Topic 9

1. Object oriented programming
2. Implementing a simple class
3. Specifying the public interface
4. Designing the data representation
5. Member functions
6. Constructors
7. Problem solving: tracing objects
8. Problem solving: discovering classes
9. Separate compilation
10. Pointers to objects
11. Problem solving: patterns for object data

Separate Compilation

For small programs, all your code fits in a single file.

When your programs get larger or you work in a team, you will want to split your code into separate source files, because:

1. You don't want to wait for the compiler to keep translating code that doesn't change. If your code is distributed over several source files, then only those files that you changed need to be recompiled.

2. On a team project, code is broken up so that each programmer can work on his/her files without conflict.

.

Separate Compilation: Header Files

If your program is composed of multiple files, some of these files will define data types or functions that are needed in other files.

There must be a path of communication between the files.

In C++, that communication happens through the inclusion of header files.

Yes, **`#include`**.

Separate Compilation: What to include

The code will be in two kinds of files:

header files (filename.h)
(which will be **`#include`**-ed)

source files (filename.cpp)
(which we usually do not **`#include`**)

Separate Compilation: Files

A header file contains

- the interface:
  - Definitions of classes.
  - Definitions of constants.
  - Declarations of nonmember functions.

A source file contains

- the implementation:
  - Definitions of member functions.
  - Definitions of nonmember functions.
  - May or may not contain `main()`

Separate Compilation: Example

`cashregister.h`

the interface – the class definition

`cashregister.cpp`

the implementation – all the member function definitions

Separate Compilation: cashregister.h

This is the header file, **`cashregister.h`**

Notice the `#ifndef …  #define` at the top.

There is an ending **`#endif`** at the end of the file.

This makes sure the header is only included once, to prevent compiler errors such as "redefined".

```
#ifndef CASHREGISTER_H
#define CASHREGISTER_H

/**
   A simulated cash register that tracks
   the item count and the total amount due.
*/
class CashRegister …
```

Separate Compilation: .h file (cont)

```
   ...
private:
   int item_count;
   double total_price;
};

#endif
```

You include this header file whenever the definition of the **CashRegister** class is required.

Since this file is not a standard header file, you must enclose its name in quotes, not **<...>**, when you include it, like this:

```
#include "cashregister.h"
```

Separate Compilation: The Class .cpp file

Notice that the implementation file **#include**s its header file.

```
#include "cashregister.h"
CashRegister::CashRegister()
{
   clear();
}
void CashRegister::clear()
{
   item_count = 0;
   total_price = 0;
}

Etc...
```

Separate Compilation: The .cpp file (2)

```
void CashRegister::add_item(double price)
{
   item_count++;
   total_price = total_price + price;
}

double CashRegister::get_total() const
{
   return total_price;
}
int CashRegister::get_count() const
{
   return item_count;
}
```

Separate Compilation: The `main()` Program (1)

```
#include <iostream>
#include <iomanip>
#include "cashregister.h"
using namespace std;

/**
   Displays the item count and total
   price of a cash register.
   @param reg the cash register to display
*/
void display(CashRegister reg)
{
   cout << reg.get_count() << " $"
      << fixed << setprecision(2)
      << reg.get_total() << endl;
}
```

Separate Compilation: The `main()` Program (2)

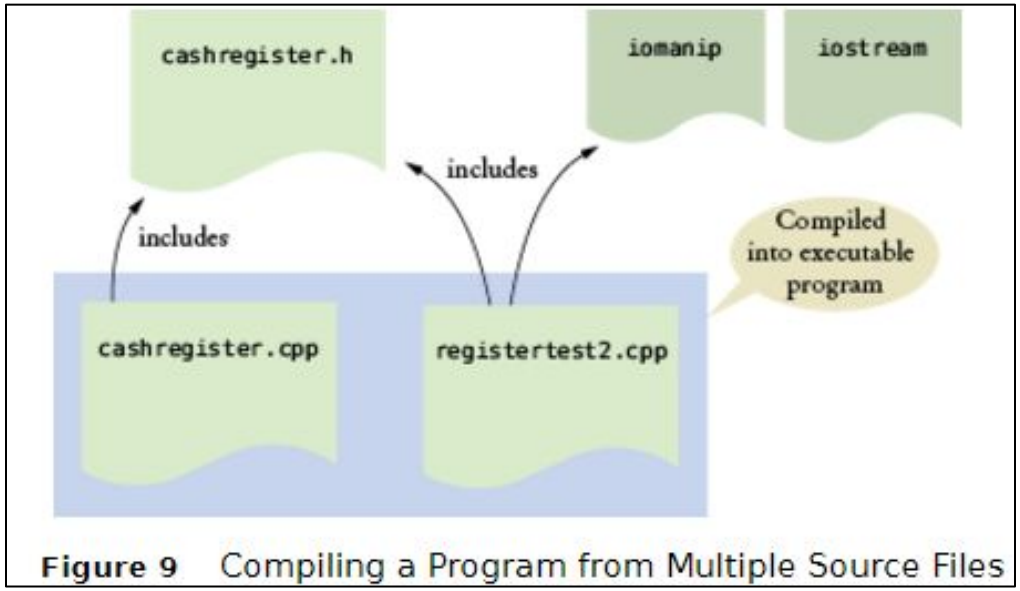


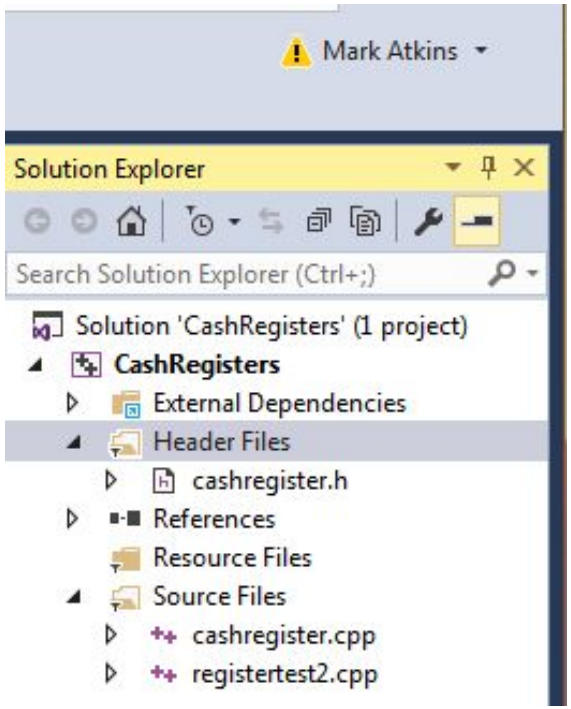
Figure 9 Compiling a Program from Multiple Source Files

```
int main()
{
   CashRegister register1;
   register1.clear();
   register1.add_item(1.95);
   display(register1);
   register1.add_item(0.95);
   display(register1);
   register1.add_item(2.50);
   display(register1);
   return 0;
}
```

Separate Compilation: The Mechanics of the IDE

- In an Integrated Development Environment, you must specify all the source files that need to be compiled/linked together
  - In Visual Studio, you should put the
    - .h file(s) into the Header Files

    or Source Files folders of the Solution Explorer tree
    - .cpp files into the Source Files folder



- In a command-line compiler, you can supply a “makefile” which lists the various files and their containing folders
  - Then you run the “make” utility to build the program .exe
  - See https://www.gnu.org/software/make/manual/ for details