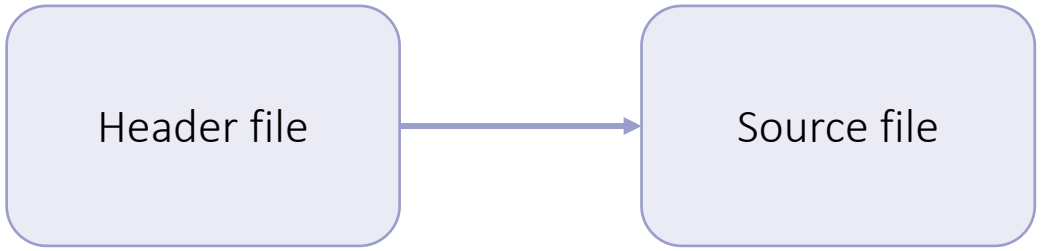
Multiple Files

Introduction to Computers and Programming

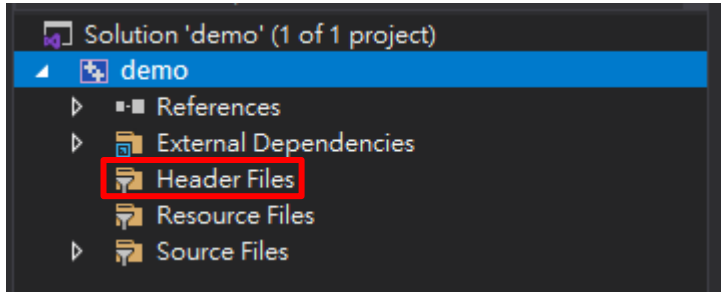
Header file

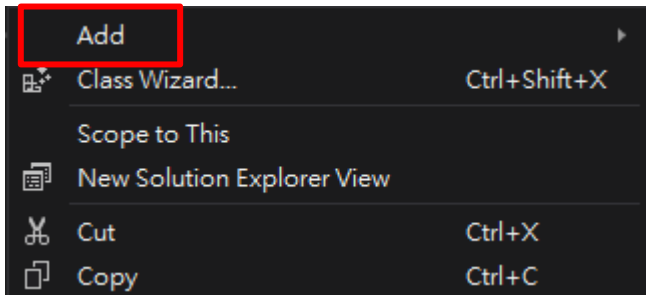
- header file is a file with extension **.h** which contains C function declarations and macro definitions to be shared between several source files.

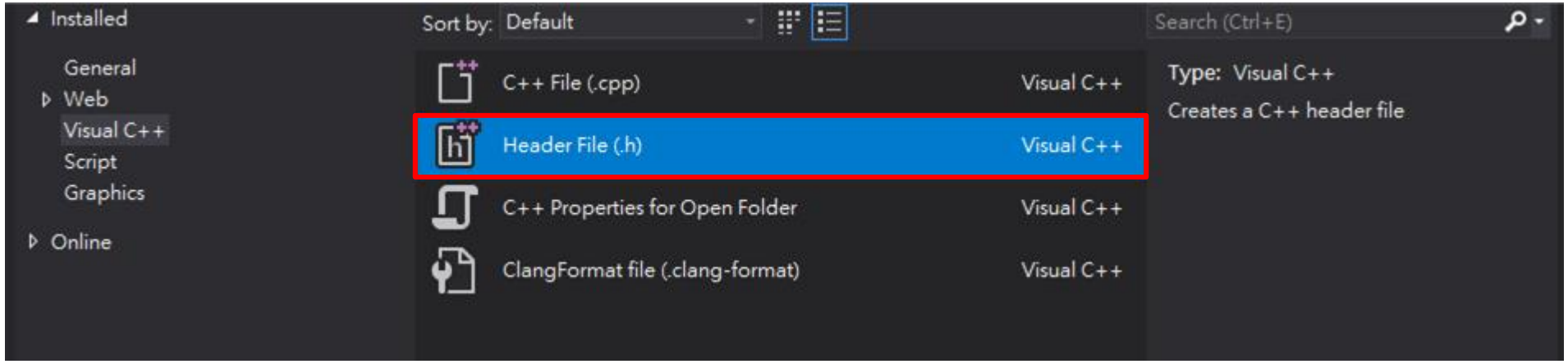
- There are two types of header files: the files that the programmer writes and the files that comes with your compiler.



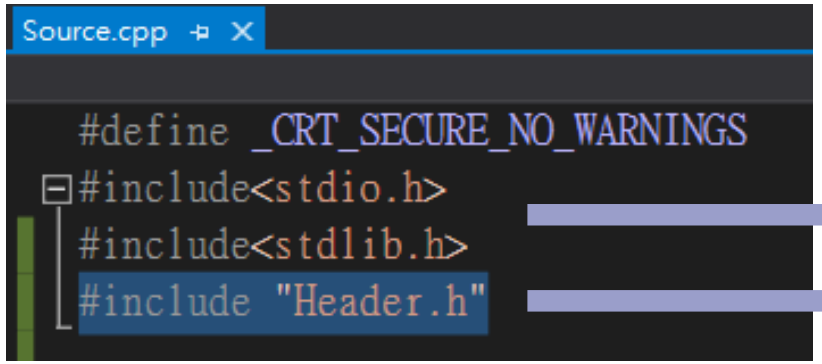
| Header file


Solution 'demo' (1 of 1 project)
demo
References
External Dependencies
Header Files
Resource Files
Source Files


Add
Class Wizard... Ctrl+Shift+X
Scope to This
New Solution Explorer View
Cut Ctrl+X
Copy Ctrl+C


Installed
General
Web
Visual C++
Script
Graphics
Online
Sort by: Default
Search (Ctrl+E)
C++ File (.cpp) Visual C++
Header File (.h) Visual C++
C++ Properties for Open Folder Visual C++
ClangFormat file (.clang-format) Visual C++
Type: Visual C++
Creates a C++ header file

Header file



used for system header files (#include <file>)

used for header files of your own program (#include "file")

| #define and macro

- #define is a keyword in C language. It is used for implementing macro.
- Macro in computer science is a rule or pattern that specifies how a certain input should be mapped to a replacement output.

For example :

#define MAX 10

(In the pre-compile state, compiler will replace all of the words MAX to 10.)

Or you can use it to define some pseudo function

#define MAX(a, b) a > b ? a : b

Benefit : Doesn’t need any function call cost.

Drawback : No type check, and may cause some wrong.

| #define and macro

```c
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>

# define MAX(a, b) a > b ? a : b


int main() {

    int a = 1, b = 2;
    printf("a=%d, b=%d\n", a, b);
    printf("MAX(a, b): %d\n", MAX(a, b));
    printf("3*MAX(a, b): %d\n", 3*MAX(a, b));

    return 0;
}
```

 # define MAX(a, b) a > b ? a : b

It’s not rigid enough to implement max function

```
MAX(a, b): 2
3*MAX(a, b): 1
```

3 * MAX(a, b), it will be parse to 3 * a > b? a : b

| #define and macro

```c
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>

# define MAX(a, b) ((a) > (b) ? (a) : (b))

int main() {

    int a = 1, b = 2;
    printf("a=%d, b=%d\n", a, b);
    printf("MAX(a, b): %d\n", MAX(a, b));
    printf("3*MAX(a, b): %d\n", 3*MAX(a, b));

    return 0;
}
```

define MAX(a, b) a > b ? a : b

 # define MAX(a, b) ((a) > (b)? (a) : (b))

```
a=1, b=2
MAX(a, b): 2
3*MAX(a, b): 6
```

Operator precedence

Level	Operators	Description	Associativity
15	() [] -> . ++ --	Function Call Array Subscript Member Selectors Postfix Increment/Decrement	Left to Right
14	++ -- + - ! ~ (type) * & sizeof	Prefix Increment / Decrement Unary plus / minus Logical negation / bitwise complement Casting Dereferencing Address of Find size in bytes	Right to Left
13	* / %	Multiplication Division Modulo	Left to Right
12	+ -	Addition / Subtraction	Left to Right
11	>> <<	Bitwise Right Shift Bitwise Left Shift	Left to Right
10	< <= > >=	Relational Less Than / Less than Equal To Relational Greater / Greater than Equal To	Left to Right
9	== !=	Equality Inequality	Left to Right
8	&	Bitwise AND	Left to Right
7	^	Bitwise XOR	Left to Right
6	|	Bitwise OR	Left to Right
5	&&	Logical AND	Left to Right
4	||	Logical OR	Left to Right
3	?:	Conditional Operator	Right to Left
2	= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operators	Right to Left
1	,	Comma Operator	Left to Right

 # define MAX(a, b) a > b ? a : b

3*MAX(a, b)

→ 3*a > b ? a : b

 # define MAX(a, b) ((a) > (b)? (a) : (b))

3*MAX(a, b)

→ 3*((a) > (b)? (a) : (b))

| #ifdef, #ifndef, #if, #elif, #else, #endif

- Conditional statement in pre-compile stage

```
#ifdef MACRO
    //successful code
#else
    //else code
#endif
```

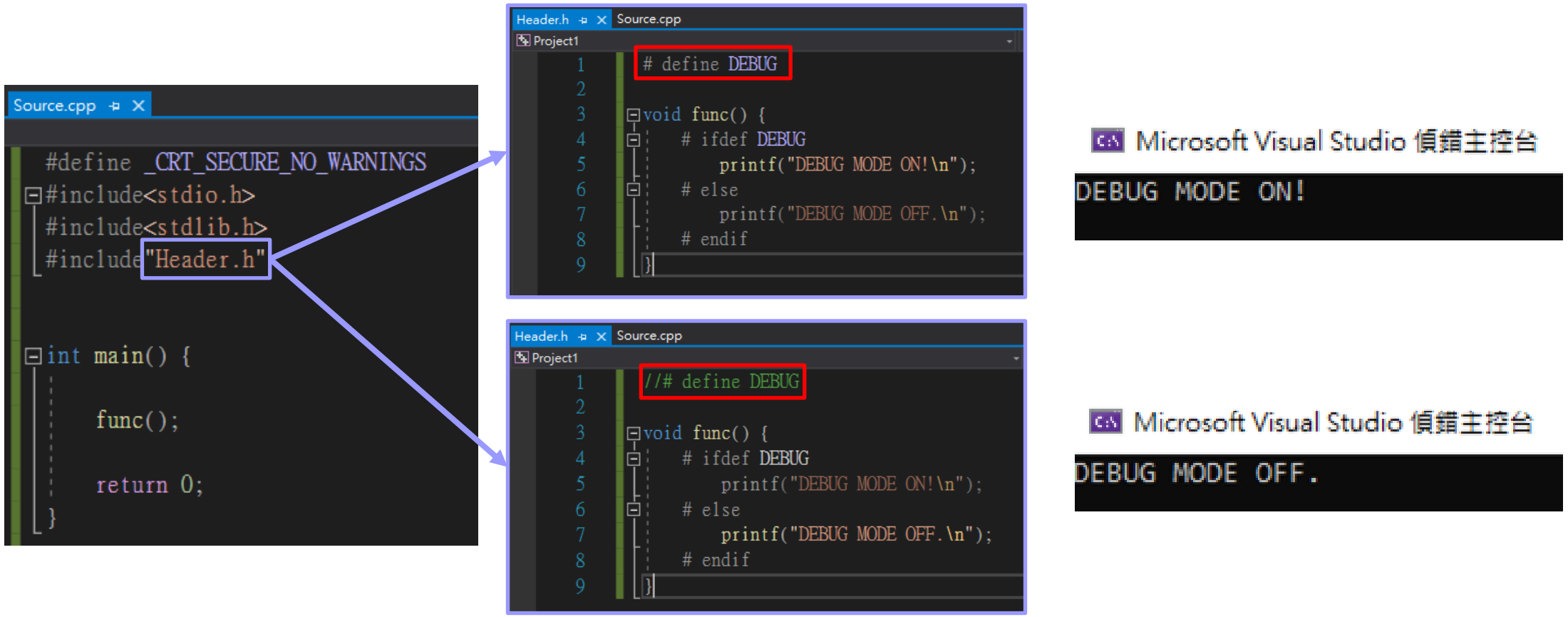
```
#ifndef MACRO
    //successful code
#else
    //else code
#endif
```

```
#if expression
    //if code
#elif expression
    //elif code
#else
    //else code
#endif
```

```
void func() {
    #if (NUMBER==10)
        printf("Value of Number is: 10");
    #else
        printf("Value of Number is: %d", NUMBER);
    #endif
}
```

#ifdef, #ifndef, #if, #elif, #else, #endif

For example, if we want to compile some block only in the debug mode:



| #ifdef, #ifndef, #if, #elif, #else, #endif

- If we want to create multiple header file, and they have some dependency relationship, it may occur redefined problem.

Source.cpp

```cpp
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>
#include "vector.h"
#include "matrix.h"

int main() {

    vector v;
    matrix2D mat;

    printf("OK");

    return 0;
}
```

vector.h

```cpp
typedef struct _vector {
    int length;
    int* vec;
}vector;
```

matrix.h

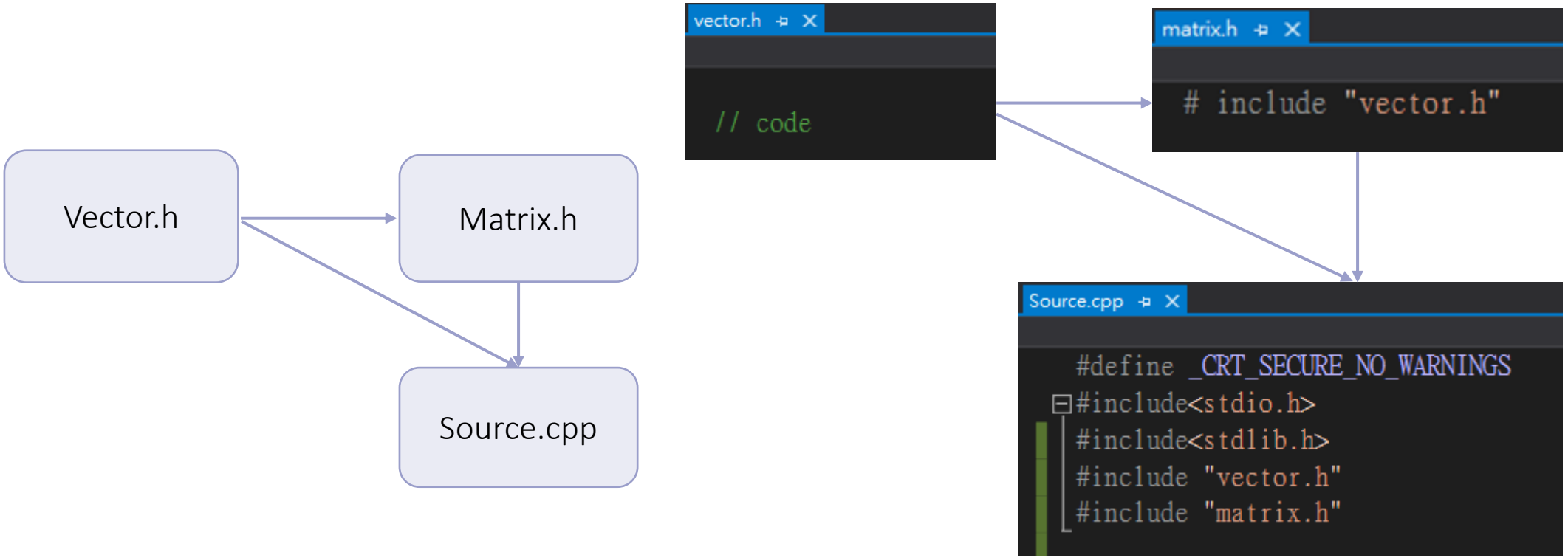
```cpp
# include "vector.h"

typedef struct _matrix2D {
    int row, column;
    vector* mat;

}matrix2D;
```

ERROR!

C2011 '_vector': 'struct' type redefinition

#ifdef, #ifndef, #if, #elif, #else, #endif

- If we want to create multiple header file, and they have some dependency relationship, it may occur redefined problem.



#ifdef, #ifndef, #if, #elif, #else, #endif

- If we want to create multiple header file, and they have some dependency relationship, it may occur redefined problem.

Source.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>
#include "vector.h"
#include "matrix.h"

int main() {

    vector v;
    matrix2D mat;

    printf("OK");

    return 0;
}
```

vector.h

```
# ifndef __VECTOR_H__
# define __VECTOR_H__

typedef struct _vector {
    int length;
    int* vec;
}vector;

# endif
```

matrix.h

```
# include "vector.h"

typedef struct _matrix2D {
    int row, column;
    vector* mat;

}matrix2D;
```

| extern and static

- If we want multiple source files share the same variable, we can define this variable in the header file directly and every source file which want to use this variable just need to include it.

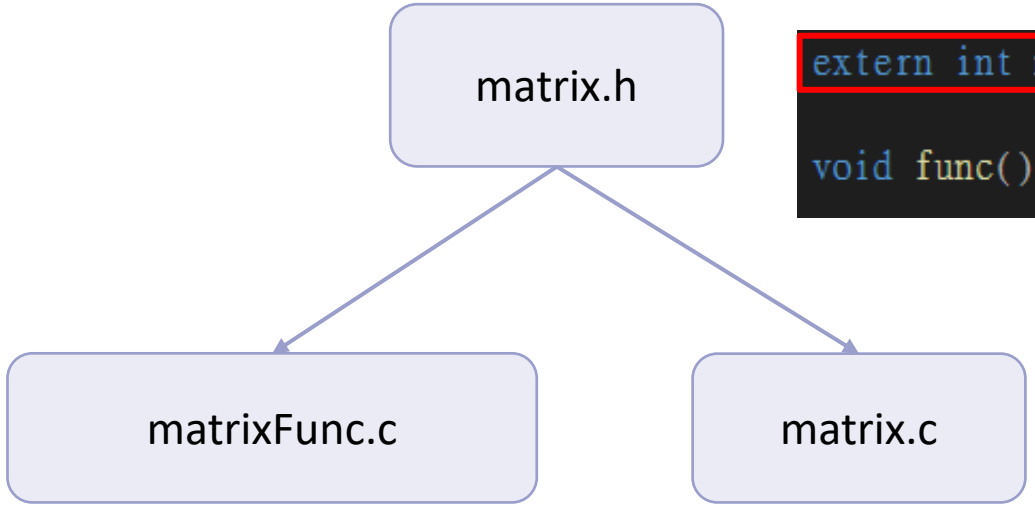
- But it is not a best way to define a global variable for these file, because it will initial the variable when you declare in the header file.

- Use the keyword extern to tell your compiler there is a variable need to be declare and initial in the source file.

extern and static

Microsoft Visual Studio 偵錯主控台

```
matrixCount=1
```



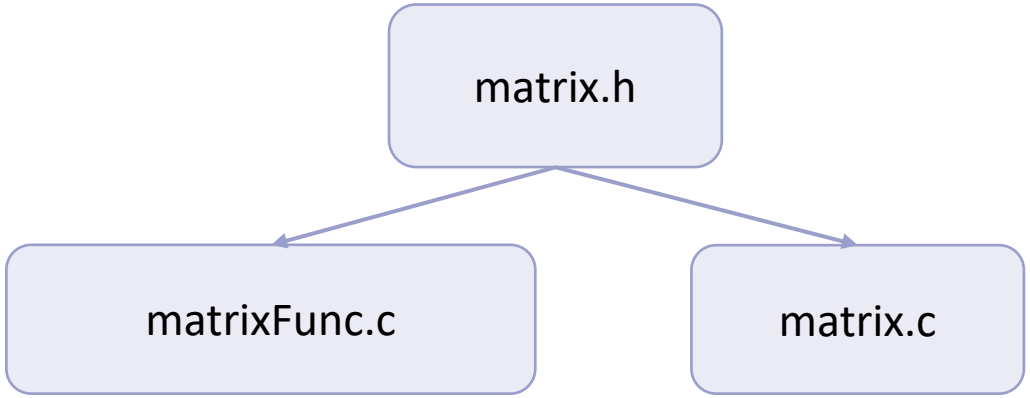

```h
extern int matrixCount;

void func();
```

```c
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>
#include "matrix.h"

void func() {
    printf("matrixCount=%d\n", matrixCount);
}
```

```c
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>
#include "matrix.h"

int matrixCount = 1;

int main() {

    func();

    return 0;
}
```

extern and static

- If we want to define a variable with a same name in different files, but we don't want they are shared:




```
#include "matrix.h"

matrix2D A;
```

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>
#include "matrix.h"

matrix2D A;
```

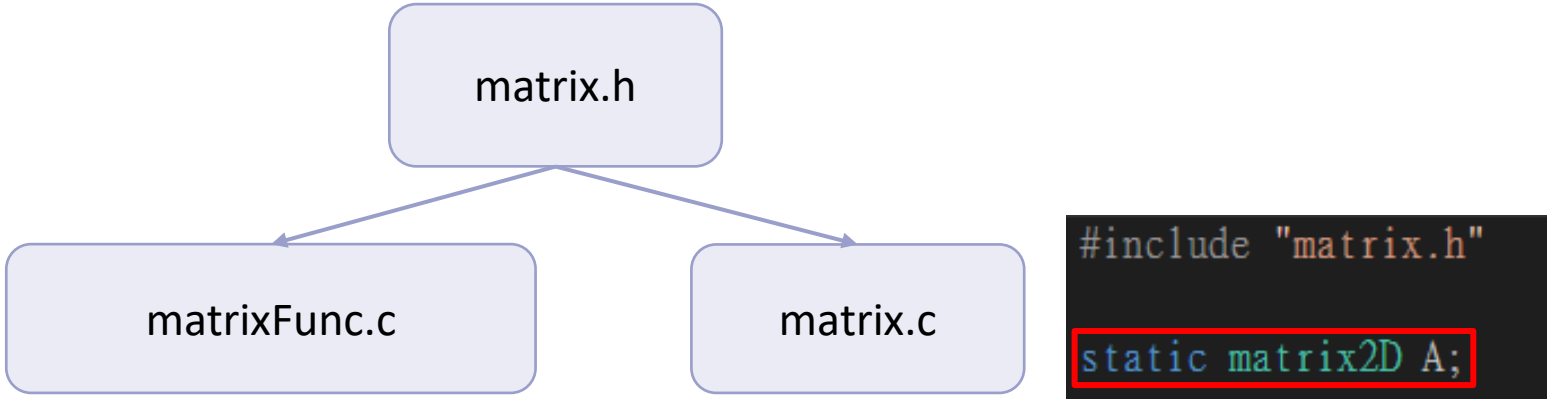
There is a multiple definition error.

But, don't worry, the keyword static can help you.

extern and static

- The keyword static means the variable will be only initialized once and only live in this scope.
- Static make the variable only live in this scope, you can't use extern to share this variable.

```c
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>
#include "matrix.h"

static matrix2D A;

int main() {

    printf("OK");
    return 0;
}
```



Microsoft Visual Studio 偵錯主控台

OK

extern and static

- Another usage of static:
- If we want to create a variable which can count how many times the function be called.

For example: call test 10 times.

```c
void test() {
    static int a = 0;
    a += 1;
    printf("a = %d\n", a);
}

int main() {
    for (int i = 0; i < 10; i++) {
        test();
    }
    return 0;
}
```

```
a = 1
a = 2
a = 3
a = 4
a = 5
a = 6
a = 7
a = 8
a = 9
a = 10
```

```c
void test() {
    int a = 0;
    a += 1;
    printf("a = %d\n", a);
}

int main() {
    for (int i = 0; i < 10; i++) {
        test();
    }
    return 0;
}
```

```
a = 1
a = 1
a = 1
a = 1
a = 1
a = 1
a = 1
a = 1
a = 1
a = 1
```

| Exercise - Introduction

Matrix Multiplication

There are two matrix, A and B. You need to construct Matrix A with A_row and A_column, and then fill it with the number A_number. (Same way to construct Matrix B)

- Input:

  Contains six numbers with the following sequence:

  A_row A_column B_row B_column A_number B_number

- Output:

  Multiplication of matrix A and B.

Input Example1:

```
5 2 2 5 3 9
```

Output Example1:

```
54 54 54 54 54
54 54 54 54 54
54 54 54 54 54
54 54 54 54 54
54 54 54 54 54
```

Input/Output Example2:

```
5 2 3 5 3 9
Shape Error
```

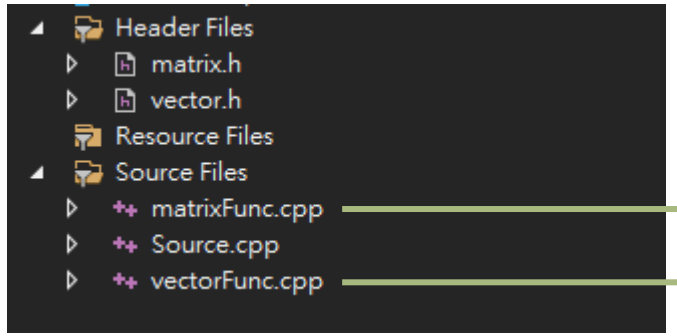
Exercise - Requirements

You will be given header files of matrix and vector, and source file with main function.

Do NOT modify them. You need to create matrixFunc.cpp and vectorFunc.cpp to implement the functions in matrix.h and vector.h, respectively. Please follow the spec to implement the function.

Note:

Your project should include the given matrix.h, vector.h, Source.cpp, and matrixFunc.cpp and vectorFunc.cpp created on your own.



implement the functions which are declared in “matrix.h”

implement the functions which are declared in “vector.h”