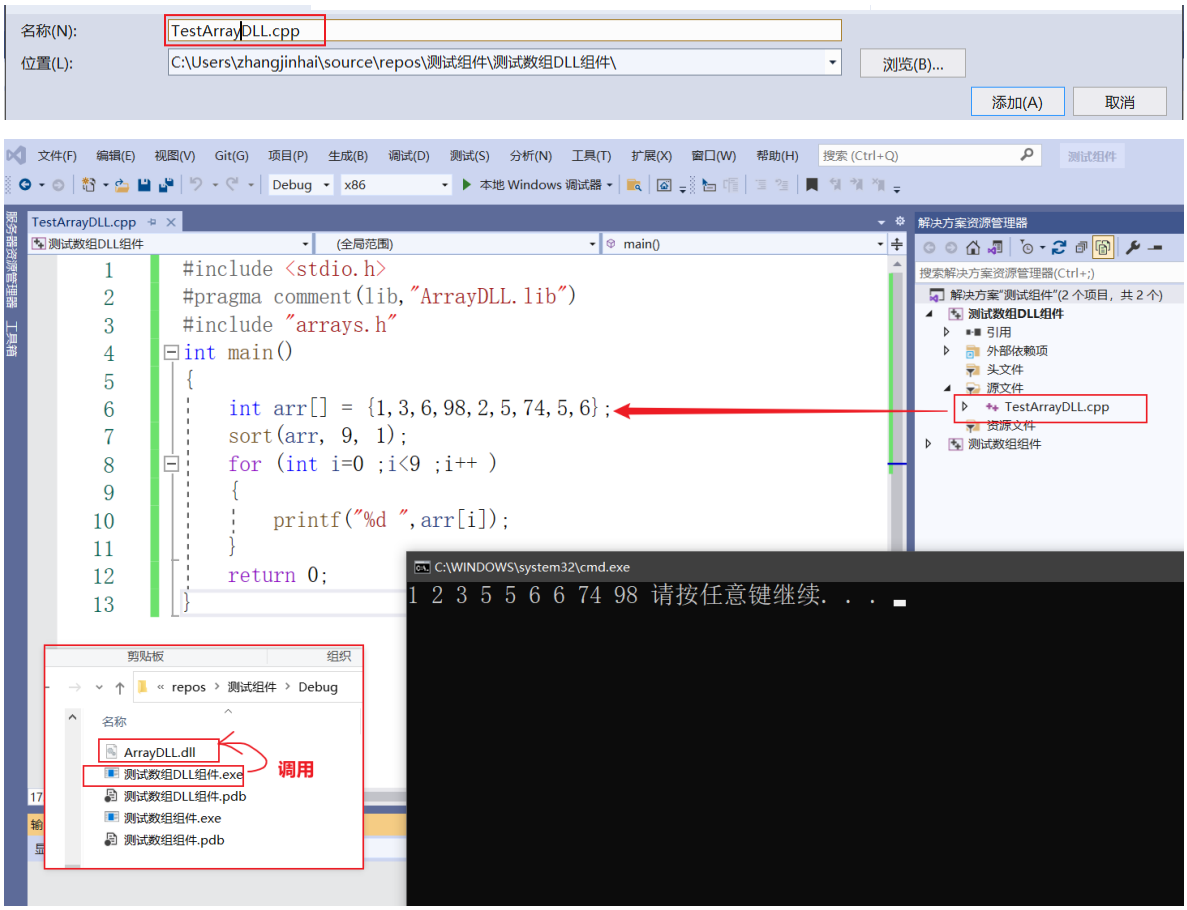


头文件也加入进来

C：xxxx.dll 放入到使用它的项目所在《解决方案》的debug文件夹下：即 生成的exe文件旁边


找到解决方案的位置
Debug
测试数组DLL组件
测试数组组件
测试组件.sln
测试数组组件.exe
2022/6/22 16:07
应用程序
42 KB
测试数组组件.pdb
2022/6/22 16:07
Program Debug ...
836 KB
复制到此处
ArrayDLL.dll
ArrayDLL.exp
ArrayDLL.lib
ArrayDLL.pdb
ArrayStaticLib.idb
ArrayStaticLib.lib
ArrayStaticLib.pdb
2 个项目

D： 写一段代码测试：


名称(N): TestArrayDLL.cpp
位置(L): C:\Users\zhangjinhai\source\repos\测试组件\测试数组DLL组件\
浏览(B)...
添加(A)
取消
#include <stdio.h>
#pragma comment(lib,"ArrayDLL.lib")
#include "arrays.h"
int main()
{
int arr[] = {1,3,6,98,2,5,74,5,6};
sort(arr, 9, 1);
for (int i=0 ;i<9 ;i++)
{
printf("%d ",arr[i]);
}
return 0;
}
C:\WINDOWS\system32\cmd.exe
1 2 3 5 5 6 6 74 98 请按任意键继续. . .
调用