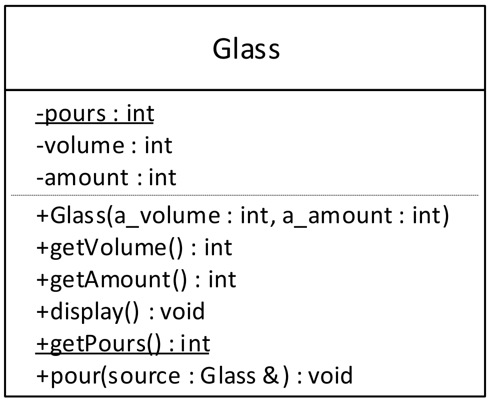
Slide 1

The Glass Class

Glass Members

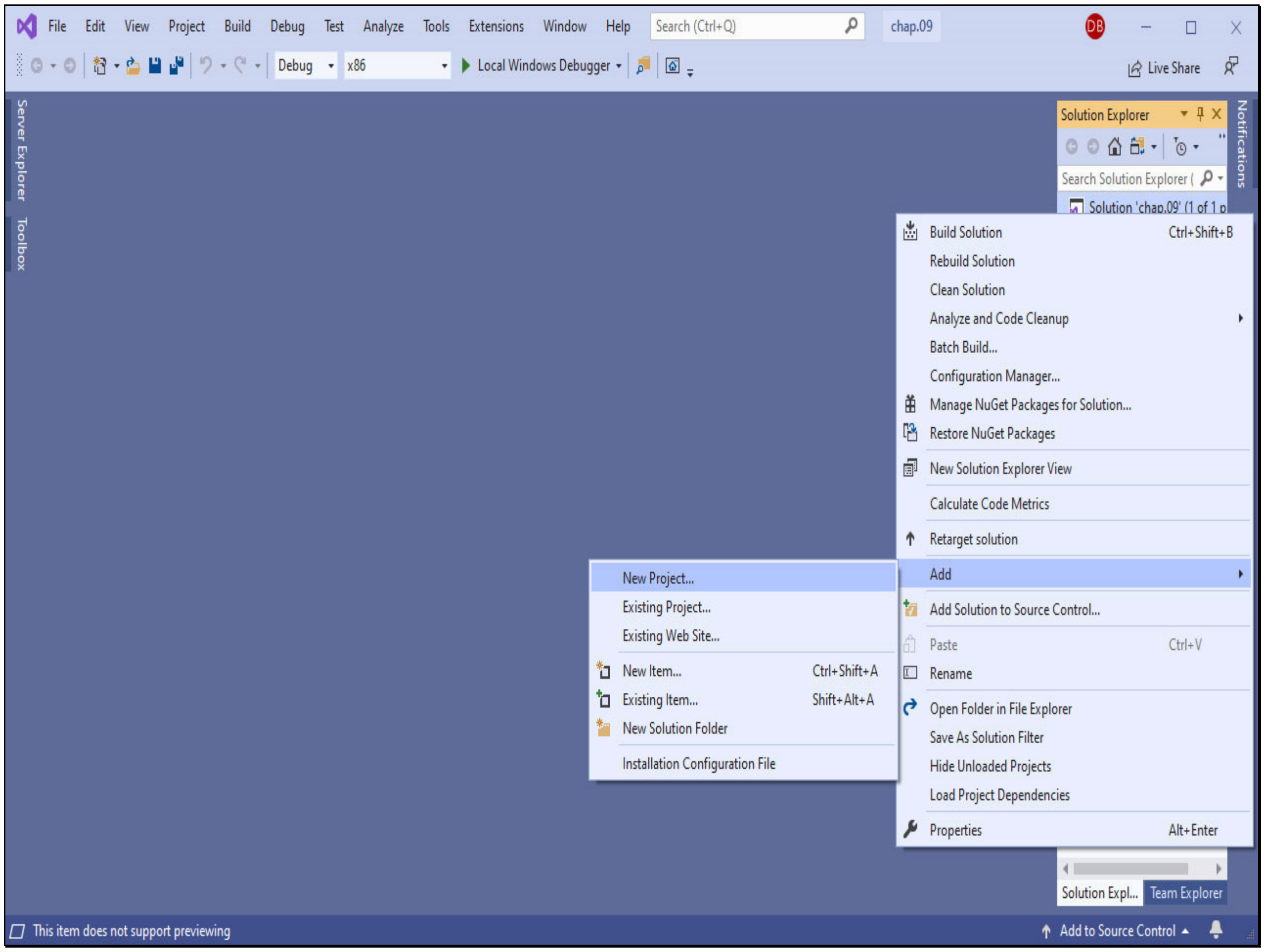
UML class diagrams should be language agnostic

For clarity, the pour function is implemented as pass by reference



UML class diagrams are meant to be independent of any specific programming language. That means that features that are specific to a given language are typically omitted from a UML diagram. For example, looking at the pour operation at the bottom of the diagram, we wouldn’t typically show it as a pass by reference on a UML diagram, and instead would let C++ programmers choose pass by reference or pass by pointer; programmers using other languages would choose a passing technique appropriate for their language. However, to simplify and clarify this programming demonstration, we’ll explicitly note that the pour function is implemented as pass by reference.

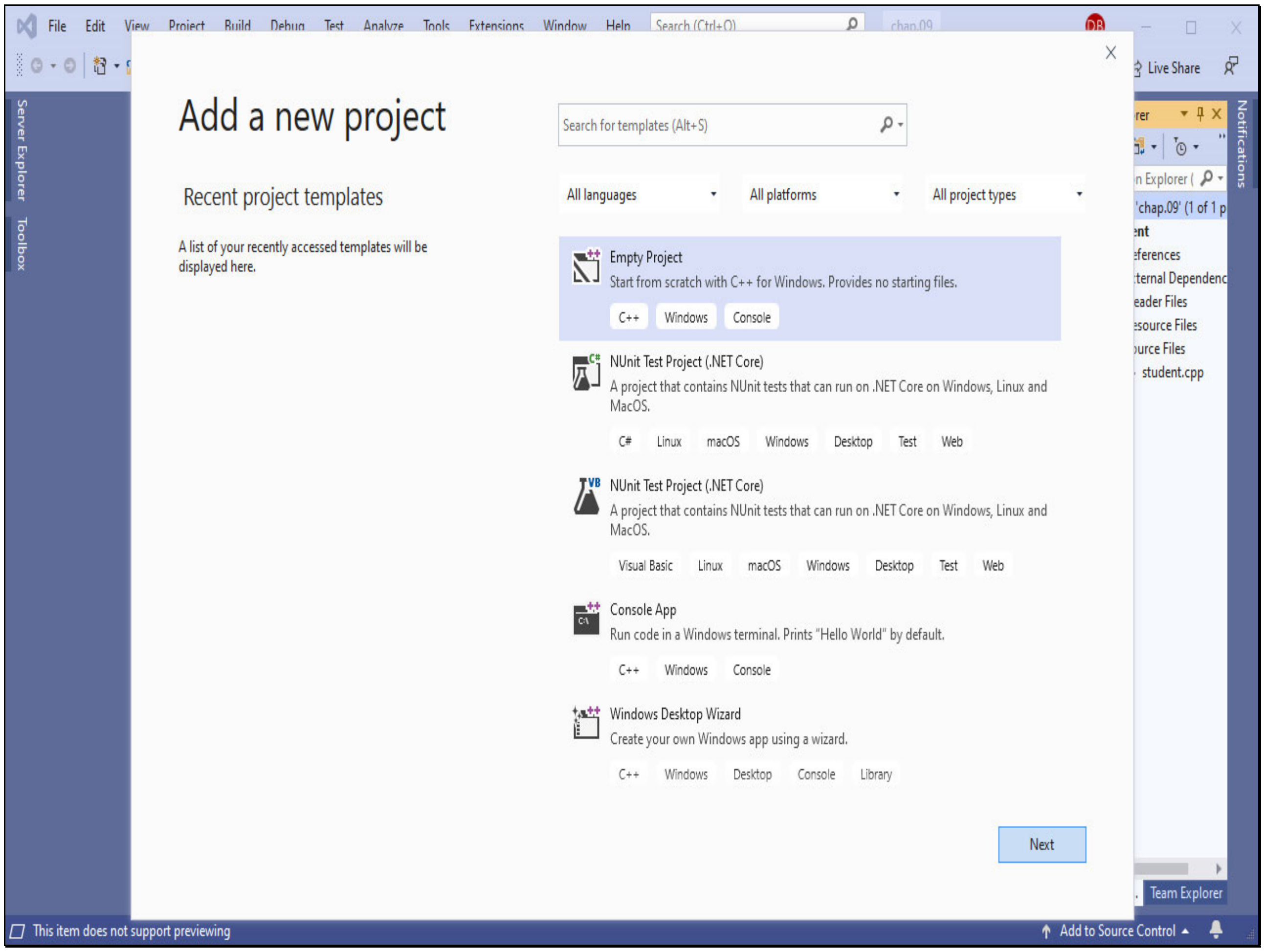
Slide 2



Text Captions

We begin by making a new project.

Slide 3



Add a new project

Search for templates (Alt+S)

Recent project templates

A list of your recently accessed templates will be displayed here.

All languages | All platforms | All project types

Empty Project
Start from scratch with C++ for Windows. Provides no starting files.
C++ Windows Console

NUnit Test Project (.NET Core)
A project that contains NUnit tests that can run on .NET Core on Windows, Linux and MacOS.
C# Linux macOS Windows Desktop Test Web

NUnit Test Project (.NET Core)
A project that contains NUnit tests that can run on .NET Core on Windows, Linux and MacOS.
Visual Basic Linux macOS Windows Desktop Test Web

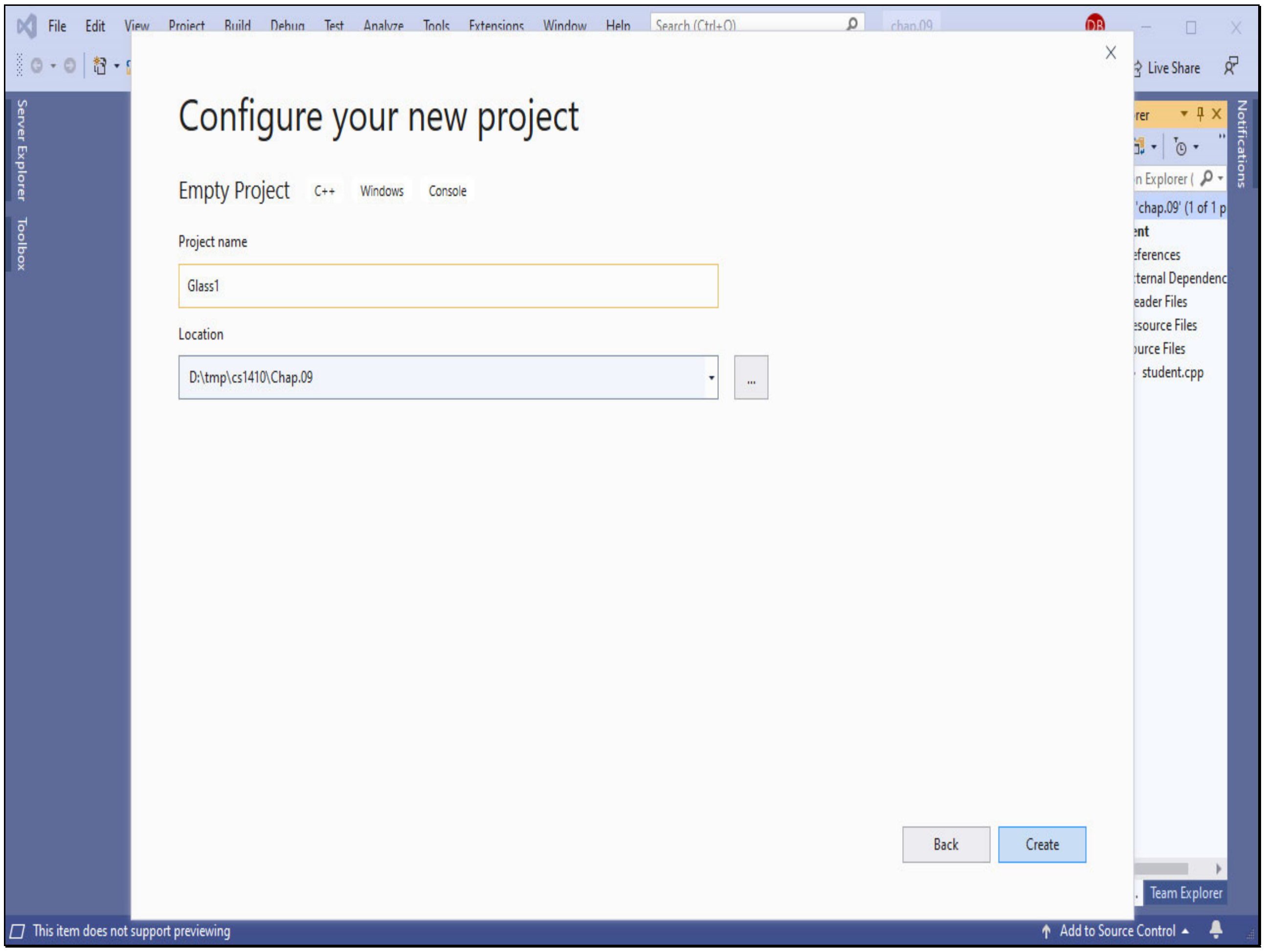
Console App
Run code in a Windows terminal. Prints "Hello World" by default.
C++ Windows Console

Windows Desktop Wizard
Create your own Windows app using a wizard.
C++ Windows Desktop Console Library

Text Captions

Make sure that the project is empty.

Slide 4



Configure your new project

Empty Project C++ Windows Console

Project name: Glass1

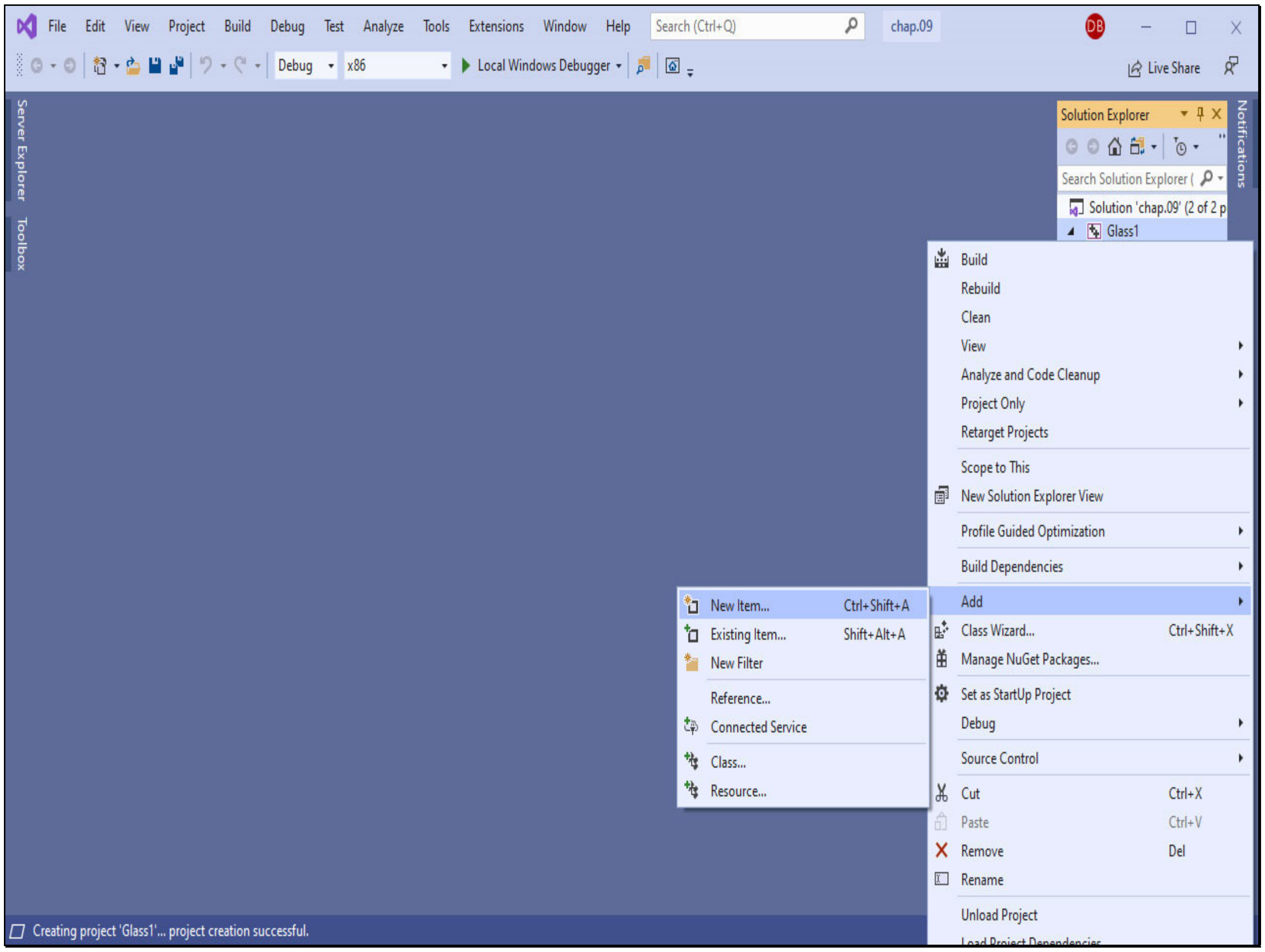
Location: D:\tmp\cs1410\Chap.09

Back | Create

Text Captions

Name the project “Glass1” (we’ll write a slightly different version in the next section).

Slide 5

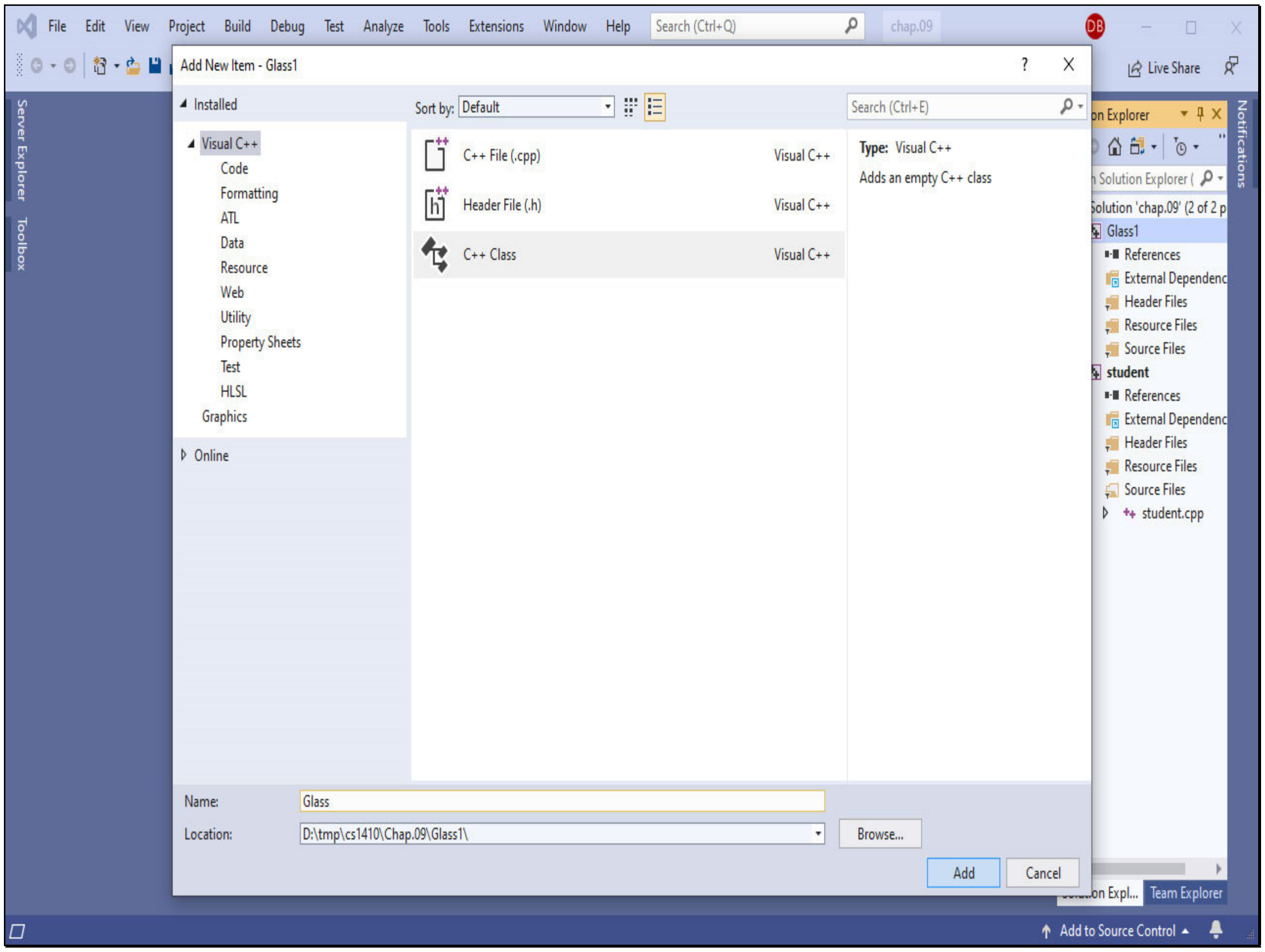


Creating project 'Glass1'... project creation successful.

Text Captions

Our next step will add a both a header and a source code file to the project.

Slide 6



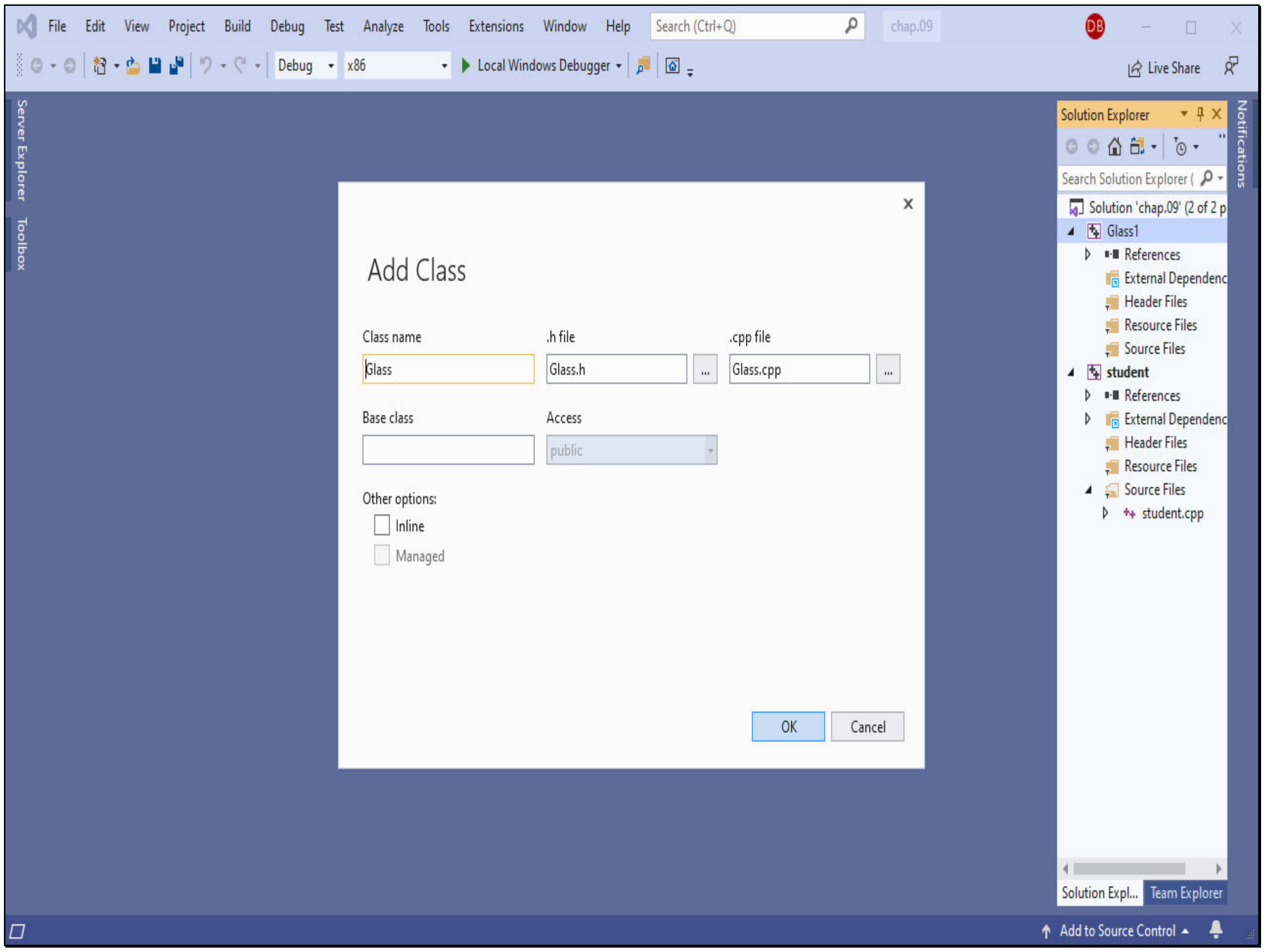
Name: Glass

Location: D:\tmp\cs1410\Chap.09\Glass1\

Text Captions

Choose "C++ Class" from the menu and name it "Glass." Doing this creates a header file with the Glass class specification started and a source code file – all in one step!

Slide 7



Add Class

Class name: Glass

.h file: Glass.h

.cpp file: Glass.cpp

Base class:

Access: public

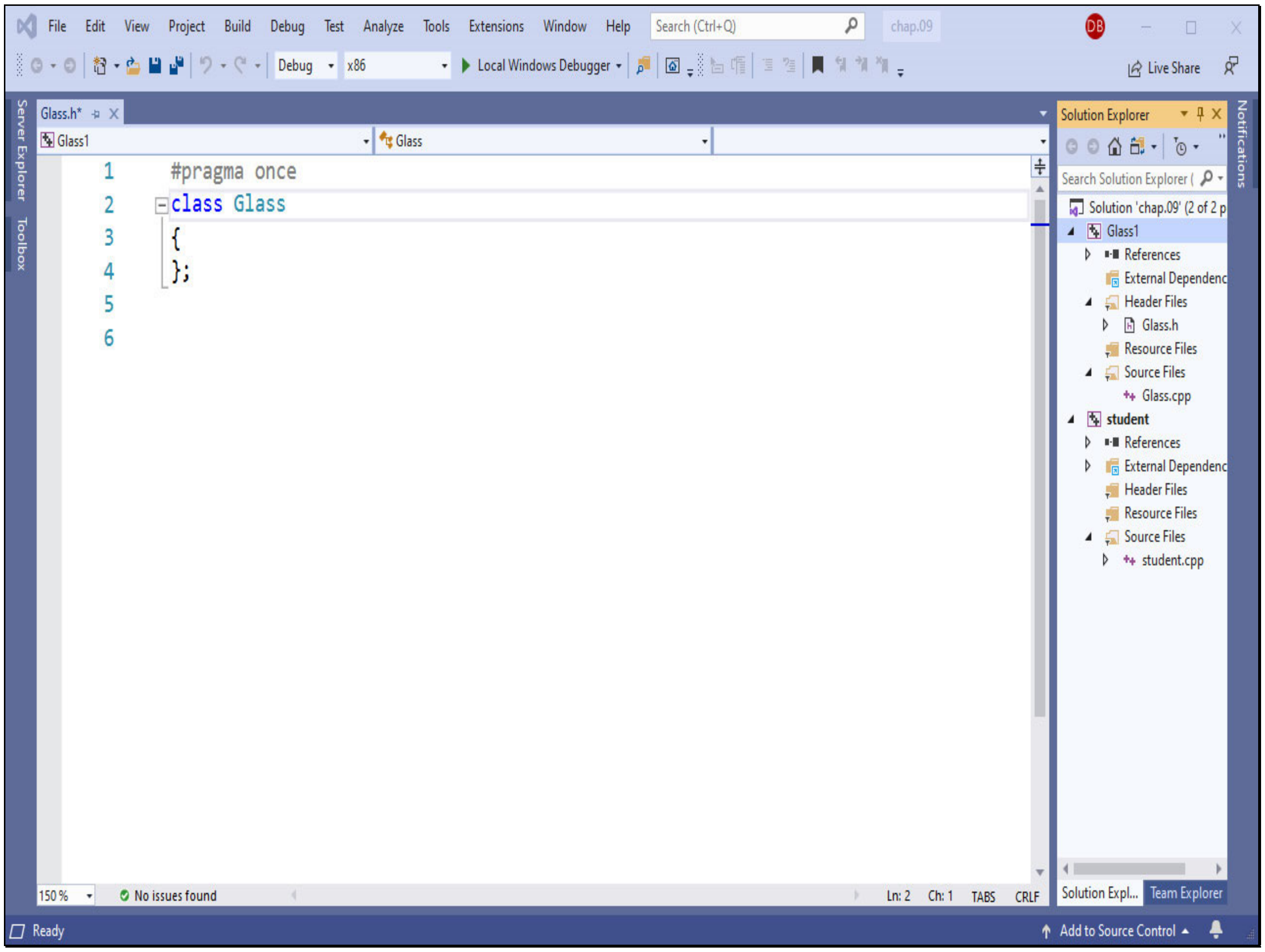
Other options:

- Inline
- Managed

Text Captions

The next pop up window allows us to change the name of the class and/or the names of the two files. We'll take the default values, so just press the "OK" button.

Slide 8

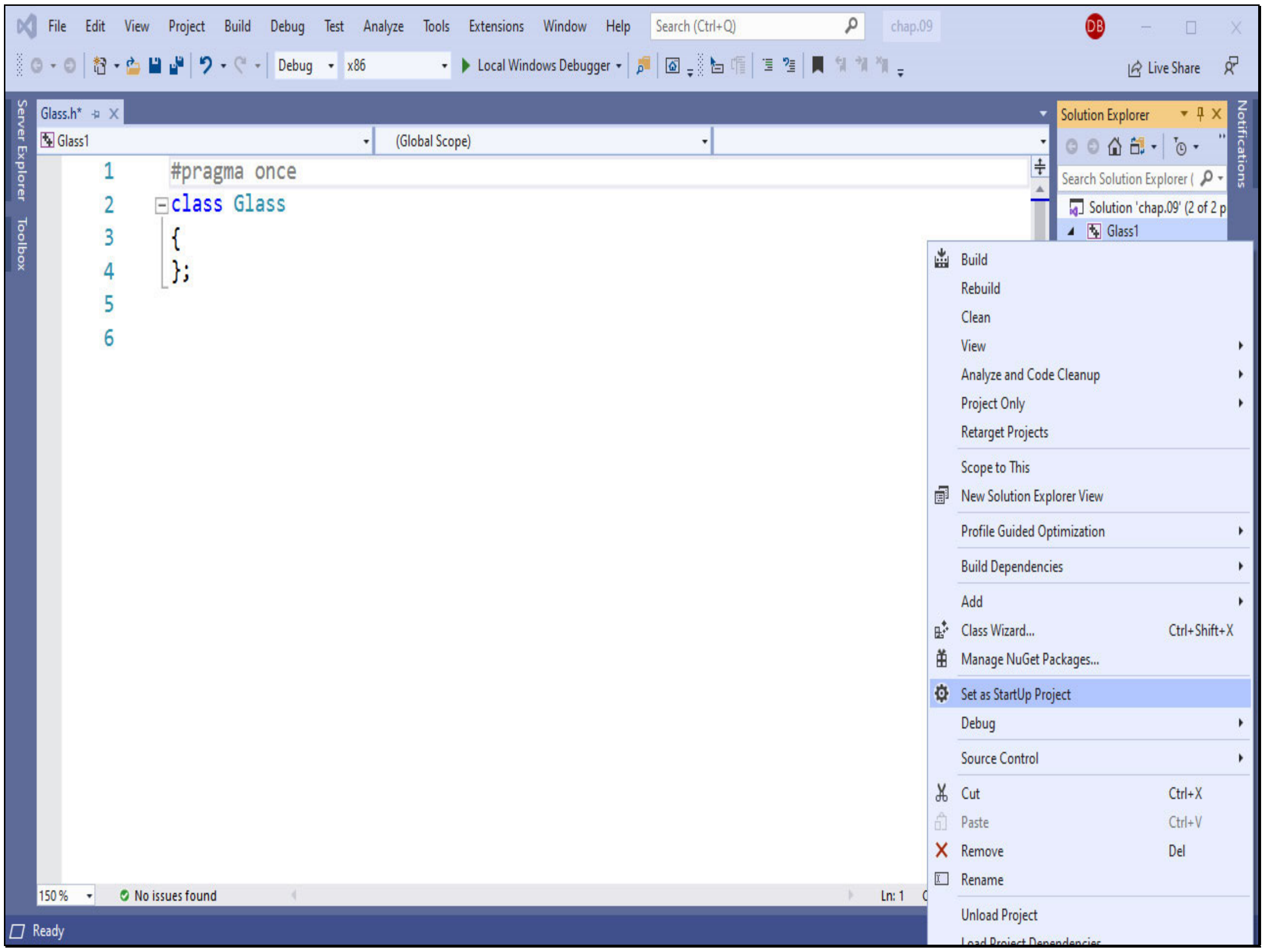


```
#pragma once
class Glass
{
};
```

Text Captions

You can now see both file names in the “Solution Explorer” pane. Click on “Glass.h” in the Solution Explorer to open the file. We’ll begin by filling out the class specification started by Visual Studio.

Slide 9

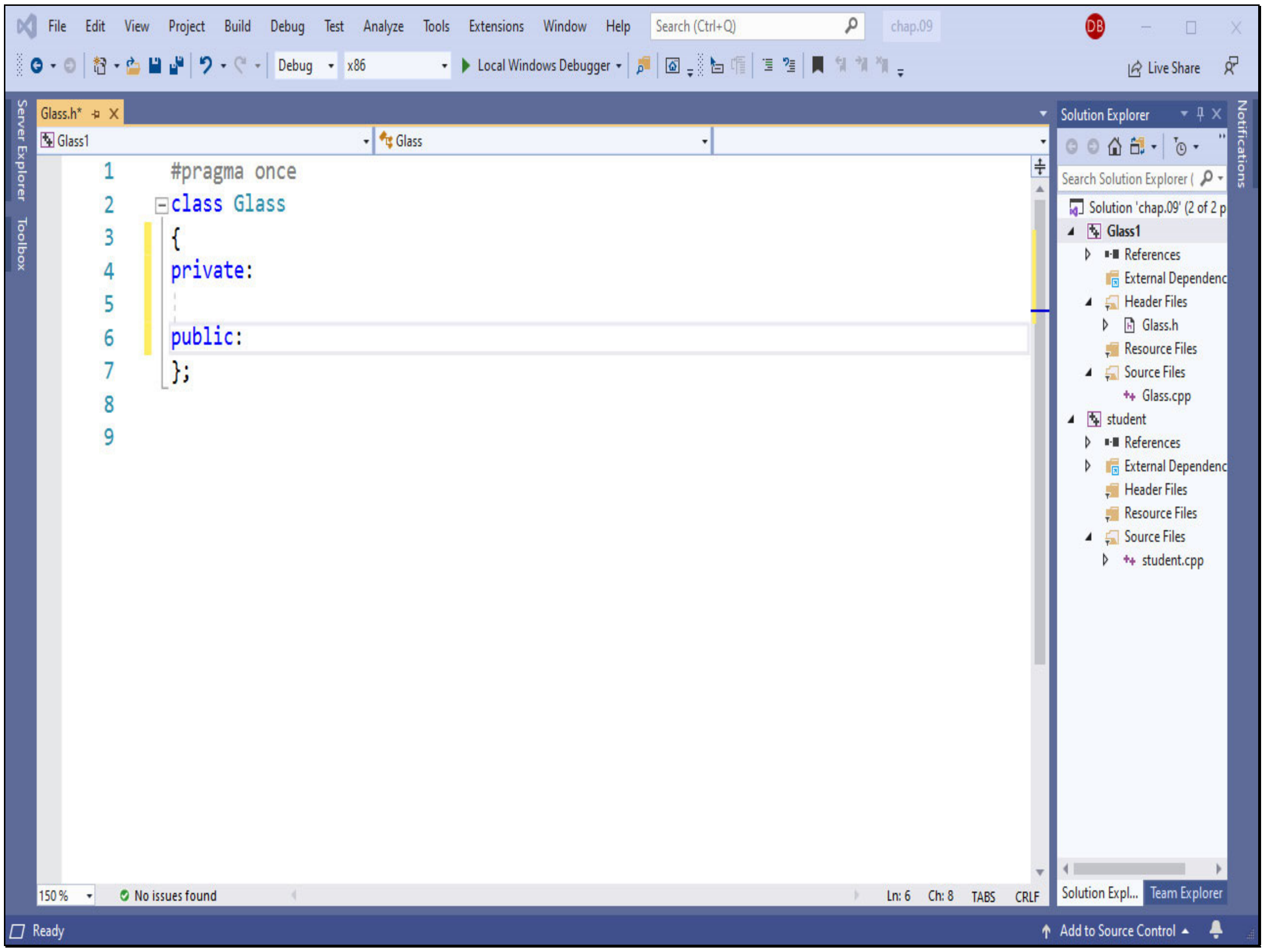


```
#pragma once
class Glass
{
};
```

Text Captions

I often forget to make the new project the startup project, so let's do that now by right-clicking the project and selecting "Set as Startup Project."

Slide 10

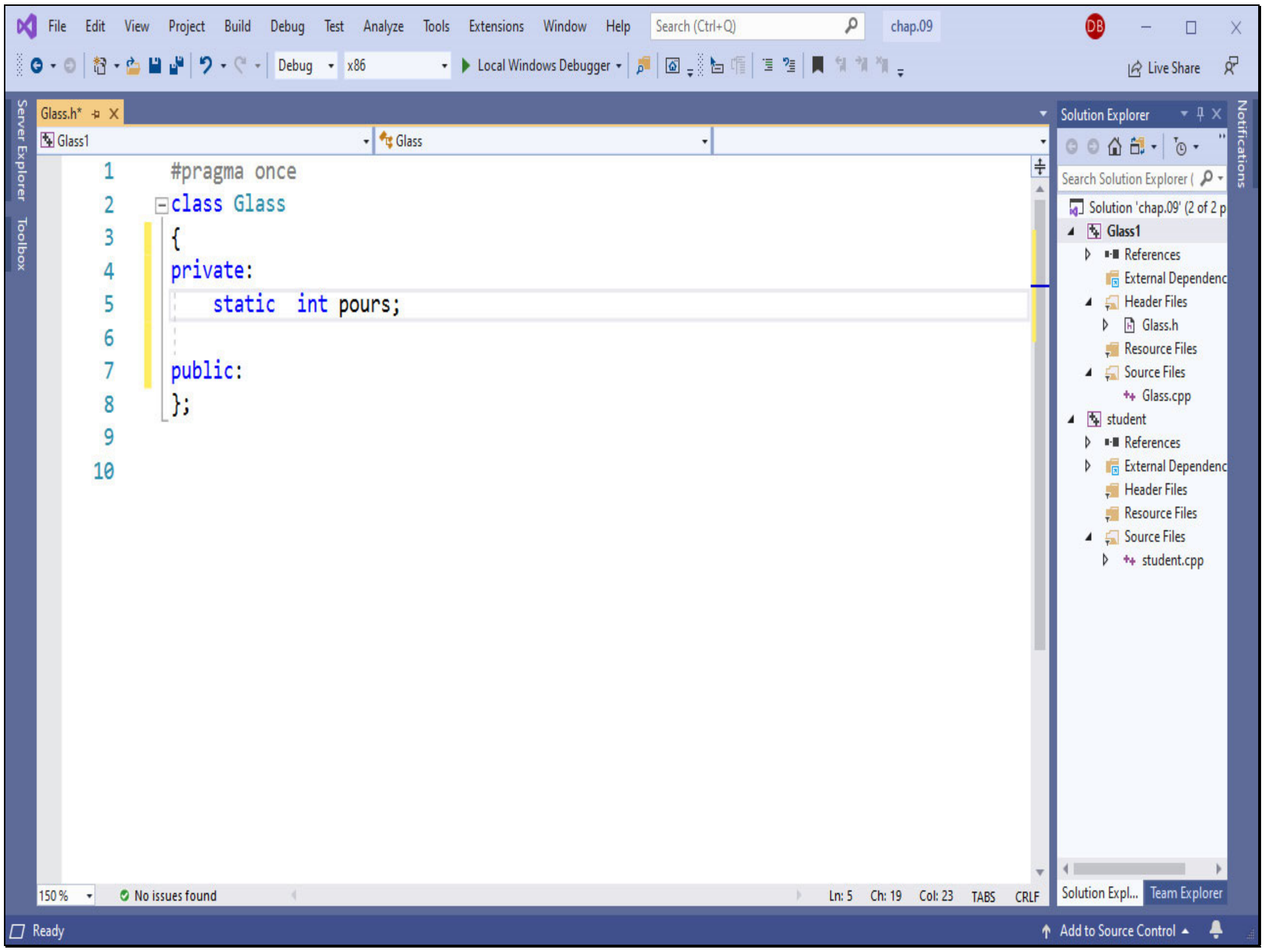


```
#pragma once
class Glass
{
private:

public:
};
```

Text Captions

We’ll need both a private and a public section.

Slide 11

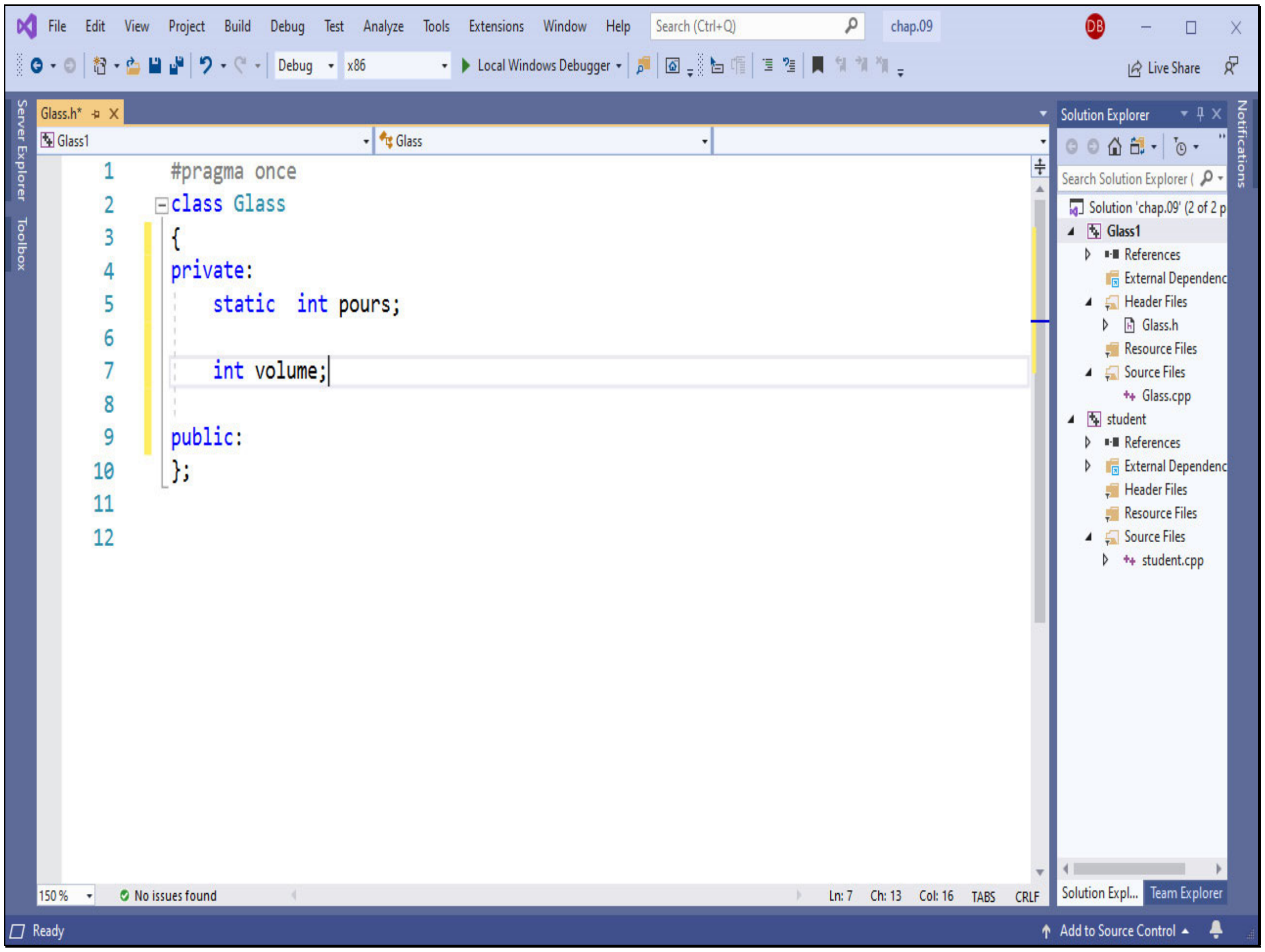


```
#pragma once
class Glass
{
private:
    static  int pours;

public:
};
```

Text Captions

One of the challenges of the game is to minimize the number of times that we must pour water from one glass to another to solve the puzzle. There are a total three glasses, but only two glasses are involved during each pouring operation. This fact makes it impossible to track the number of times that water is poured from one glass to another with a member variable in each Glass object. So, our approach is to create a single variable that is shared by all three Glass objects. We do this by making the variable “static,” which means that it is a class variable rather than an instance variable (that is, it’s a variable that is owned by the class as a whole rather than a variable that is owned by a single object). The UML class diagram denotes class or static variables by underling them.

Slide 12



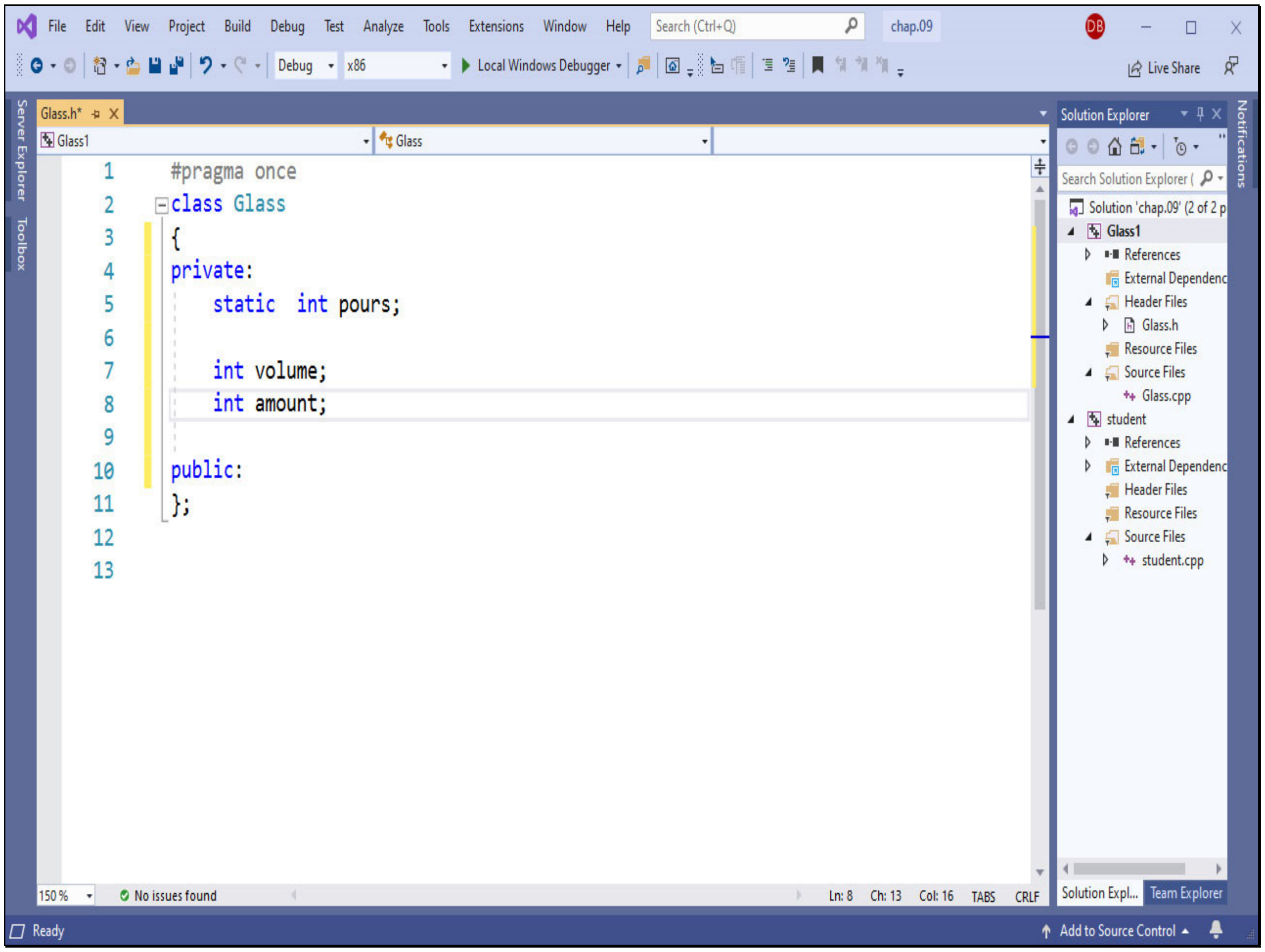
```
#pragma once
class Glass
{
private:
    static  int pours;

    int volume;

public:
};
```

Text Captions

From the results of the previous section, where we solved the problem and designed the program, we know that three values characterize each glass: the glass's total volume, the current amount of water in the glass, and the current amount of empty space in the glass. It's convenient to represent these values with member variables in the Glass class.

Given any two of these values, it's possible to calculate the third. So, our Glass class only needs two member variables to solve the pouring problem and I choose the total volume as the first.

Slide 13

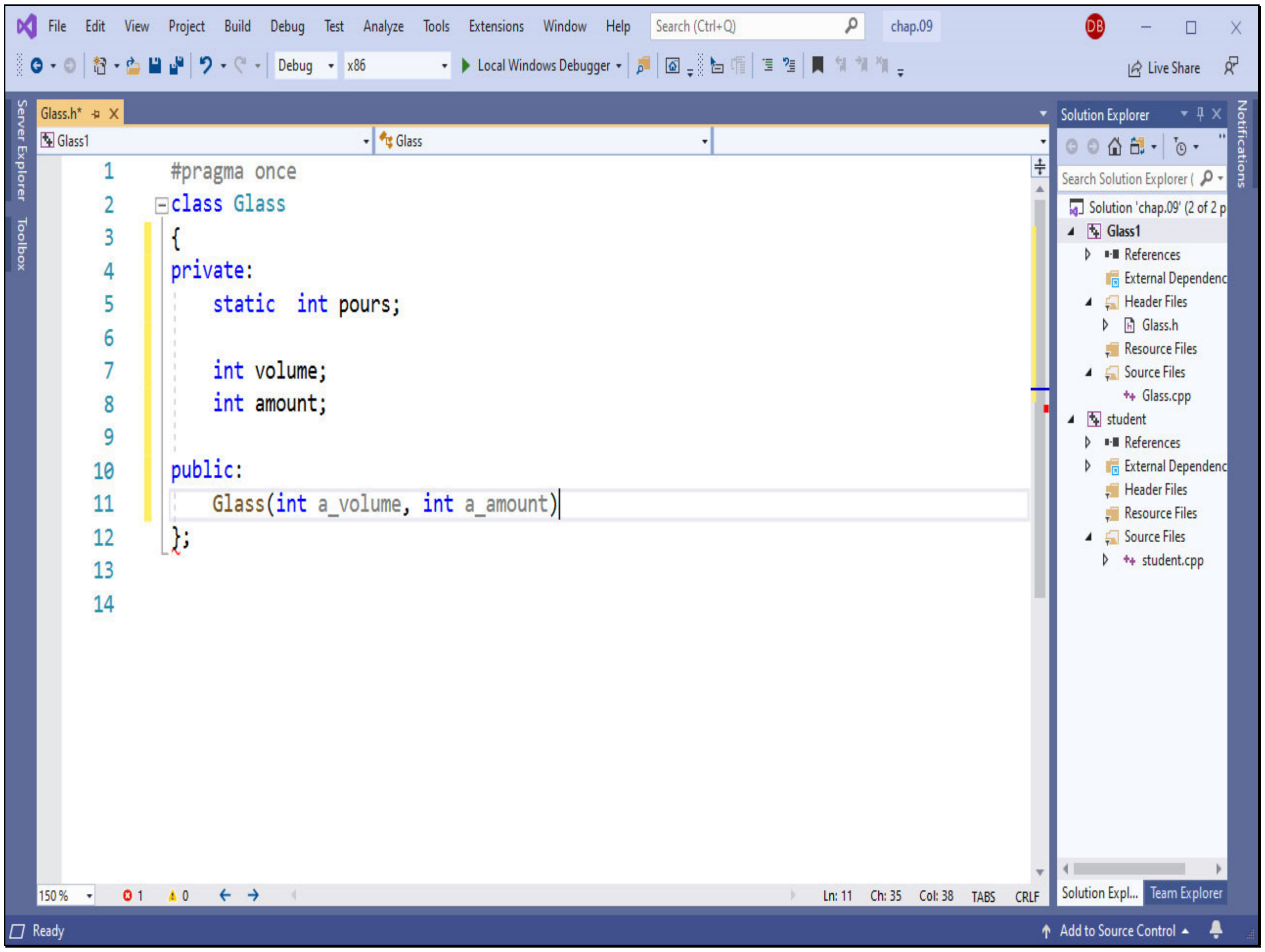


```
#pragma once
class Glass
{
private:
    static  int pours;

    int volume;
    int amount;

public:
};
```

Text Captions

For the second value or variable, I choose the current amount of water in the glass. While the choices are arbitrary, once they are made, the functions that follow rely on these variables. If we change the Glass member variables, then must also update the Glass functions.

Slide 14

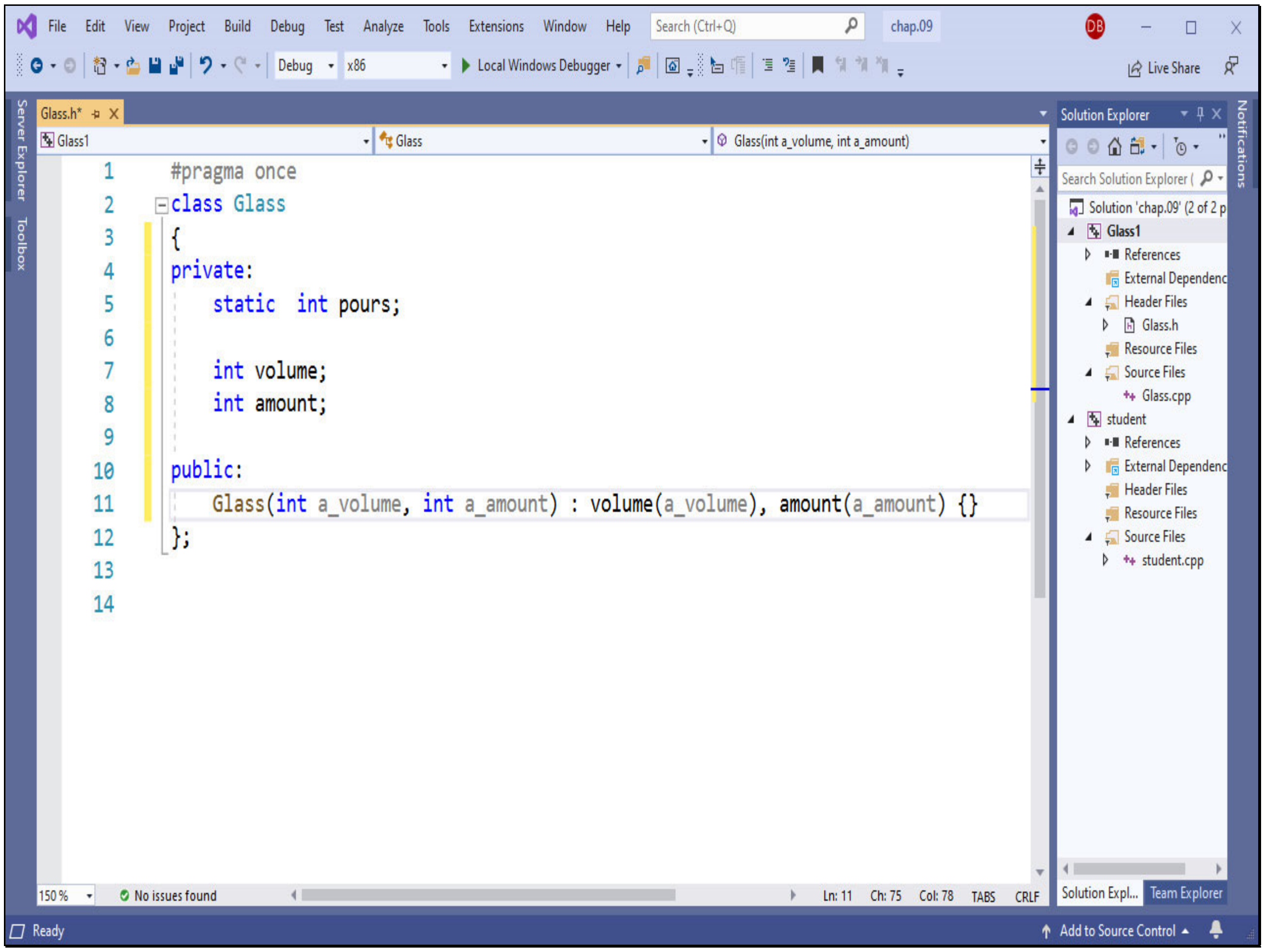


```
#pragma once
class Glass
{
private:
    static  int pours;

    int volume;
    int amount;

public:
    Glass(int a_volume, int a_amount)
};
```

Text Captions

Our next task is to develop the Glass member functions. I usually start with the constructor or constructors. We'll use the constructor to initialize the Glass's volume and the amount of water initially in the Glass as we instantiate each glass object.

Slide 15



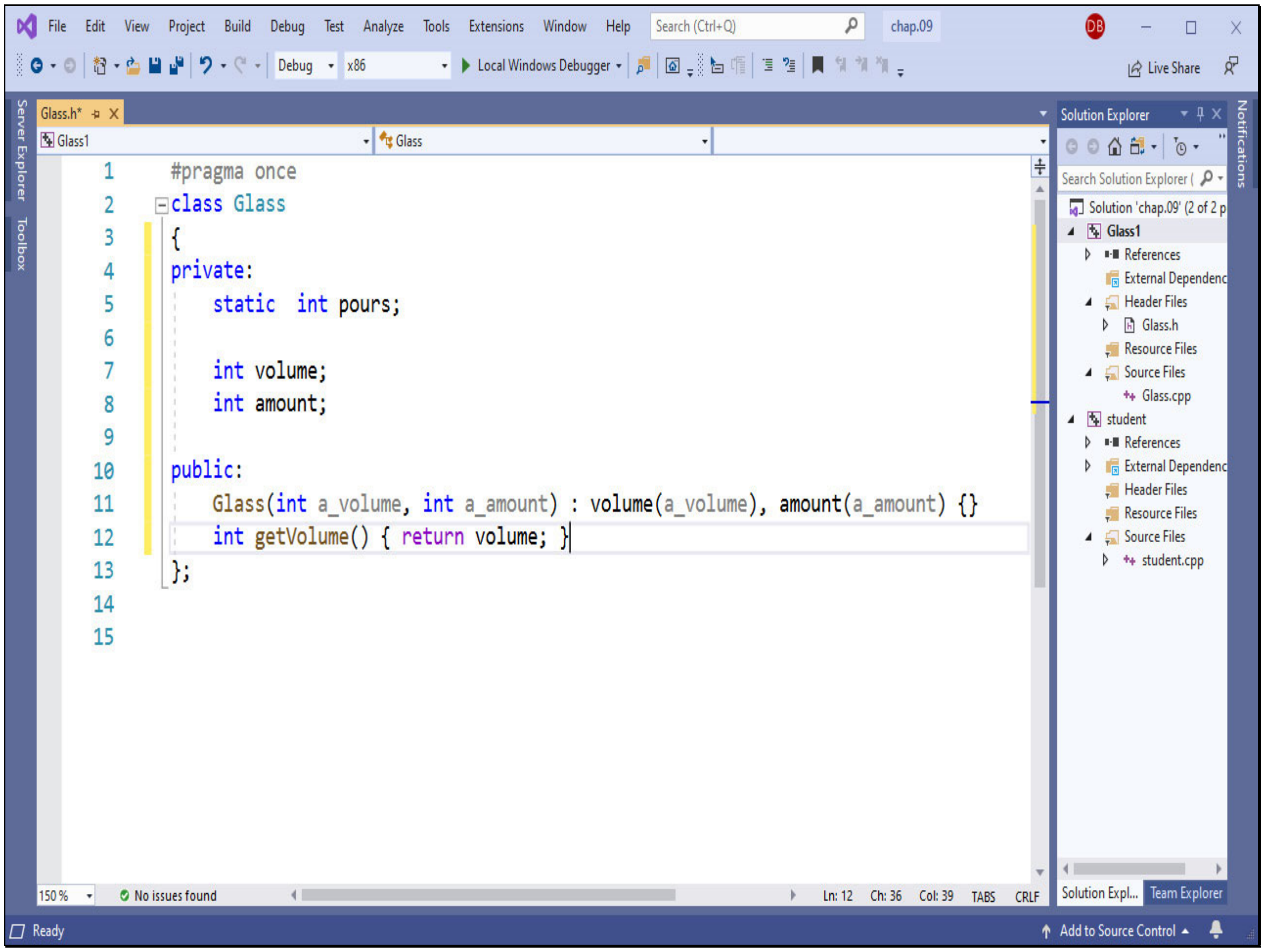
```cpp
#pragma once
class Glass
{
private:
    static  int pours;

    int volume;
    int amount;

public:
    Glass(int a_volume, int a_amount) : volume(a_volume), amount(a_amount) {}
};
```

Text Captions

This constructor is a very simple function and is a perfect place to use an initializer list. Recall that an initializer list can only be used with a constructor, begins with a colon, and has one element for each member variable. Each element consists of the member variable's name and its initial value, which in this example, is one of the constructor's arguments. The initial value is enclosed with parentheses.

Once the member variables are initialized, there are no remaining tasks for the constructor to do. So, we end the list with a pair of empty braces, which is the function's body.

Slide 16

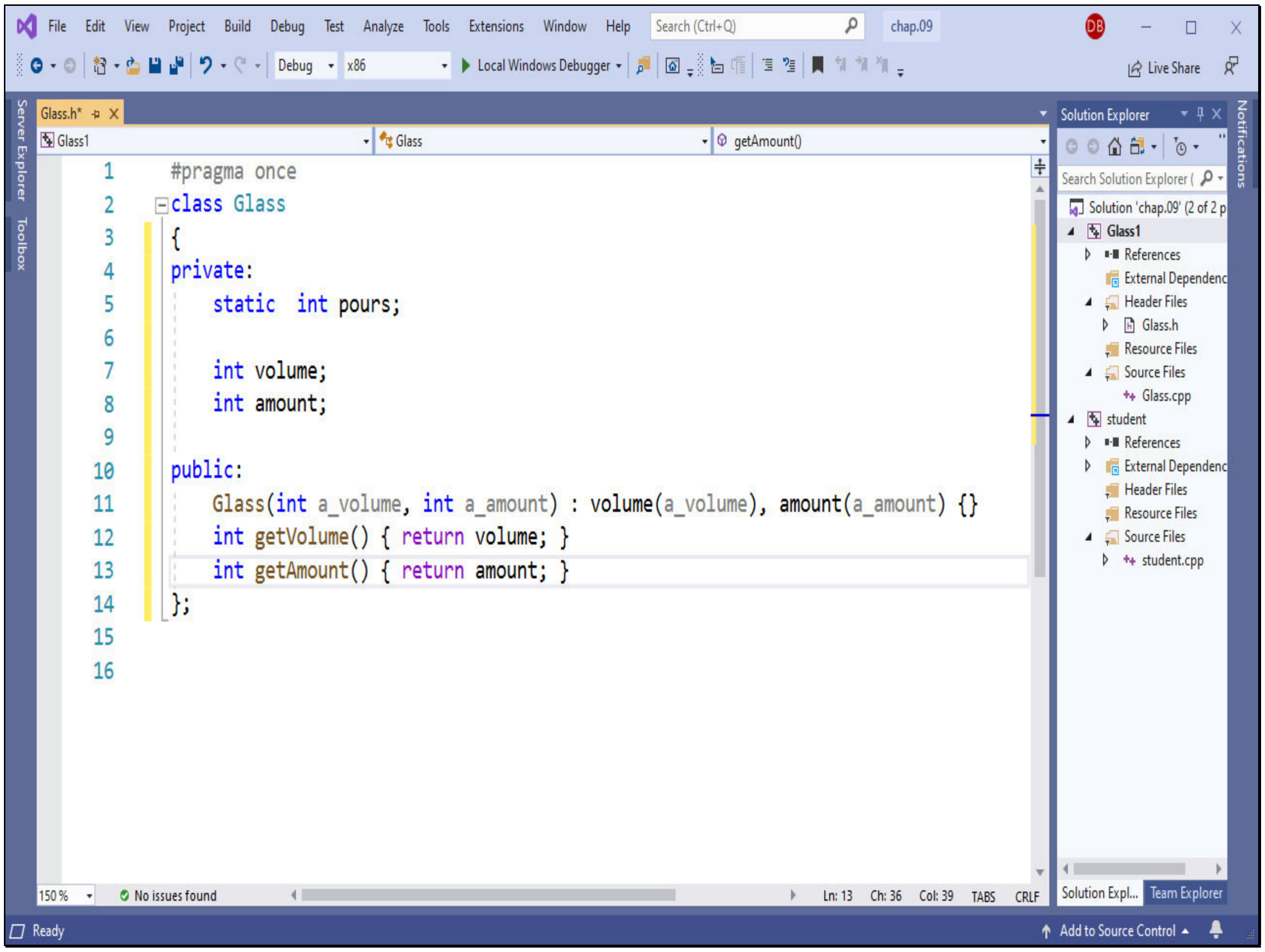


```
#pragma once
class Glass
{
private:
    static  int pours;

    int volume;
    int amount;

public:
    Glass(int a_volume, int a_amount) : volume(a_volume), amount(a_amount) {}
    int getVolume() { return volume; }
};
```

Text Captions

It’s often convenient to have getter functions for some of the member variables and this is the case for the overall puzzle or game that we are creating. Getters are typically very simple functions as illustrated by getVolume.

Slide 17

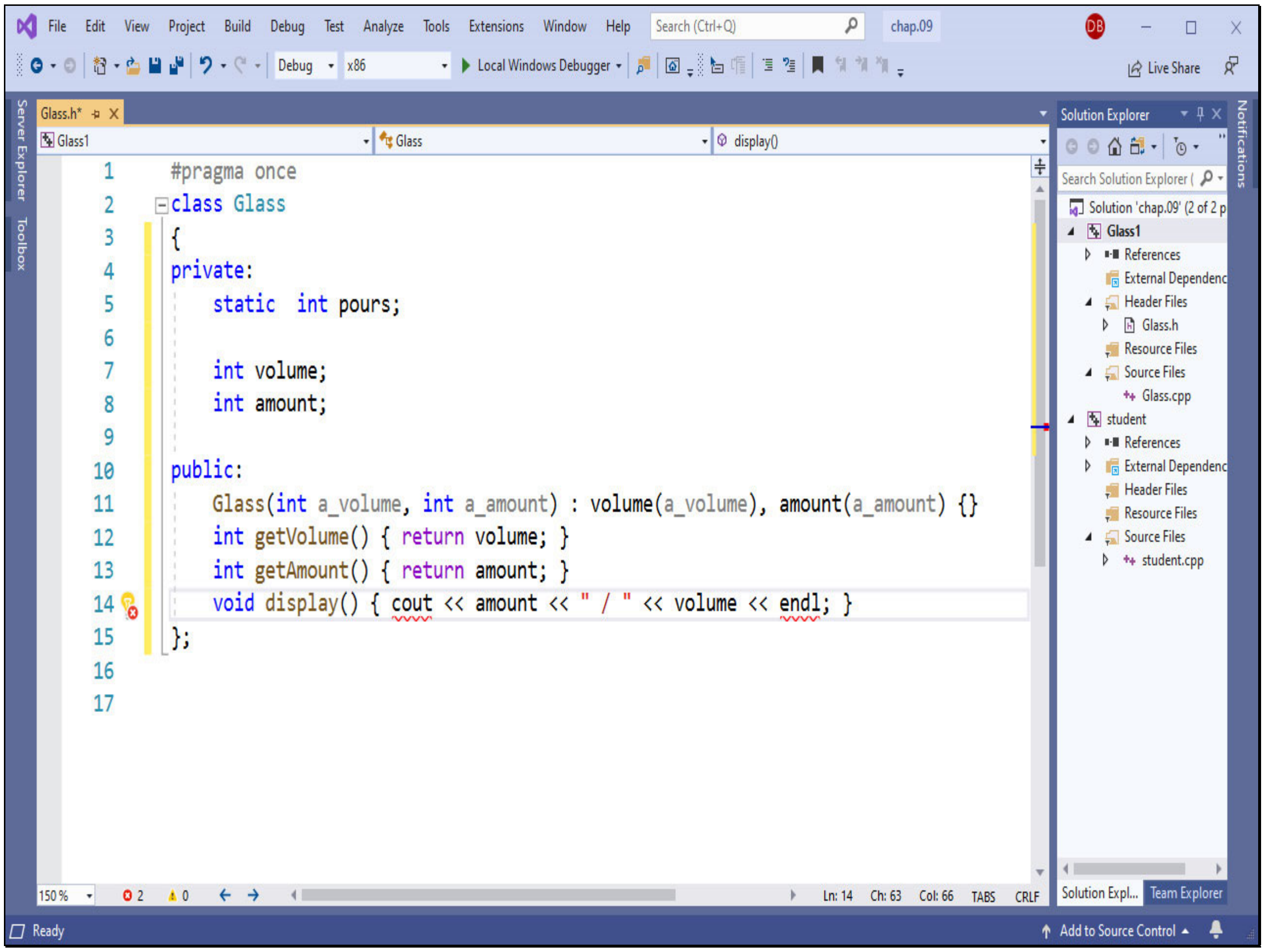


```
#pragma once
class Glass
{
private:
    static  int pours;

    int volume;
    int amount;

public:
    Glass(int a_volume, int a_amount) : volume(a_volume), amount(a_amount) {}
    int getVolume() { return volume; }
    int getAmount() { return amount; }
};
```

Text Captions

A second getter function, getAmount, will allow users to see the amount of water currently stored in the Glass object.

Slide 18



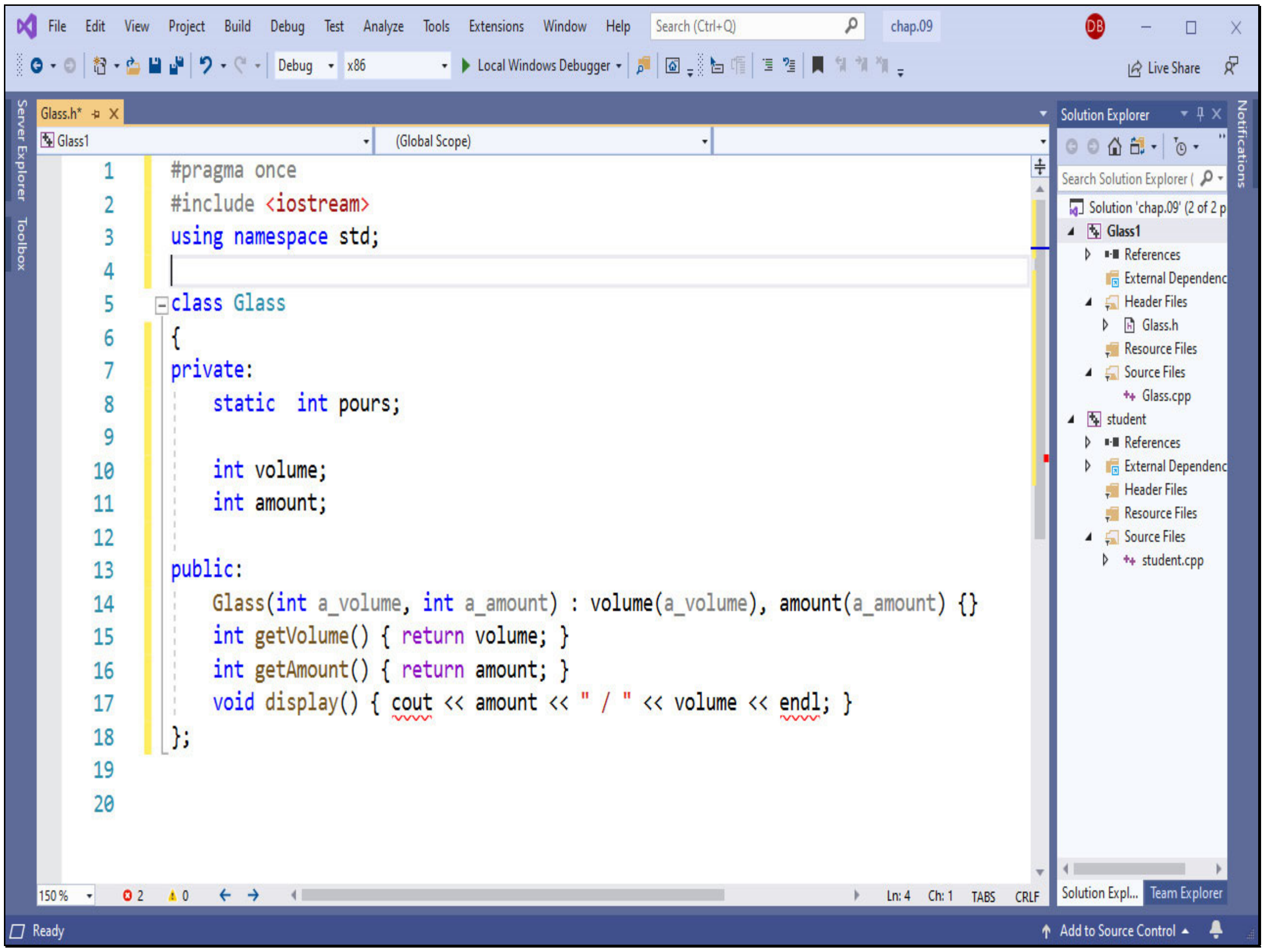
```
#pragma once
class Glass
{
private:
    static  int pours;

    int volume;
    int amount;

public:
    Glass(int a_volume, int a_amount) : volume(a_volume), amount(a_amount) {}
    int getVolume() { return volume; }
    int getAmount() { return amount; }
    void display() { cout << amount << " / " << volume << endl; }
};
```

Text Captions

The program really doesn’t need a display function: we can get the same information from the two getter functions, but it is a convenient function to have.

Notice that Intellisense is flagging “cout” as an error. The standard mantra is, “When you use a feature, #include the corresponding header file.”

Slide 19

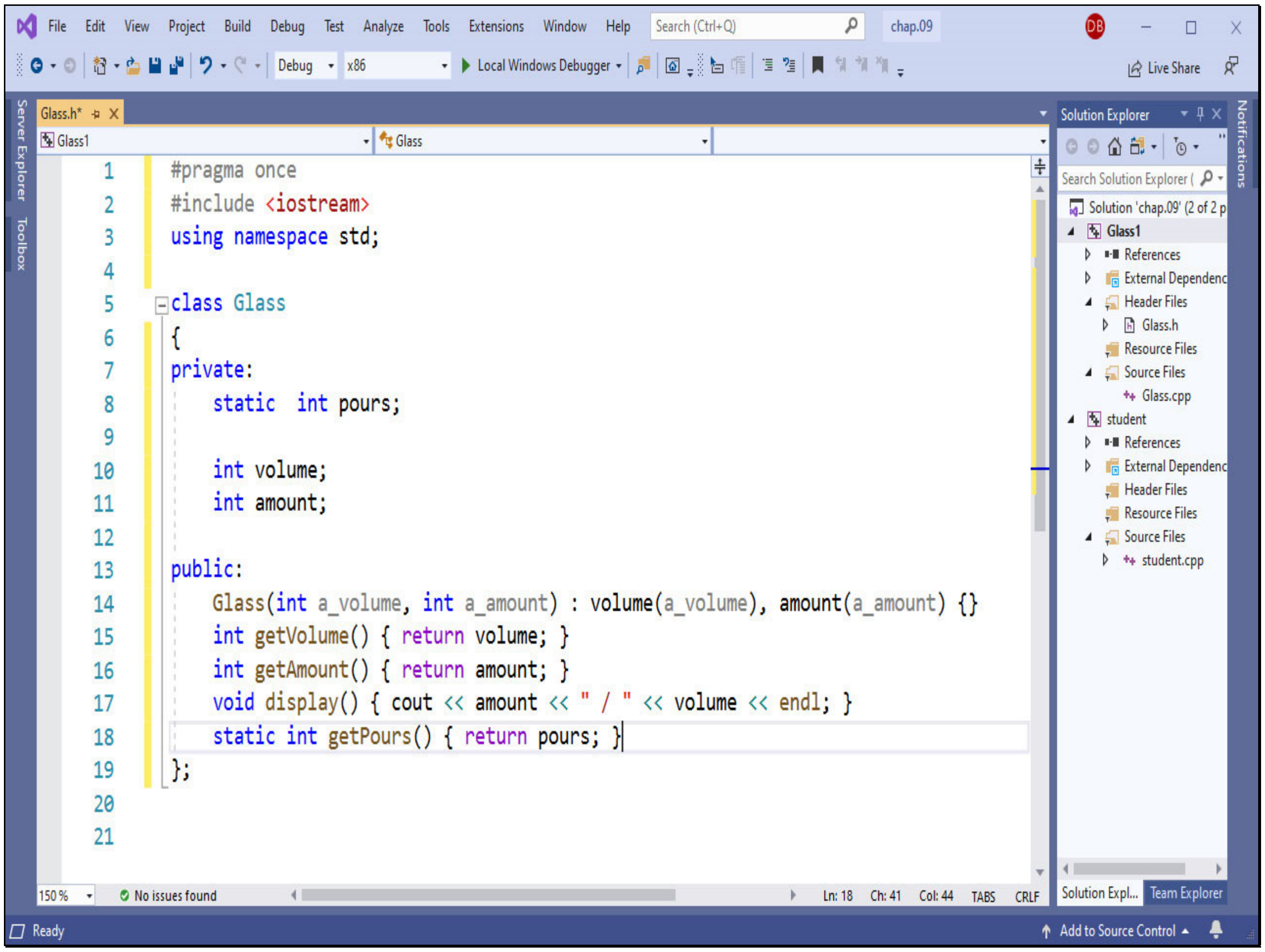


```
1   #pragma once
2   #include <iostream>
3   using namespace std;
4   
5   class Glass
6   {
7   private:
8       static  int pours;
9   
10      int volume;
11      int amount;
12  
13  public:
14      Glass(int a_volume, int a_amount) : volume(a_volume), amount(a_amount) {}
15      int getVolume() { return volume; }
16      int getAmount() { return amount; }
17      void display() { cout << amount << " / " << volume << endl; }
18  };
19  
20  
```

Text Captions

We correct the problem by adding the #include <iostream> directive and the using namespace statement at the top of the file, above the class specification. The red underlining will go away once we click the mouse at the end of line 17 or beyond.

Slide 20

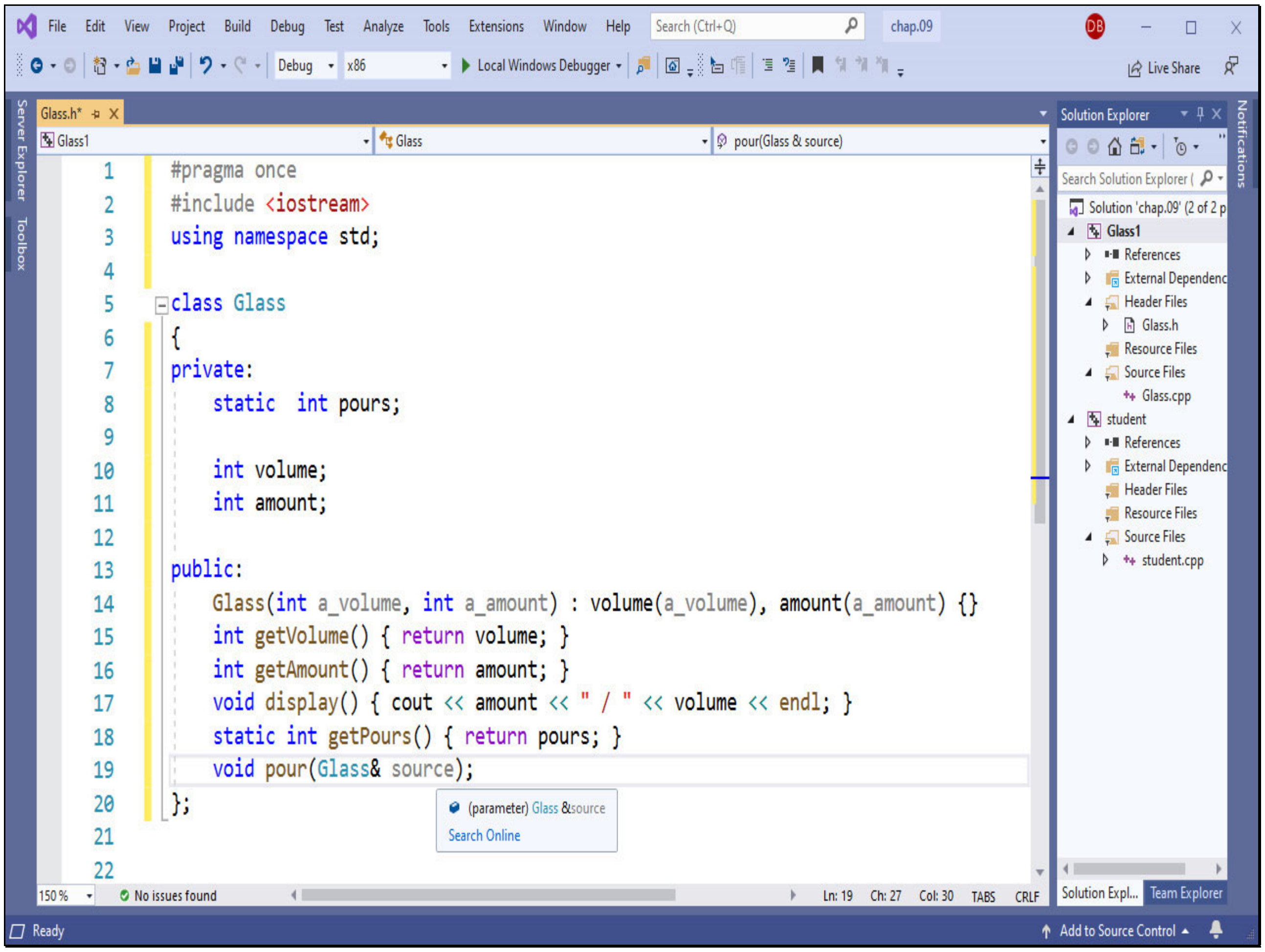


```
#pragma once
#include <iostream>
using namespace std;

class Glass
{
private:
    static  int pours;

    int volume;
    int amount;

public:
    Glass(int a_volume, int a_amount) : volume(a_volume), amount(a_amount) {}
    int getVolume() { return volume; }
    int getAmount() { return amount; }
    void display() { cout << amount << " / " << volume << endl; }
    static int getPours() { return pours; }
};
```

Text Captions

The function getPours is just another getter function. However, the pours variable is a class variable and not a member variable – or said another way, pours is a static variable. That means that the getter function must also be static. We'll see later how that impacts the way that we call the function.

Slide 21

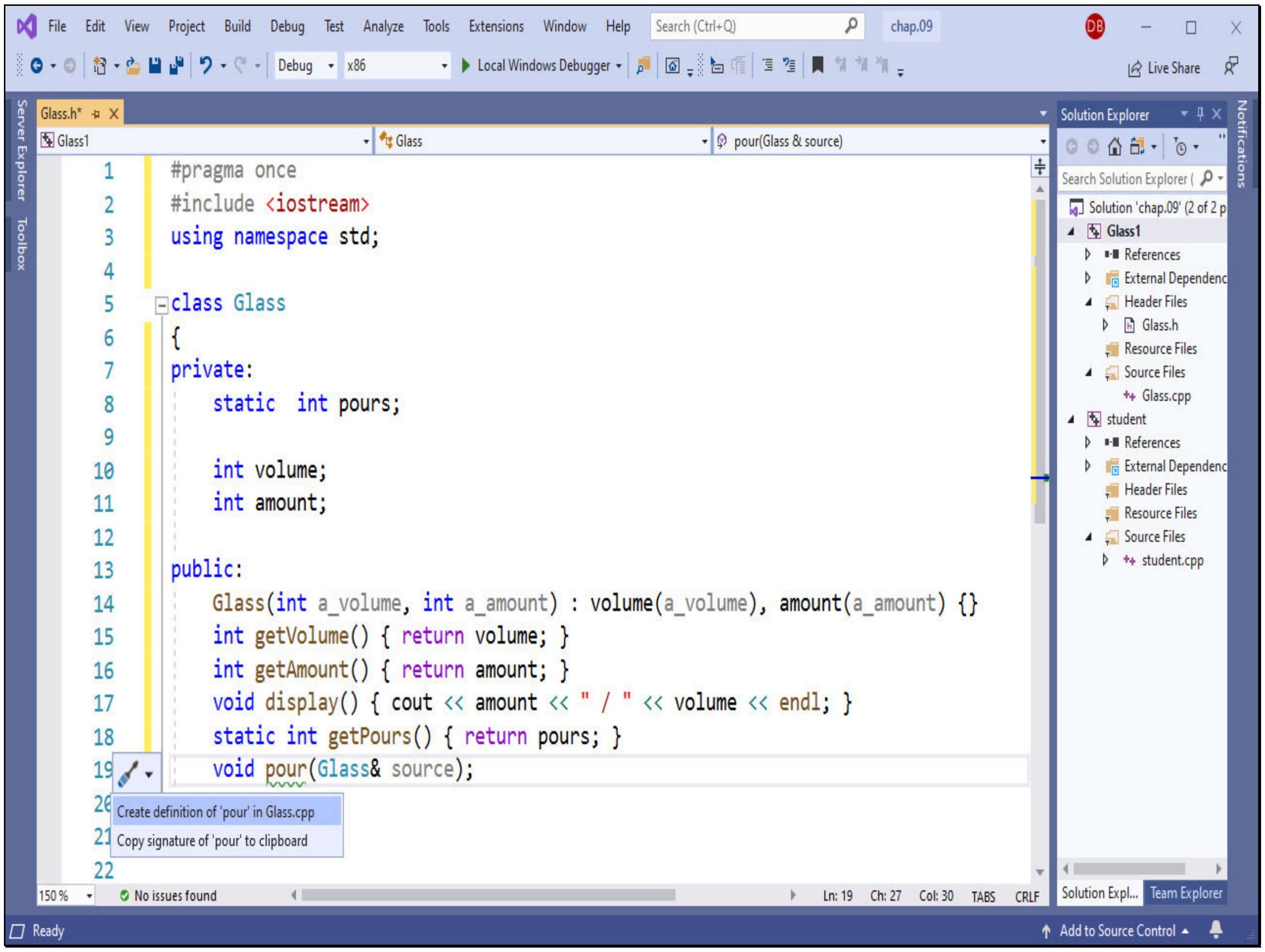


```
#pragma once
#include <iostream>
using namespace std;

class Glass
{
private:
    static  int pours;

    int volume;
    int amount;

public:
    Glass(int a_volume, int a_amount) : volume(a_volume), amount(a_amount) {}
    int getVolume() { return volume; }
    int getAmount() { return amount; }
    void display() { cout << amount << " / " << volume << endl; }
    static int getPours() { return pours; }
    void pour(Glass& source);
};
```

Text Captions

Finally, we add the pour function, which is the only complex member function in the program. Since it is a larger function, we'll only prototype it in the class and define it in a separate source code or .cpp file.

Slide 22

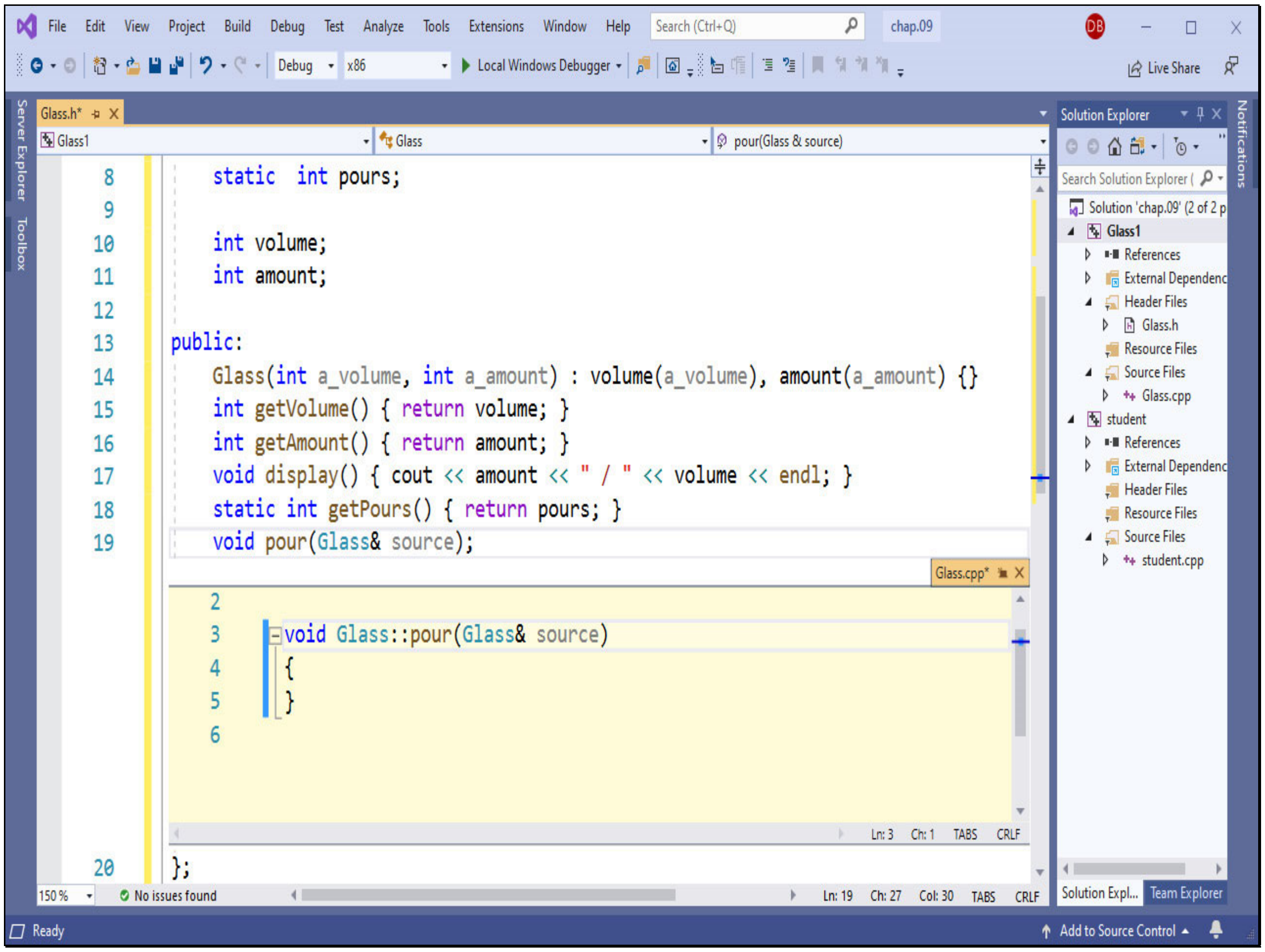


```
#pragma once
#include <iostream>
using namespace std;

class Glass
{
private:
    static  int pours;

    int volume;
    int amount;

public:
    Glass(int a_volume, int a_amount) : volume(a_volume), amount(a_amount) {}
    int getVolume() { return volume; }
    int getAmount() { return amount; }
    void display() { cout << amount << " / " << volume << endl; }
    static int getPours() { return pours; }
    void pour(Glass& source);
```

Text Captions

The newest version of the Visual Studio editor has a neat feature that speeds navigation just a little. After finishing the prototype, the editor displays a little screwdriver on the left side – you may need to wait just a moment for it to appear. If we click on the down-arrow next to the screwdriver, we'll see two options: one option is to copy the function's signature or prototype to the clipboard, which we can use to begin defining the function elsewhere. The other option is to outline the function in Glass.cpp. Let's select that option.

Slide 23



```
8      static  int pours;
9
10     int volume;
11     int amount;
12
13  public:
14     Glass(int a_volume, int a_amount) : volume(a_volume), amount(a_amount) {}
15     int getVolume() { return volume; }
16     int getAmount() { return amount; }
17     void display() { cout << amount << " / " << volume << endl; }
18     static int getPours() { return pours; }
19     void pour(Glass& source);
```

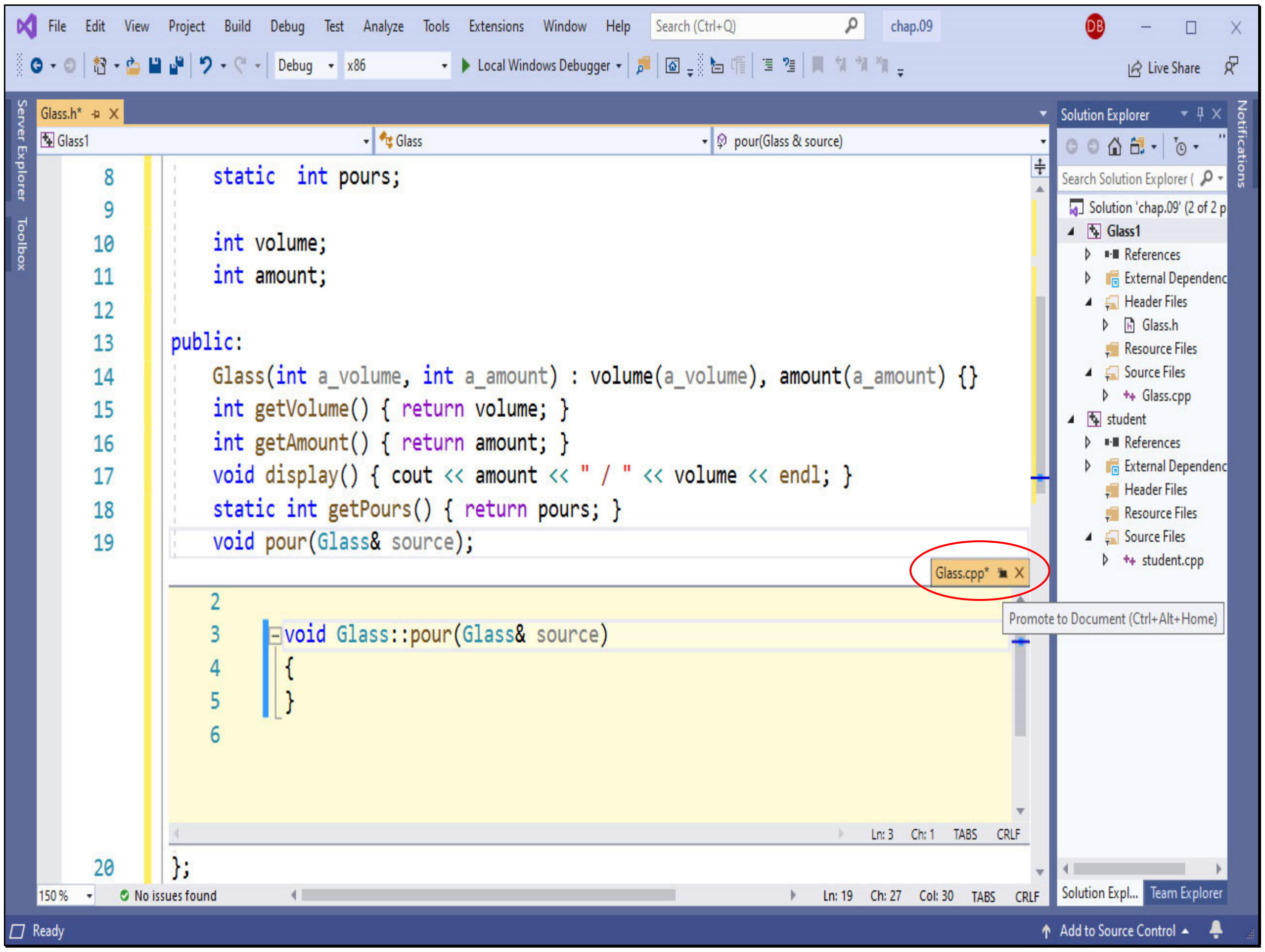
```
2
3   void Glass::pour(Glass& source)
4   {
5   }
6
```

```
20  };
```

Text Captions

It looks like this little yellow window is opening in Glass.h, but notice the line numbers and the name Glass.cpp on the tab. Visual Studio has written this outline of the pour function in Glass.cpp. While we can edit the function in the yellow window . . .

Slide 24



```
    static  int pours;

    int volume;
    int amount;

public:
    Glass(int a_volume, int a_amount) : volume(a_volume), amount(a_amount) {}
    int getVolume() { return volume; }
    int getAmount() { return amount; }
    void display() { cout << amount << " / " << volume << endl; }
    static int getPours() { return pours; }
    void pour(Glass& source);
```

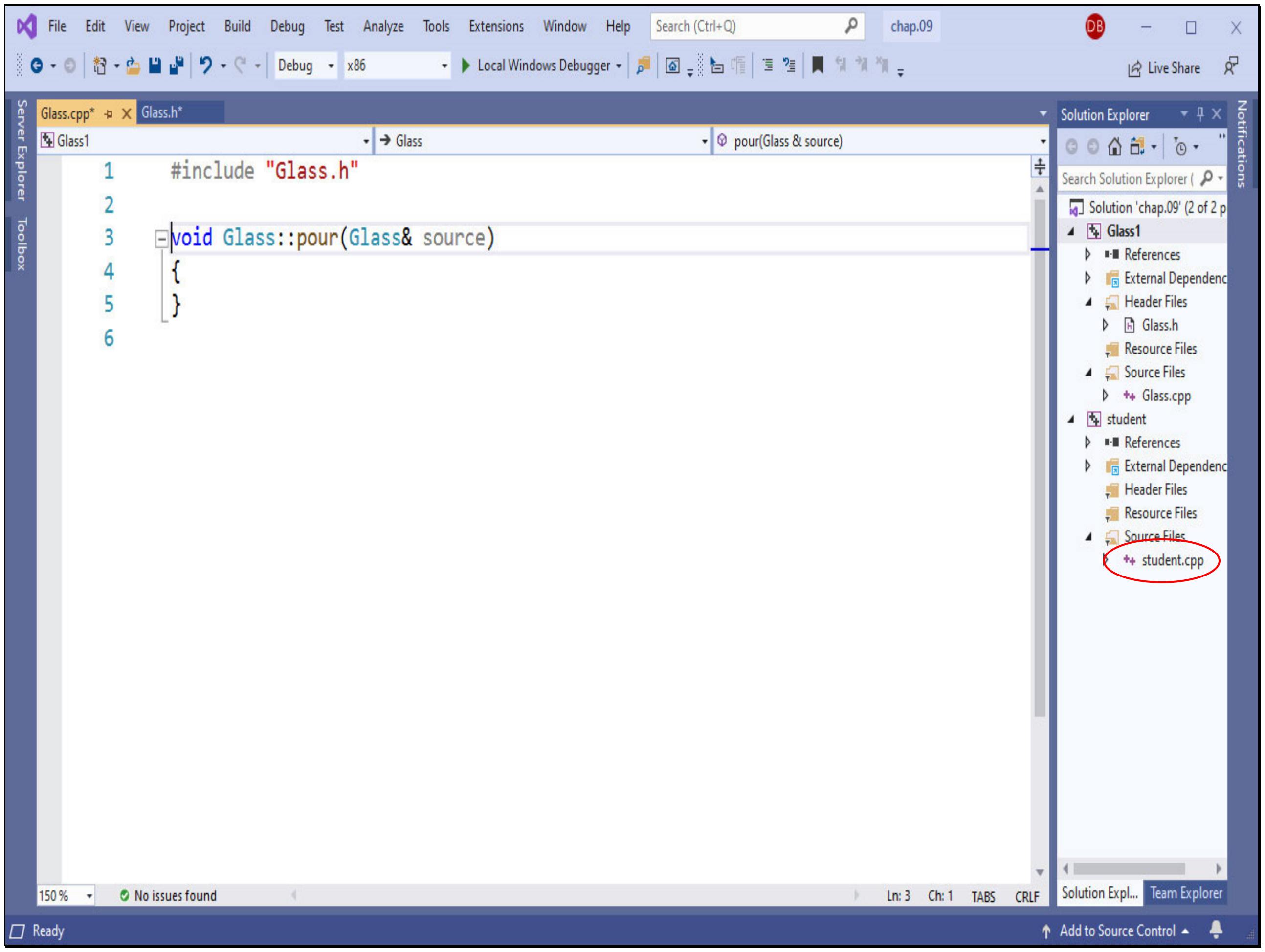
```
void Glass::pour(Glass& source)
{
}
```

```
};
```

Text Captions

I prefer - and that's all it is, a personal preference – to edit the function in a "normal" editor window. So, click the "x" button on the Glass.cpp tab to close the yellow window.

Slide 25



```
1   #include "Glass.h"
2
3   void Glass::pour(Glass& source)
4   {
5   }
6
```

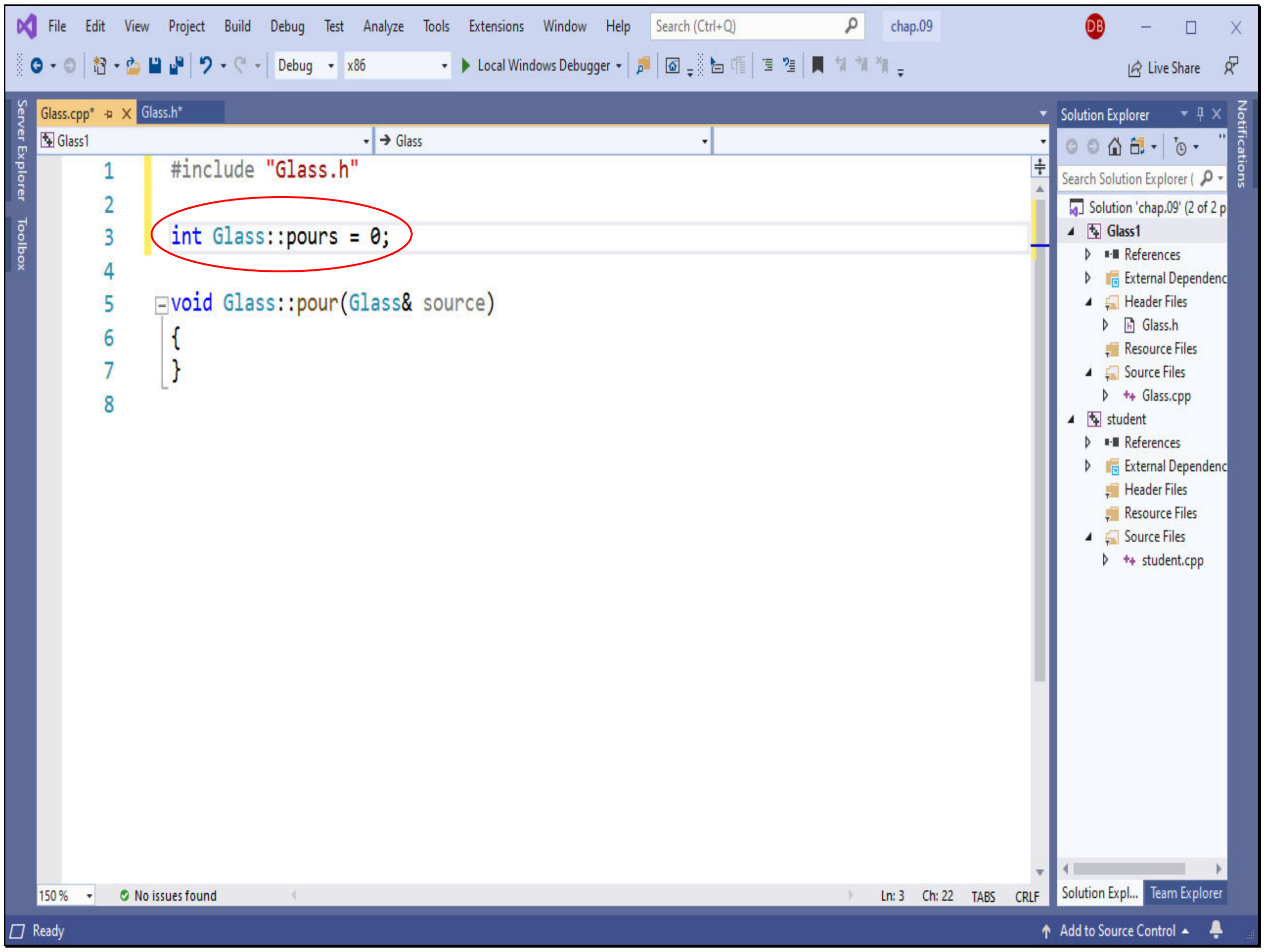
Text Captions

Find Glass.cpp in the solution explorer and double-click the name. This opens the file in a standard editor window, where we can see the function outline created just few moments ago. Notice the class name, “Glass,” followed by the scope resolution operator on line 3. This is how the compiler knows that this function is a part of the Glass class.

This function will pour or transfer water between two glass objects: the source and the destination. The role played by each glass object is determined by its position the function call:

```
destination.pour(source);
```

Slide 26



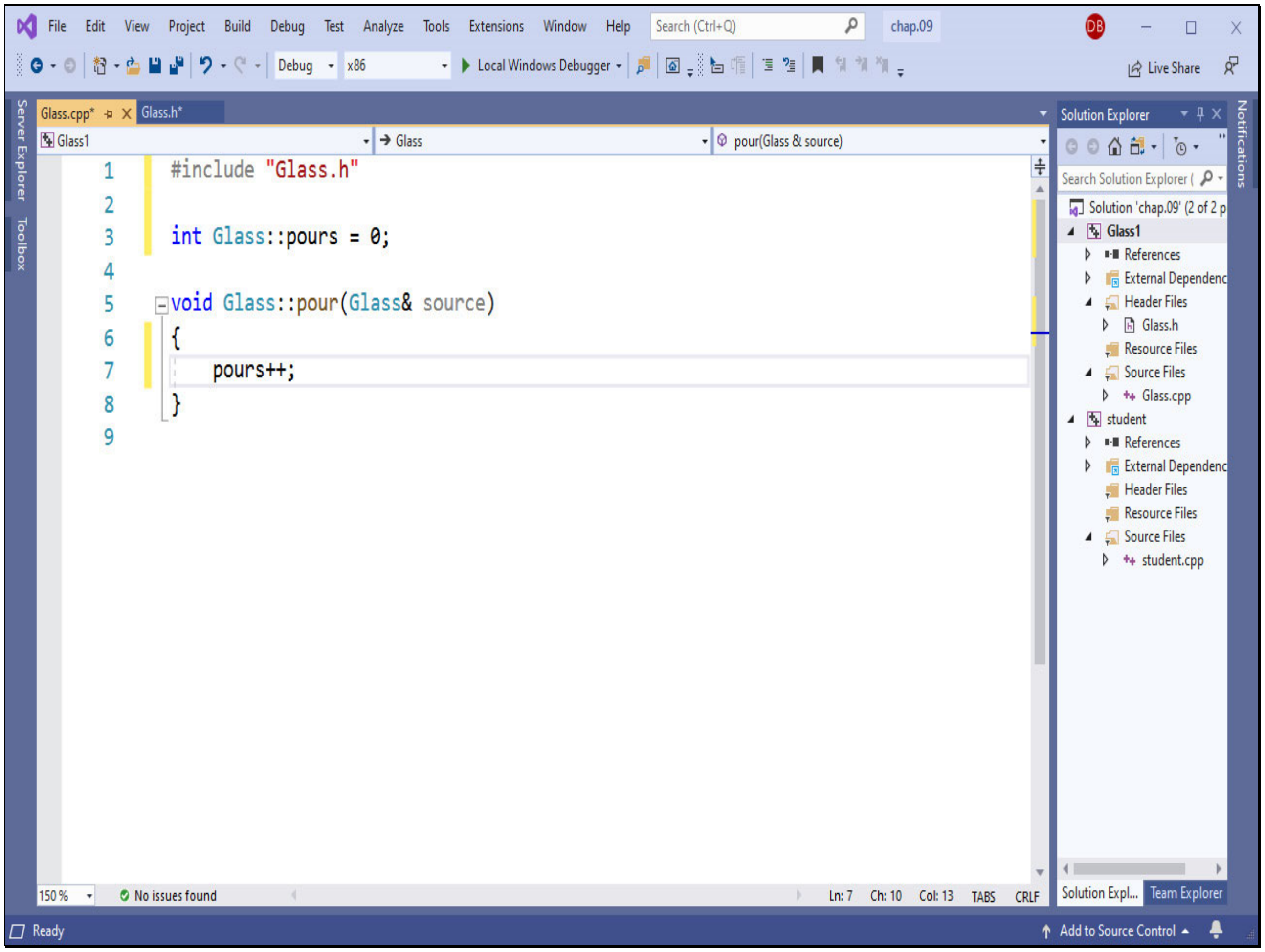
```
1  #include "Glass.h"
2
3  int Glass::pours = 0;
4
5  void Glass::pour(Glass& source)
6  {
7  }
8
```

Text Captions

Lest we forget, there is one task that we should do before completing the pour function. We need to initialize the static or class variable pours. On line 3, this looks very much like a dreaded global variable, but the class name and scope resolution operator tie this variable solidly to the Glass class. That is, it reduces the variable’s scope to the Glass class.

This example demonstrates the syntax for initializing a static variable in C++. Java has a somewhat cleaner syntax for doing this.

Slide 27

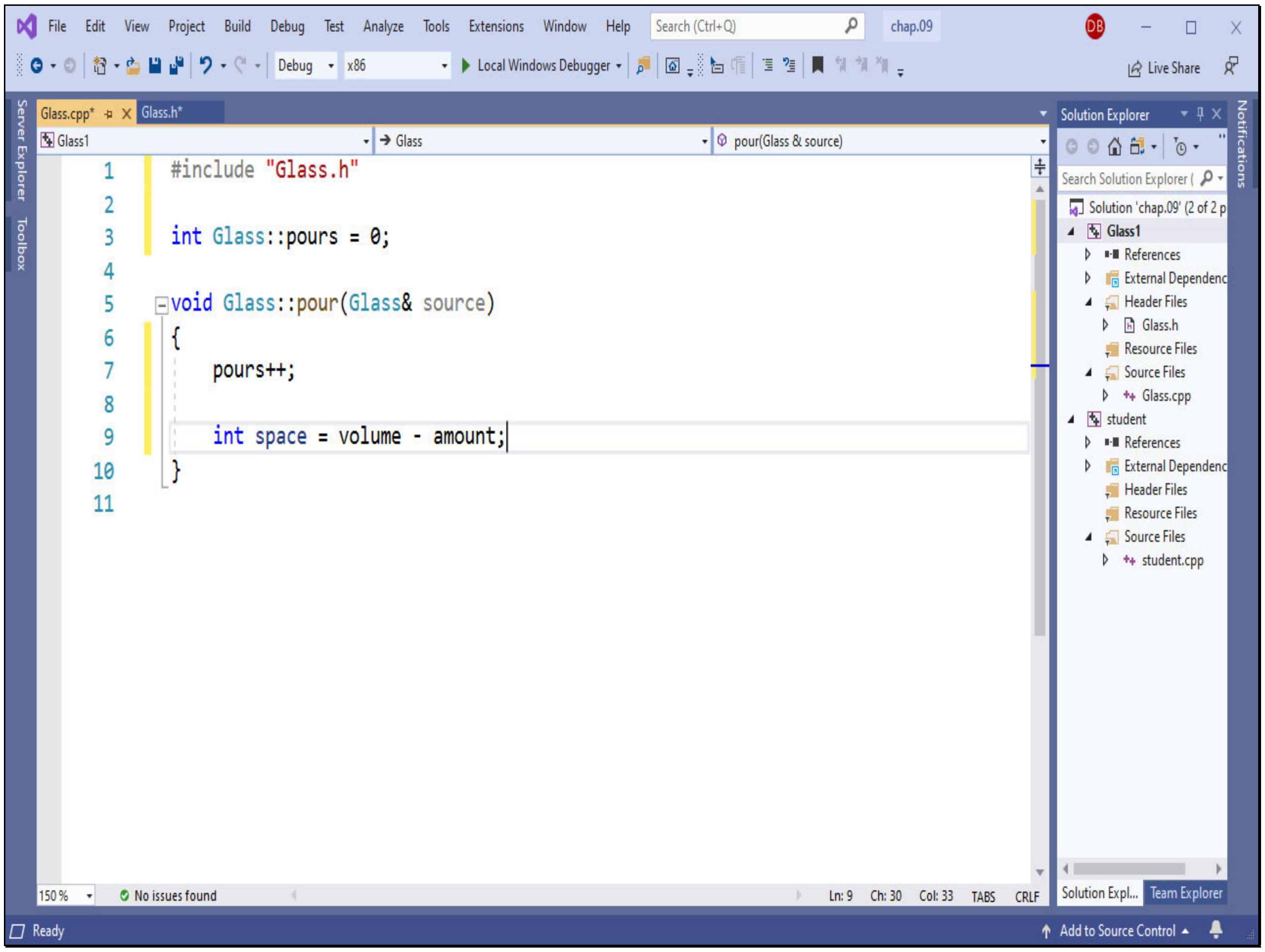


```
#include "Glass.h"

int Glass::pours = 0;

void Glass::pour(Glass& source)
{
    pours++;
}

```

Text Captions

Our goal is to minimize the number of times that we must pour water from one glass to another. So, we must count each pour operation, which we do by incrementing the pours counter whenever the pour function runs.

Slide 28



```
#include "Glass.h"

int Glass::pours = 0;

void Glass::pour(Glass& source)
{
    pours++;

    int space = volume - amount;
}
```

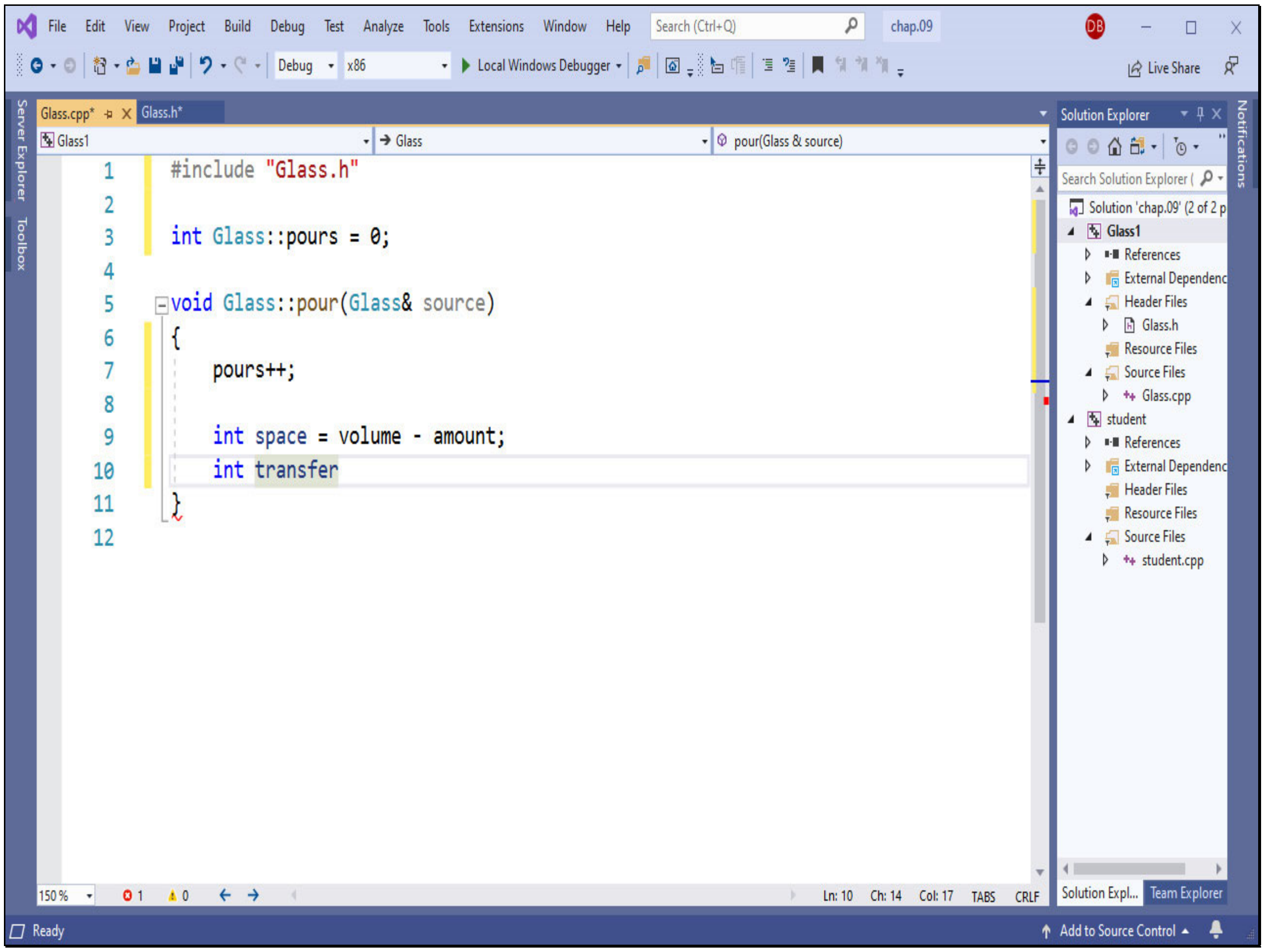
Text Captions

The rest of the pours function can be divided into two parts:

- First, calculate the maximum amount of water that can be poured or transferred from the source glass to the destination glass, and
- Second, update the two Glass objects to complete the pouring operation.

Both problems were solved in the previous section, and the solution implemented here begins by calculating how much space is available in the destination glass.

Slide 29



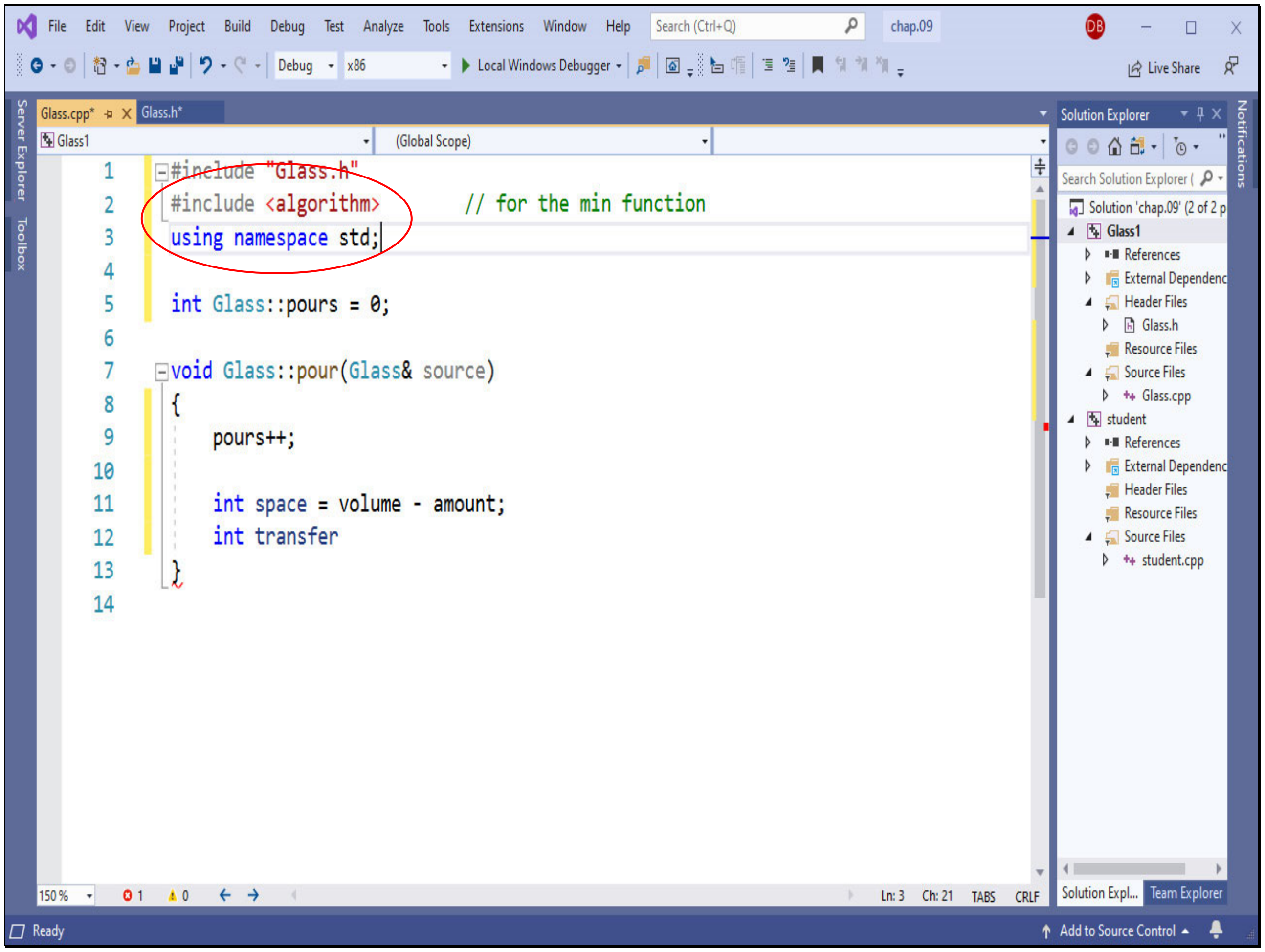
```
#include "Glass.h"

int Glass::pours = 0;

void Glass::pour(Glass& source)
{
    pours++;

    int space = volume - amount;
    int transfer
}
```

Text Captions

The next step is calculating how much water to pour or transfer from the source to the destination glass. We can’t pour more water than there is in the source glass, nor can we pour more water than there is space available in the destination glass. So, the amount that we can pour is the smallest or minimum of the space in the destination and the amount in the source.

C++ has a library function named min that we can use to select the smallest or minimum of these two values.

Slide 30

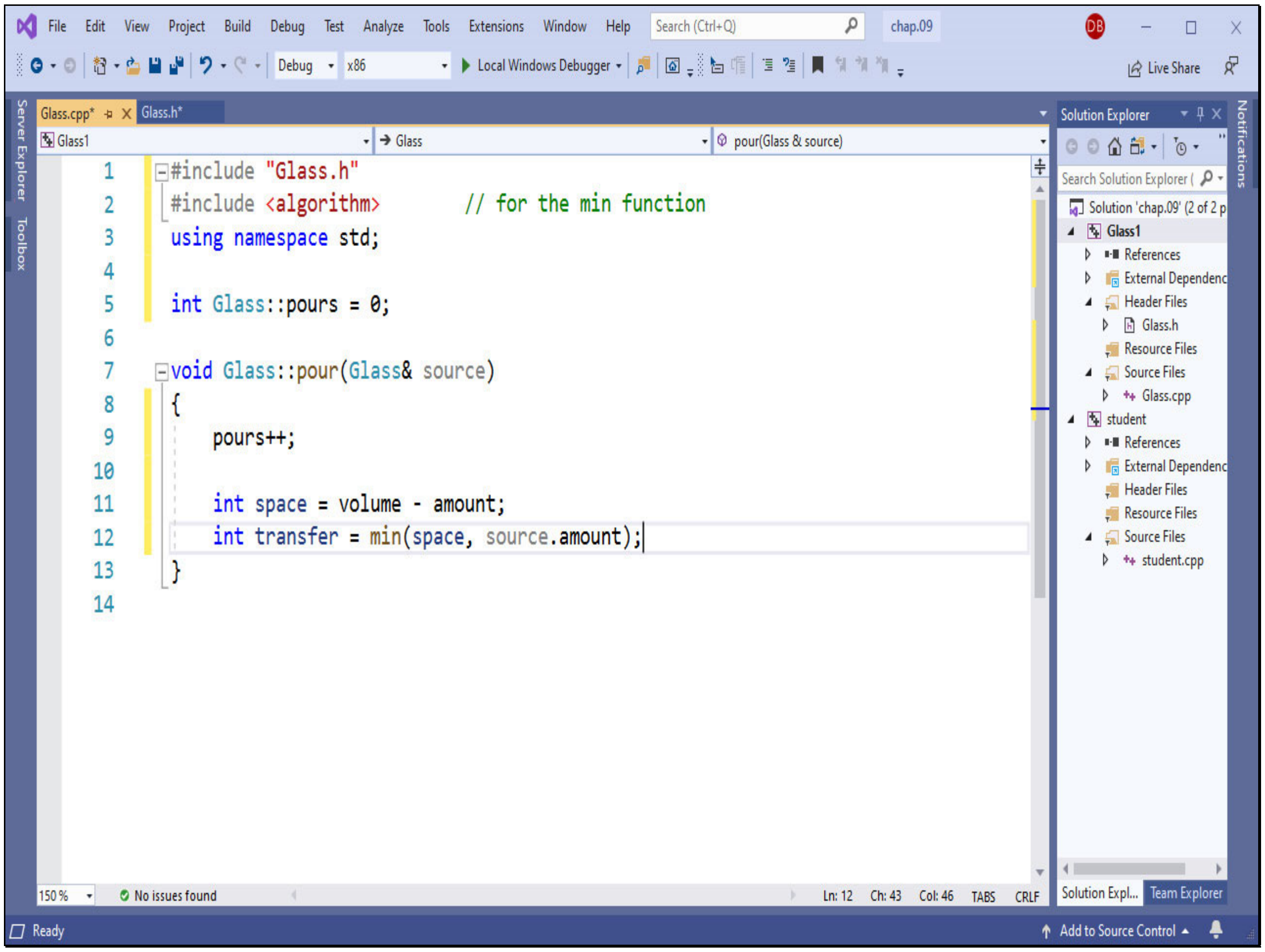


```
#include "Glass.h"
#include <algorithm>        // for the min function
using namespace std;

int Glass::pours = 0;

void Glass::pour(Glass& source)
{
    pours++;

    int space = volume - amount;
    int transfer
}

```

Text Captions

To use the min function, we need to #include the <algorithm> header file and add the using namespace statement.

Slide 31

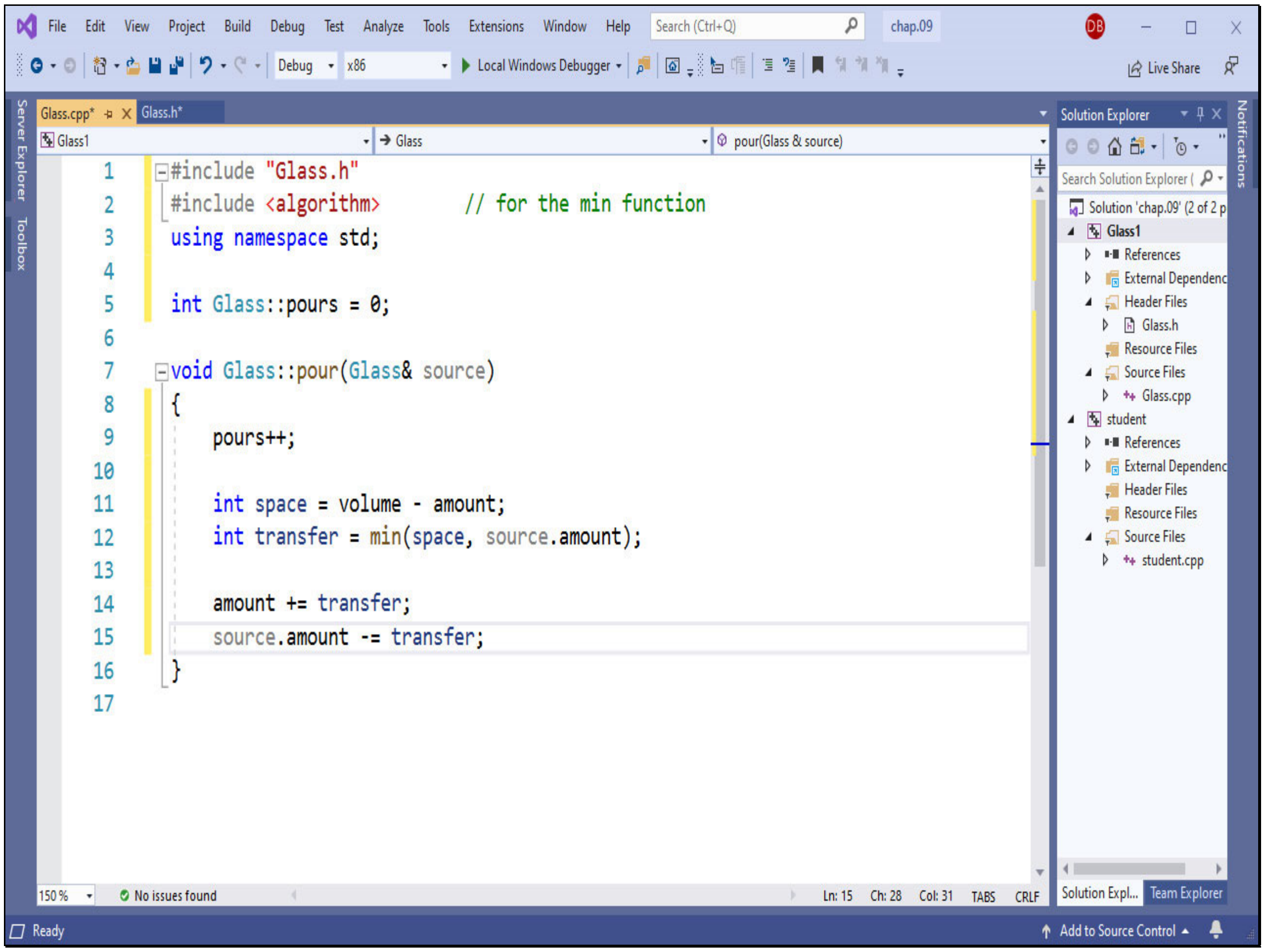


```
#include "Glass.h"
#include <algorithm>        // for the min function
using namespace std;

int Glass::pours = 0;

void Glass::pour(Glass& source)
{
    pours++;

    int space = volume - amount;
    int transfer = min(space, source.amount);
}
```

Text Captions

With the min function, it’s easy to calculate the amount of water to transfer from the source to the destination.

Slide 32



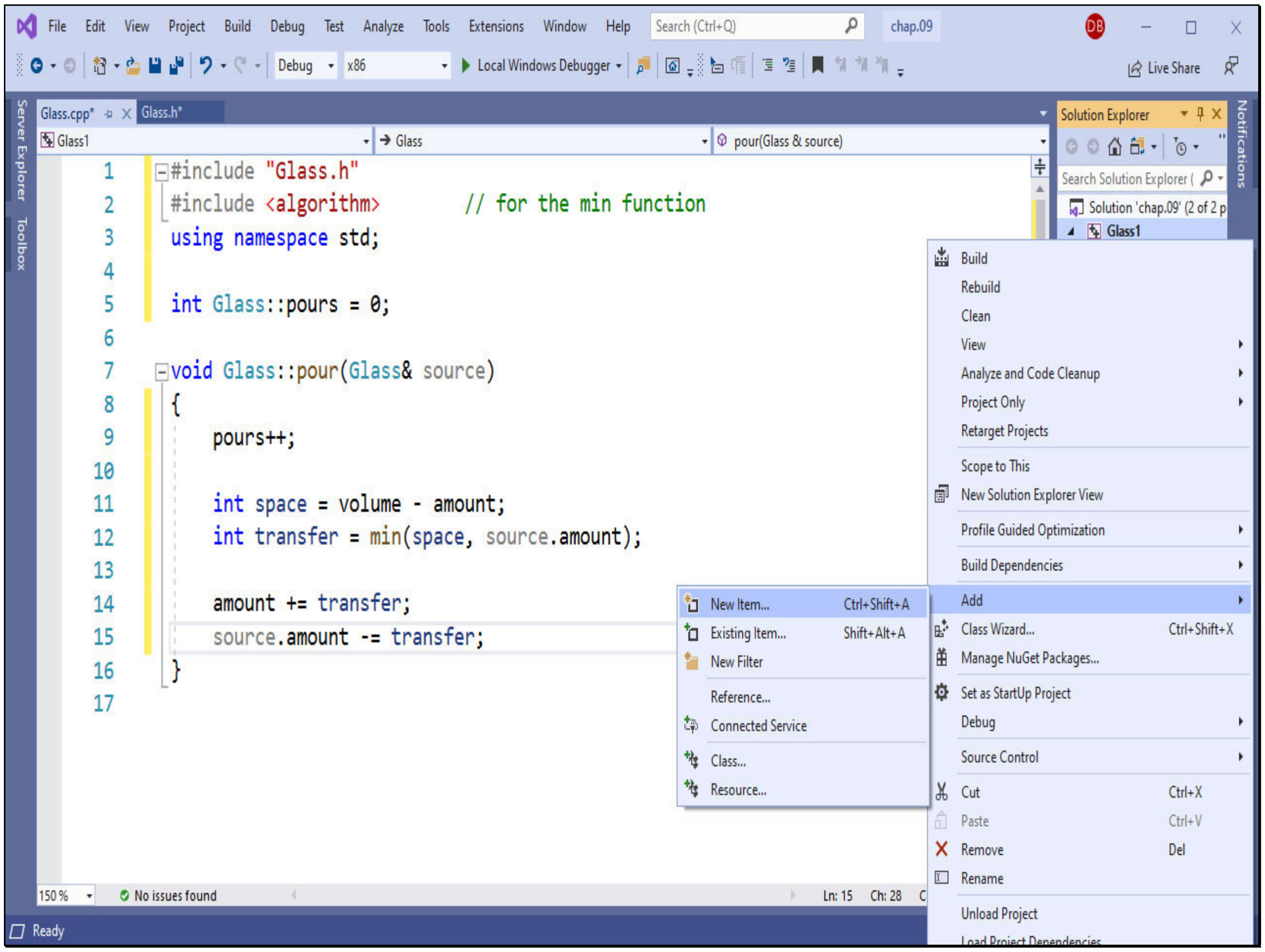
```
#include "Glass.h"
#include <algorithm>        // for the min function
using namespace std;

int Glass::pours = 0;

void Glass::pour(Glass& source)
{
    pours++;

    int space = volume - amount;
    int transfer = min(space, source.amount);

    amount += transfer;
    source.amount -= transfer;
}

```

Text Captions

The transfer amount is added to the destination glass and subtracted from the source glass.

The pour function is now complete.

Slide 33

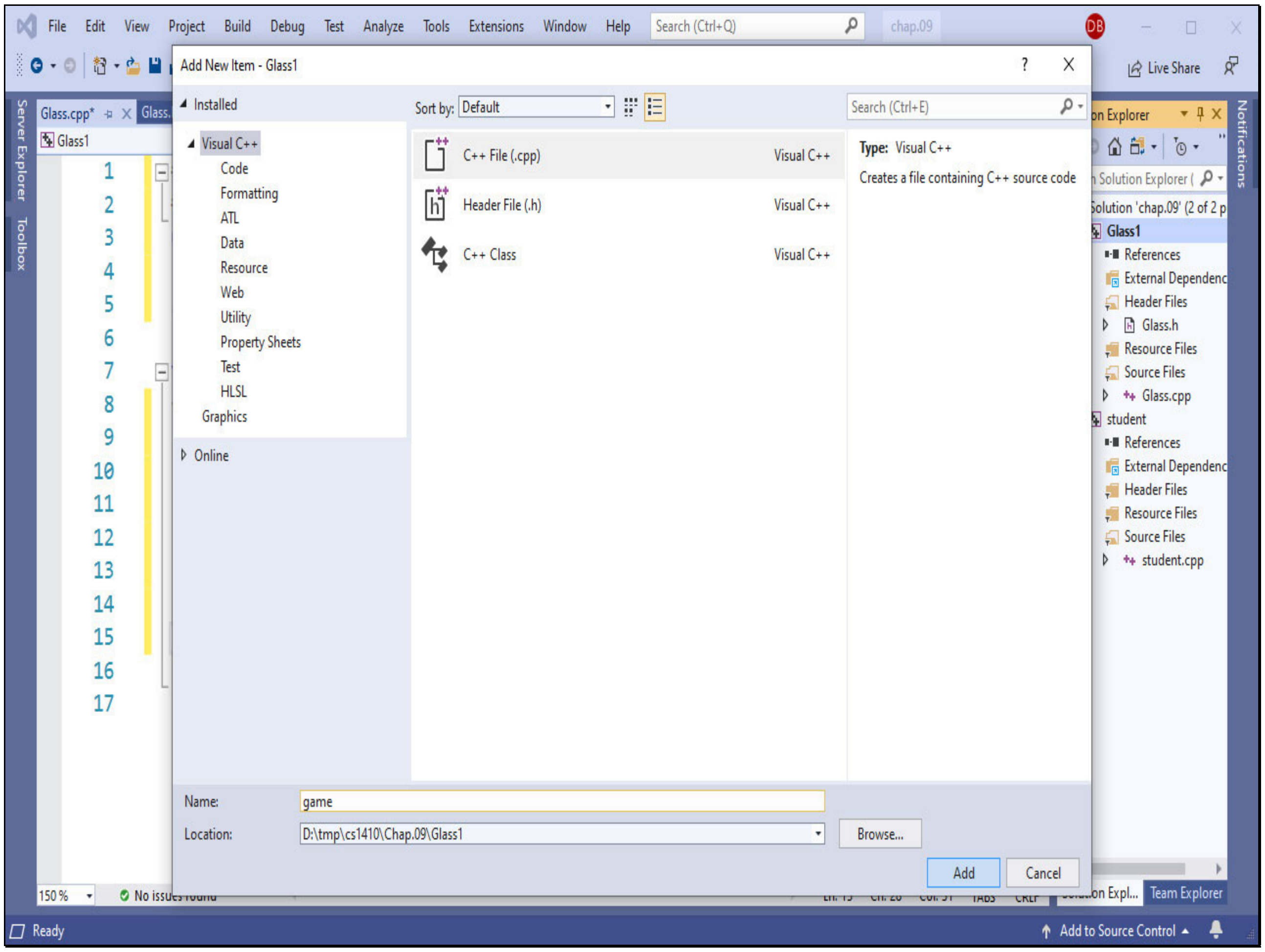


```
#include "Glass.h"
#include <algorithm>        // for the min function
using namespace std;

int Glass::pours = 0;

void Glass::pour(Glass& source)
{
    pours++;

    int space = volume - amount;
    int transfer = min(space, source.amount);

    amount += transfer;
    source.amount -= transfer;
}
```

Text Captions

The next step is adding a main function that implements the game by using the finished Glass class. Right-click the project, select “Add” and “New Item….”

Slide 34



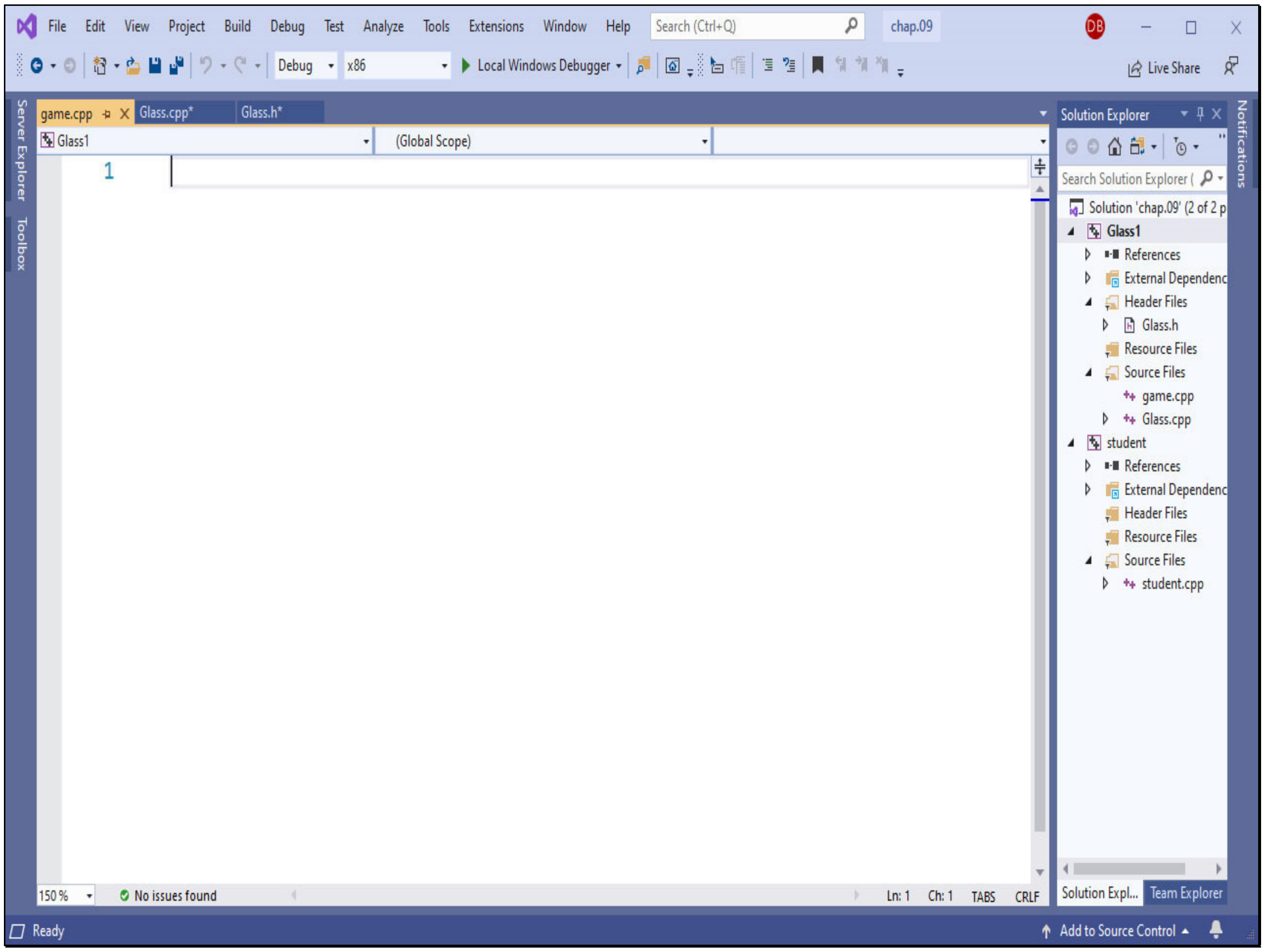
Type: Visual C++
Creates a file containing C++ source code

Name: game
Location: D:\tmp\cs1410\Chap.09\Glass1

Text Captions

Create a C++ source code file named “game” and press the “Add” button.

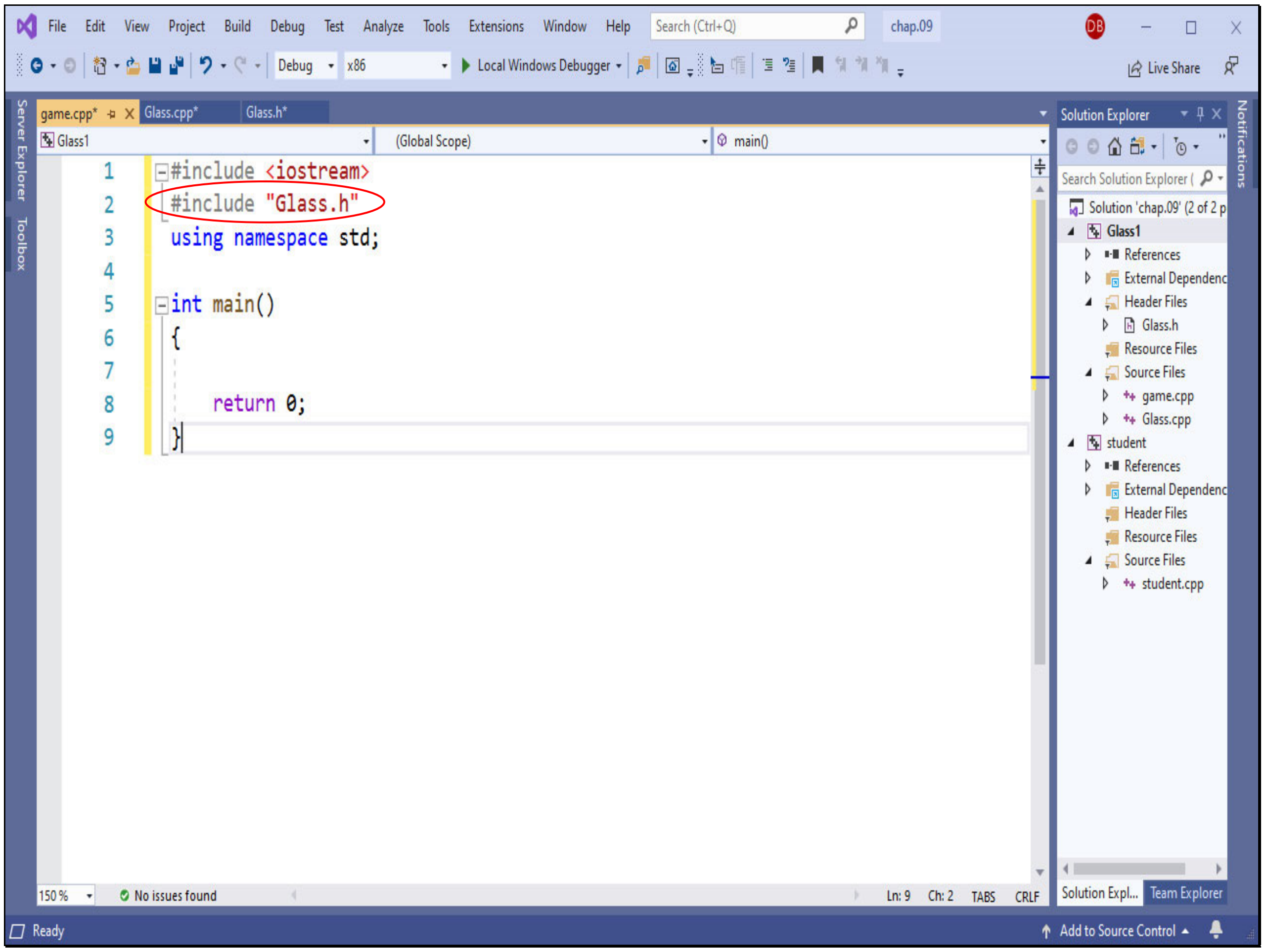
Slide 35



Text Captions

We'll place the rest of the game code in main in the game.cpp file.

Slide 36

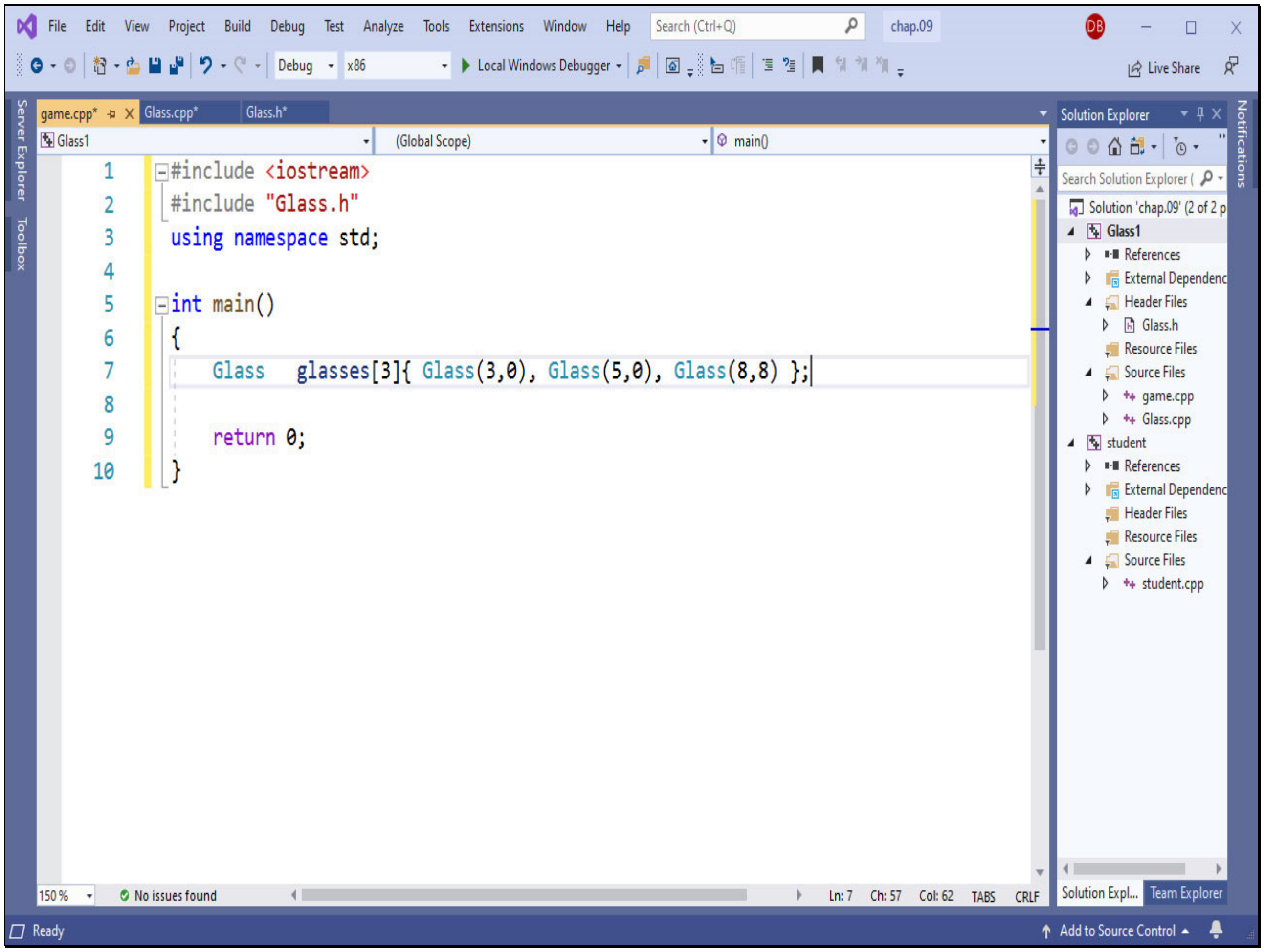


```
#include <iostream>
#include "Glass.h"
using namespace std;

int main()
{

    return 0;
}
```

Text Captions

This is basically our standard starting code with the addition of the #include “Glass.h” directive.

Slide 37

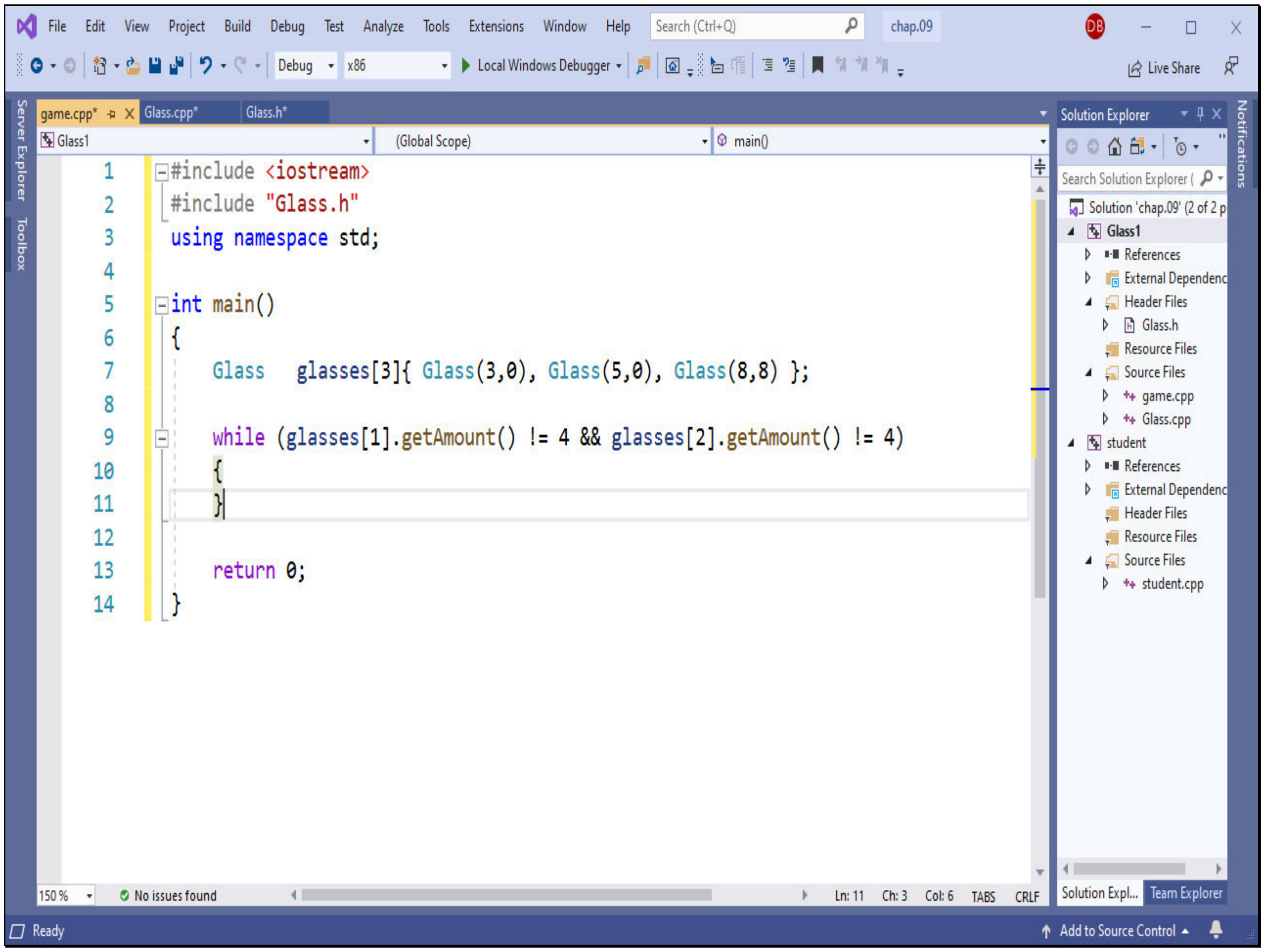


```
#include <iostream>
#include "Glass.h"
using namespace std;

int main()
{
    Glass   glasses[3]{ Glass(3,0), Glass(5,0), Glass(8,8) };

    return 0;
}
```

Text Captions

The game requires three instances of the Glass class – that is, three Glass objects. Two of the glasses, the 3- and 5-ounce, are initially empty; the 8-ounce glass is initially full. As discussed in the previous, problem solving section, we can simplify some of the operations if we use an array of Glass objects rather than three separate objects. Notice that the glasses are created on the stack and that the Glass constructor is called for each object.

Slide 38

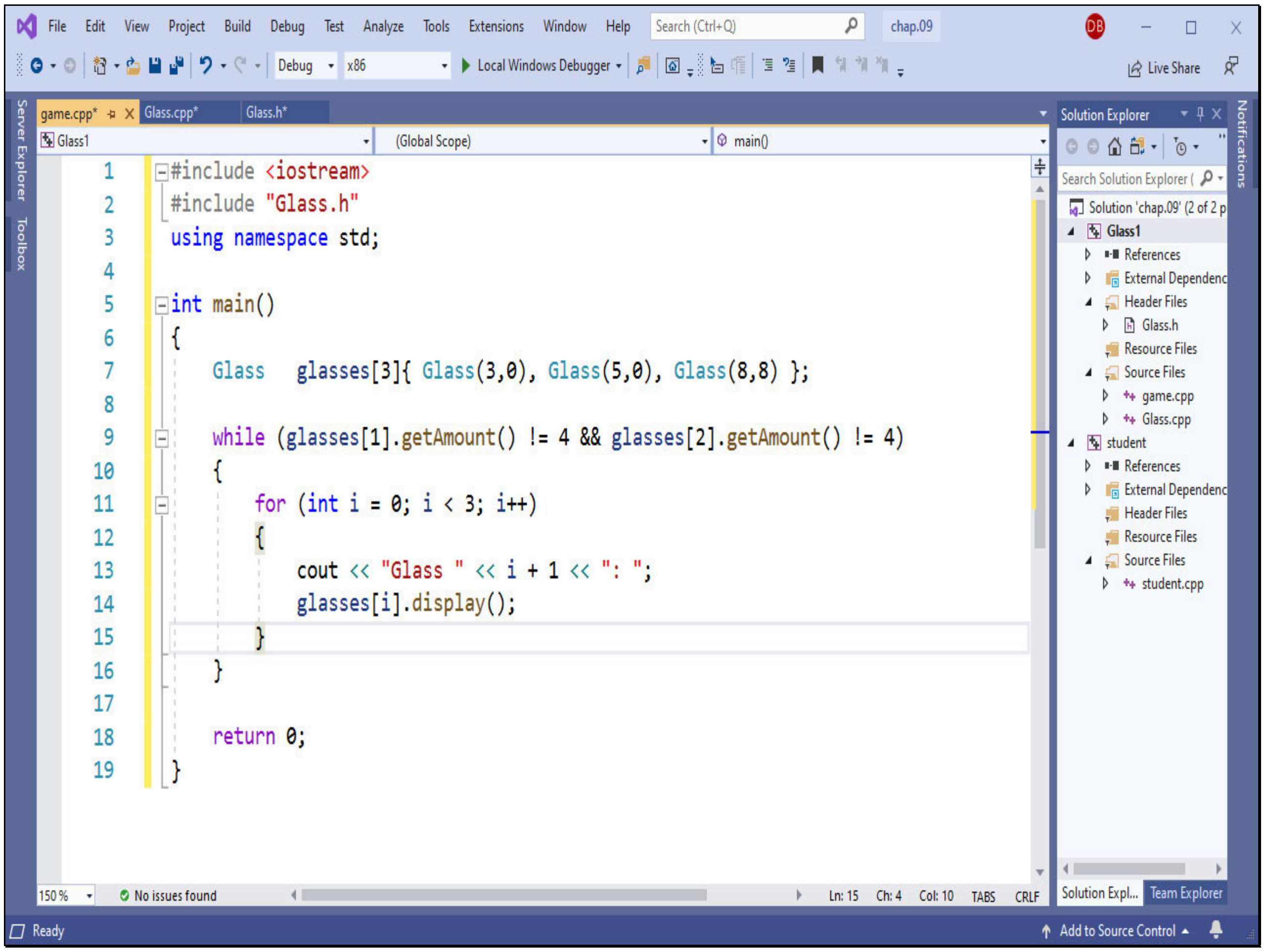


```
#include <iostream>
#include "Glass.h"
using namespace std;

int main()
{
    Glass   glasses[3]{ Glass(3,0), Glass(5,0), Glass(8,8) };

    while (glasses[1].getAmount() != 4 && glasses[2].getAmount() != 4)
    {
    }

    return 0;
}
```

Text Captions

The game continues until at least one of the three glasses contains four ounces of water. One glass has a maximum volume of three ounces, so it isn't considered in the while-loop test.

Slide 39

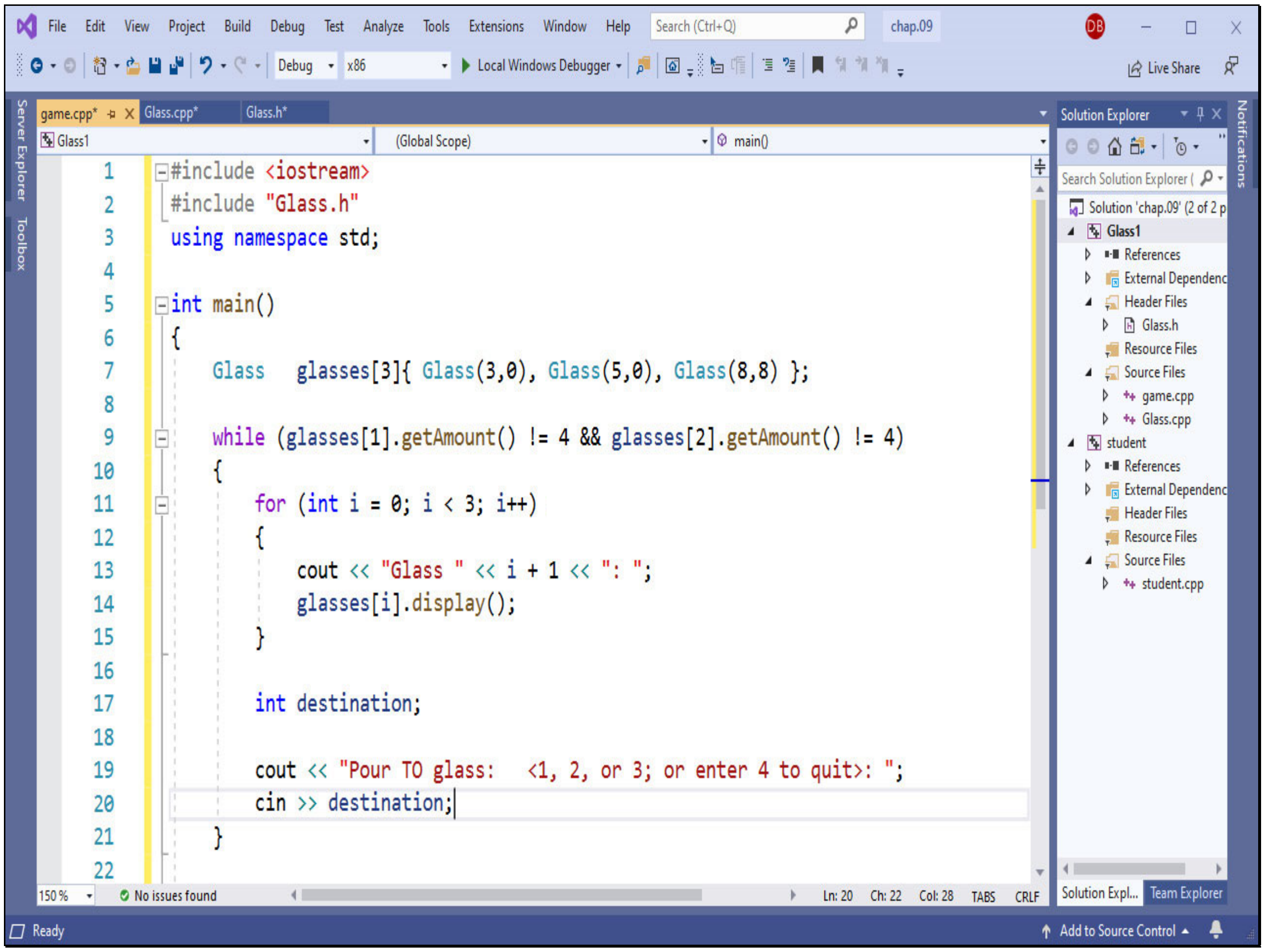


```
#include <iostream>
#include "Glass.h"
using namespace std;

int main()
{
    Glass   glasses[3]{ Glass(3,0), Glass(5,0), Glass(8,8) };

    while (glasses[1].getAmount() != 4 && glasses[2].getAmount() != 4)
    {
        for (int i = 0; i < 3; i++)
        {
            cout << "Glass " << i + 1 << ": ";
            glasses[i].display();
        }
    }

    return 0;
}
```

Text Captions

To help the player decide how to take the next step, at the beginning of each move, we display the current state of the game – that is, the amount of water in each glass.

Slide 40

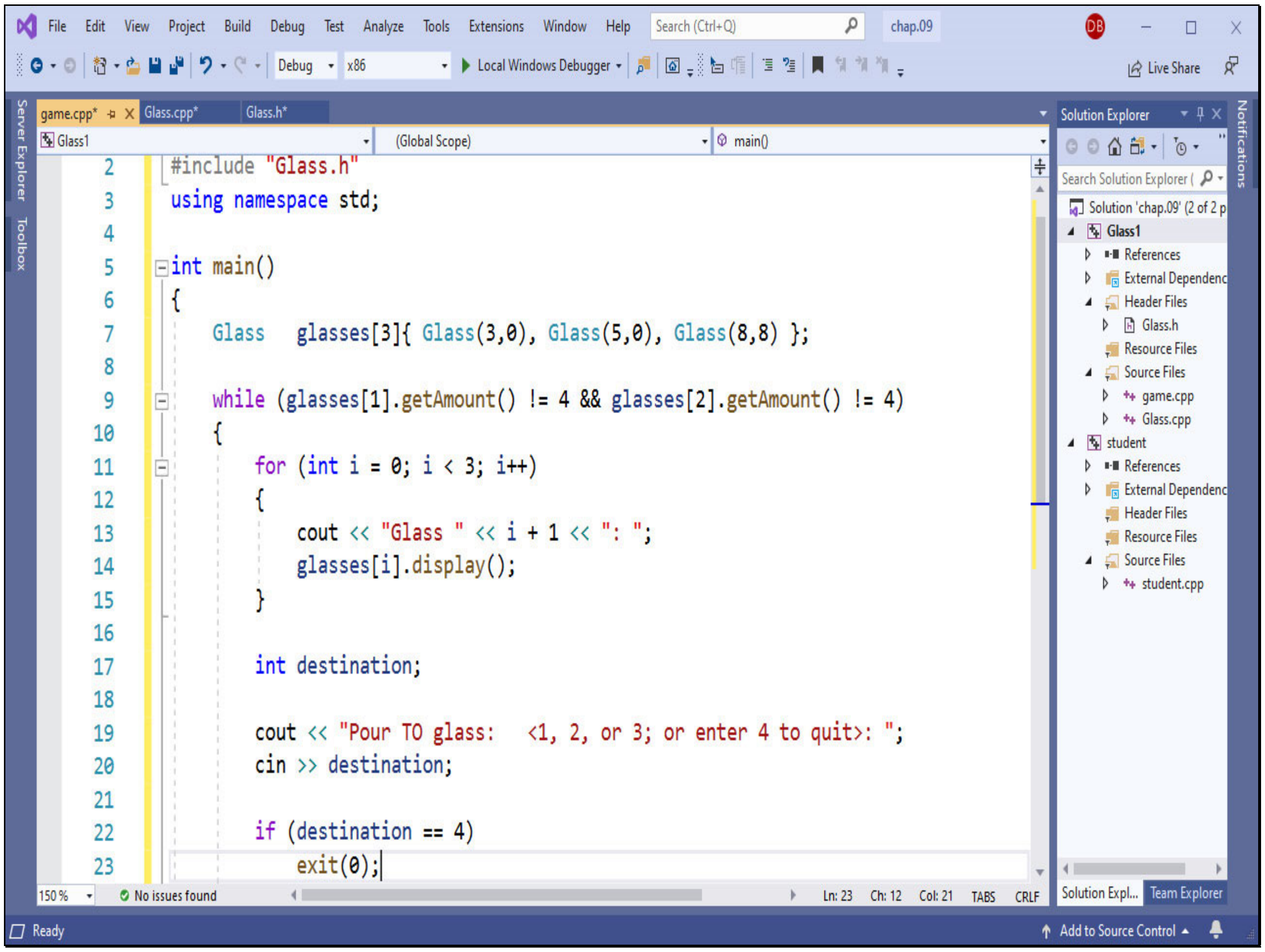


```cpp
#include <iostream>
#include "Glass.h"
using namespace std;

int main()
{
    Glass   glasses[3]{ Glass(3,0), Glass(5,0), Glass(8,8) };

    while (glasses[1].getAmount() != 4 && glasses[2].getAmount() != 4)
    {
        for (int i = 0; i < 3; i++)
        {
            cout << "Glass " << i + 1 << ": ";
            glasses[i].display();
        }

        int destination;

        cout << "Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: ";
        cin >> destination;
    }

```

Text Captions

The next step allows the player to choose to which glass the water is poured. It’s convenient to label the glasses with counting numbers – that is 1, 2, or 3 – rather than starting with 0, as that is likely more familiar to most people.

Slide 41

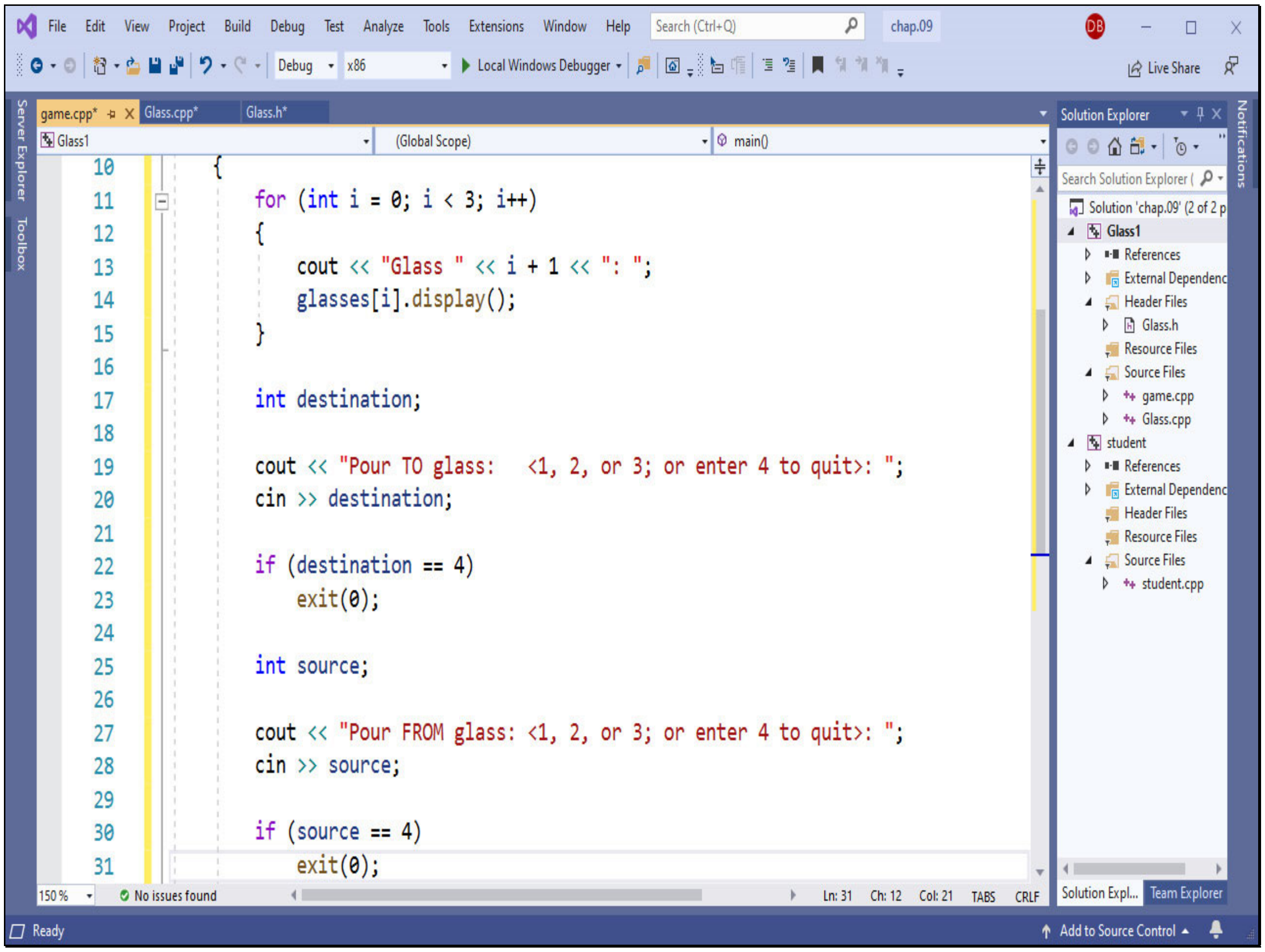


```
#include "Glass.h"
using namespace std;

int main()
{
    Glass   glasses[3]{ Glass(3,0), Glass(5,0), Glass(8,8) };

    while (glasses[1].getAmount() != 4 && glasses[2].getAmount() != 4)
    {
        for (int i = 0; i < 3; i++)
        {
            cout << "Glass " << i + 1 << ": ";
            glasses[i].display();
        }

        int destination;

        cout << "Pour TO glass:  <1, 2, or 3; or enter 4 to quit>: ";
        cin >> destination;

        if (destination == 4)
            exit(0);
```

Text Captions

As suggested by the prompt, we also allow the player to end the game by entering the value 4 and implement the early exit with an if-statement.

Slide 42

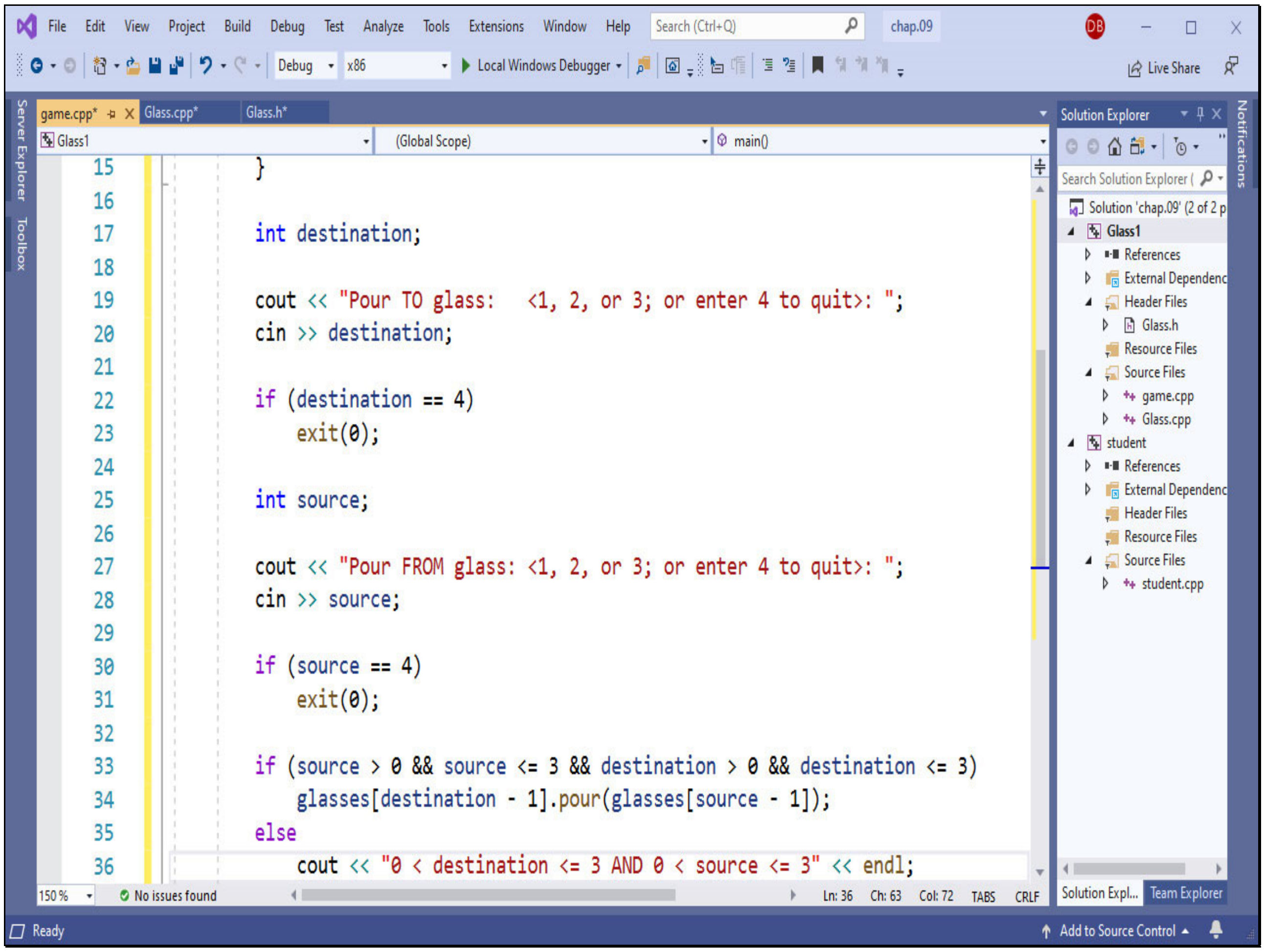


```
{
    for (int i = 0; i < 3; i++)
    {
        cout << "Glass " << i + 1 << ": ";
        glasses[i].display();
    }

    int destination;

    cout << "Pour TO glass:  <1, 2, or 3; or enter 4 to quit>: ";
    cin >> destination;

    if (destination == 4)
        exit(0);

    int source;

    cout << "Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: ";
    cin >> source;

    if (source == 4)
        exit(0);
```

Text Captions

Next, we allow the player to choose from which glass the water is poured. Again, we label the glasses as 1, 2, or 3, and allow the player to end the game early by entering 4.

Slide 43


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
chap.09
Debug x86 Local Windows Debugger
Live Share
game.cpp* Glass.cpp* Glass.h*
Glass1 (Global Scope) main()
Solution Explorer
Search Solution Explorer (
Solution 'chap.09' (2 of 2 p
Glass1
References
External Dependenc
Header Files
Glass.h
Resource Files
Source Files
game.cpp
Glass.cpp
student
References
External Dependenc
Header Files
Resource Files
Source Files
student.cpp
150 % No issues found Ln: 36 Ch: 63 Col: 72 TABS CRLF
Solution Expl... Team Explorer
Ready
Add to Source Control
Server Explorer Toolbox Notifications

```
15      }
16
17      int destination;
18
19      cout << "Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: ";
20      cin >> destination;
21
22      if (destination == 4)
23          exit(0);
24
25      int source;
26
27      cout << "Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: ";
28      cin >> source;
29
30      if (source == 4)
31          exit(0);
32
33      if (source > 0 && source <= 3 && destination > 0 && destination <= 3)
34          glasses[destination - 1].pour(glasses[source - 1]);
35      else
36          cout << "0 < destination <= 3 AND 0 < source <= 3" << endl;
```

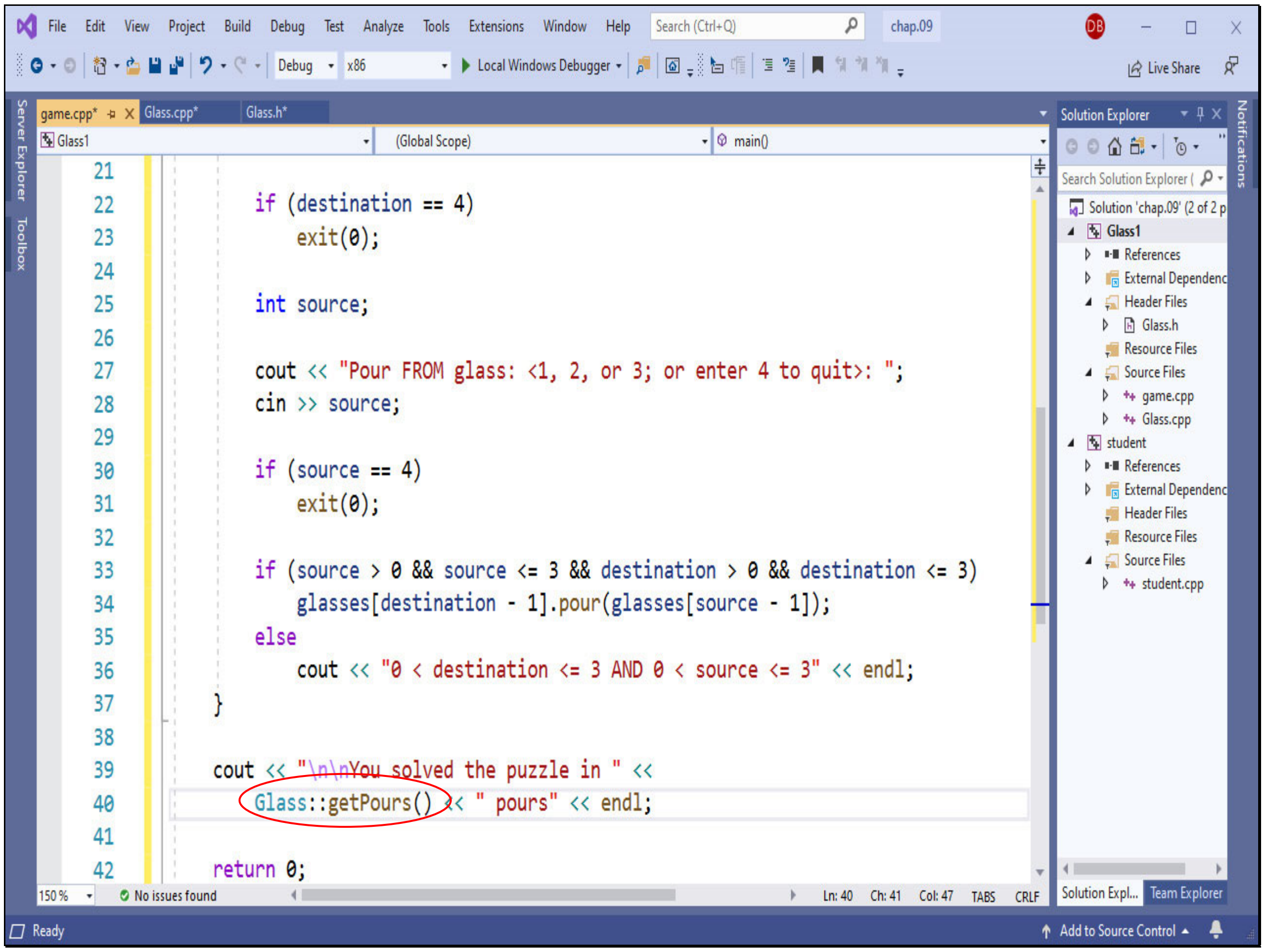
Text Captions

Anytime that user input is used as an index into an array, it must be validated – this is an important security issue: indexing an array out of bounds causes a buffer overrun or a buffer overflow. What happens depends on several factors that are completely beyond our control. In the worst-case scenario, nothing bad happens while we are testing our code, but later, the program can crash, or it can leave a vulnerability that a bad actor can exploit to coopt or infect the system.

A simple if-statement can check the user input and ensure that it represents a valid array index. Assume that the index is valid and focus your attention on line 34. To correctly index both arrays, the program subtracts 1 from each of the user input values. This step is done because we labeled the glasses with counting numbers: 1, 2, and 3. But C++ arrays are zero-indexed, so, for an array of three objects, valid index values are 0, 1, and 2.

Still looking at line 34, once two of the three objects are identified in the array, the pour function is called, which pours or transfers water from the glass object inside the parentheses to the glass object appearing to the left of the dot operator. This completes the code inside the while-loop.

Slide 44

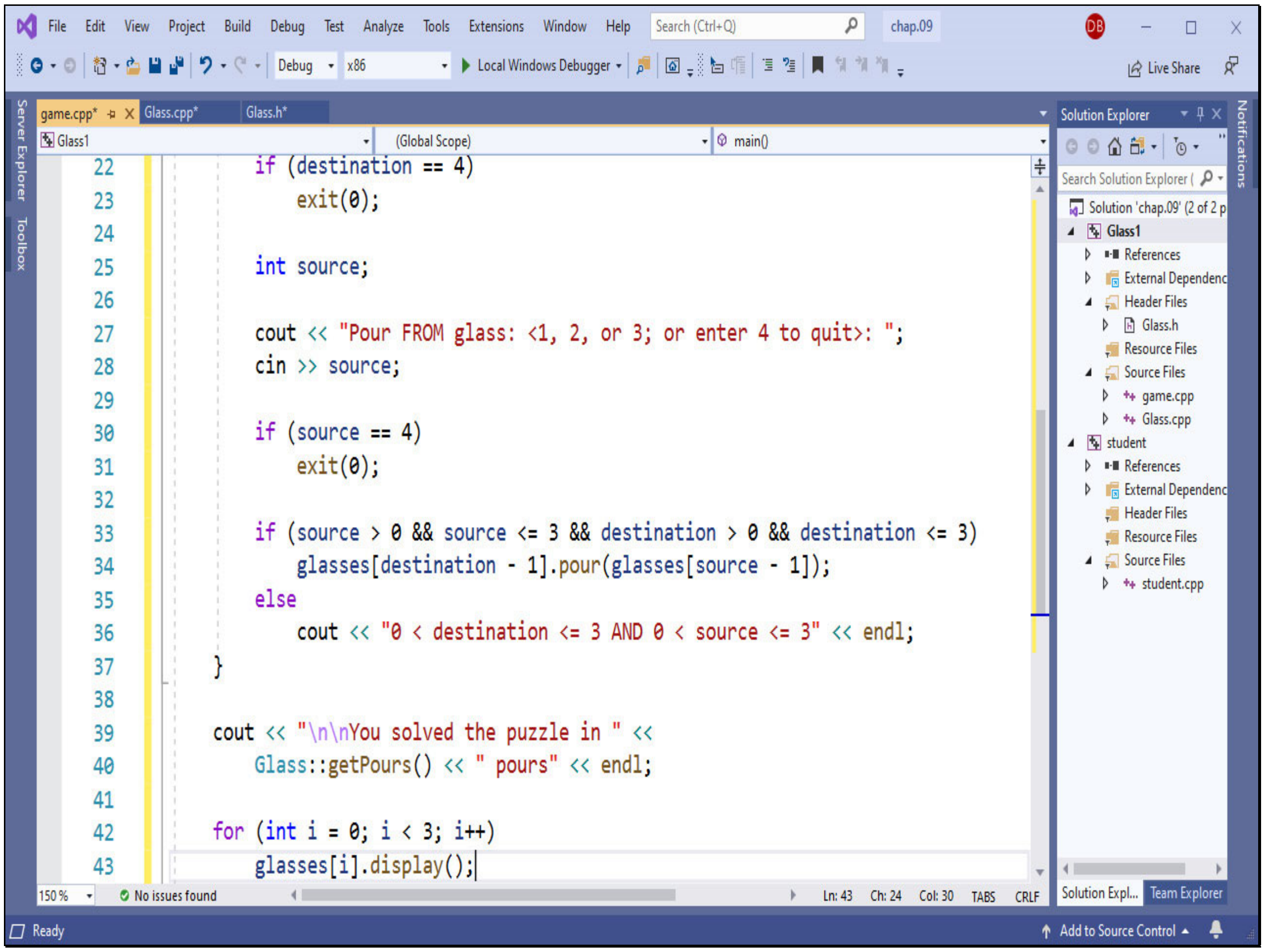


```
21
22          if (destination == 4)
23              exit(0);
24
25          int source;
26
27          cout << "Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: ";
28          cin >> source;
29
30          if (source == 4)
31              exit(0);
32
33          if (source > 0 && source <= 3 && destination > 0 && destination <= 3)
34              glasses[destination - 1].pour(glasses[source - 1]);
35          else
36              cout << "0 < destination <= 3 AND 0 < source <= 3" << endl;
37      }
38
39      cout << "\n\nYou solved the puzzle in " <<
40          Glass::getPours() << " pours" << endl;
41
42      return 0;
```

Text Captions

Solving the puzzle ends the while-loop and the program prints the total number of pour operations taken to solve the puzzle. Recall that the getPours function is static – line 40 illustrates the preferred way to call a static function – the class name, the scope resolution operator, and then the function name.

Slide 45

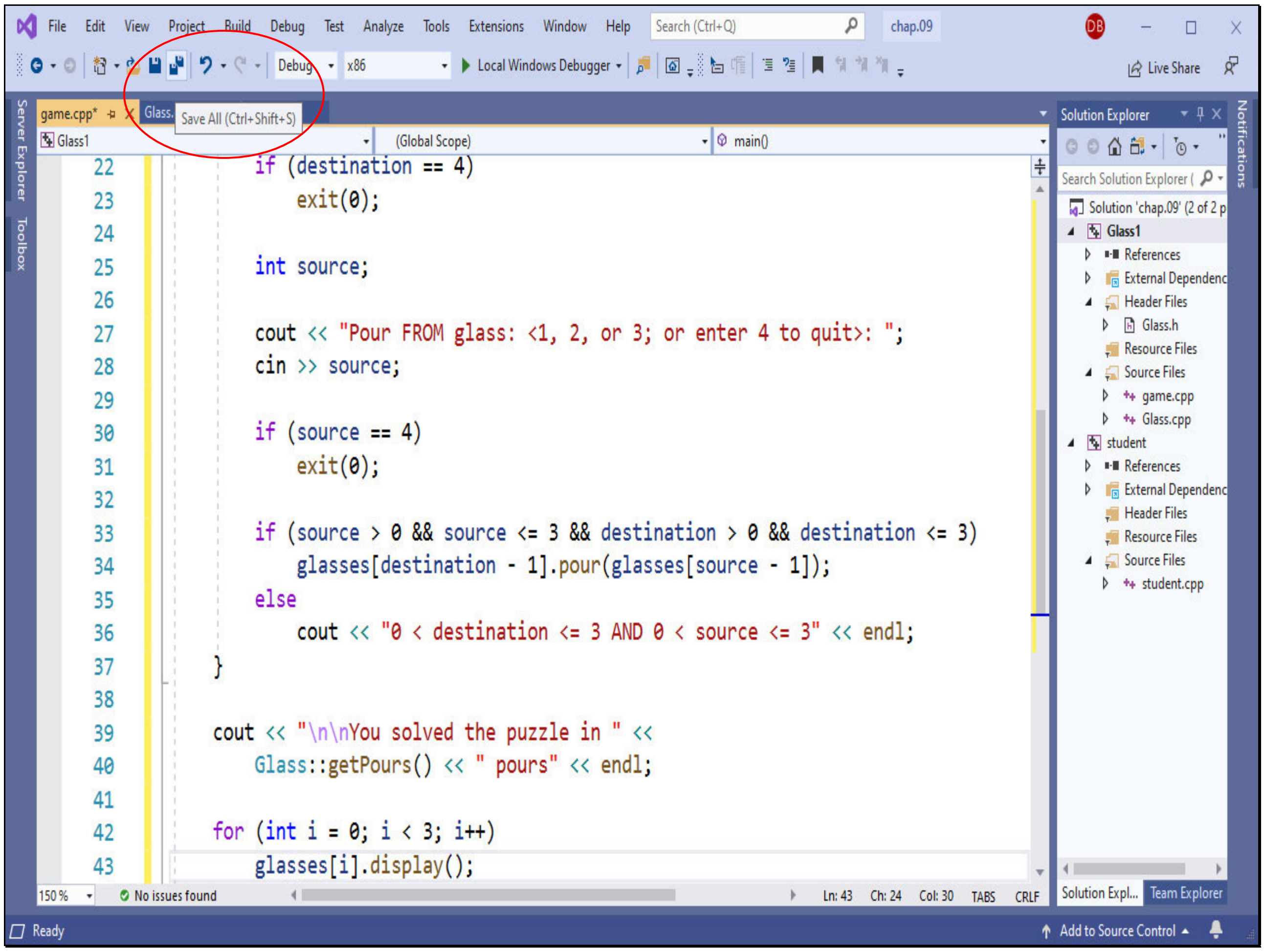


```
        if (destination == 4)
            exit(0);

        int source;

        cout << "Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: ";
        cin >> source;

        if (source == 4)
            exit(0);

        if (source > 0 && source <= 3 && destination > 0 && destination <= 3)
            glasses[destination - 1].pour(glasses[source - 1]);
        else
            cout << "0 < destination <= 3 AND 0 < source <= 3" << endl;
    }

    cout << "\n\nYou solved the puzzle in " <<
        Glass::getPours() << " pours" << endl;

    for (int i = 0; i < 3; i++)
        glasses[i].display();
```

Text Captions

Finally, the program prints the final state of the game – that is, the amount of water in all three glasses.

Slide 46

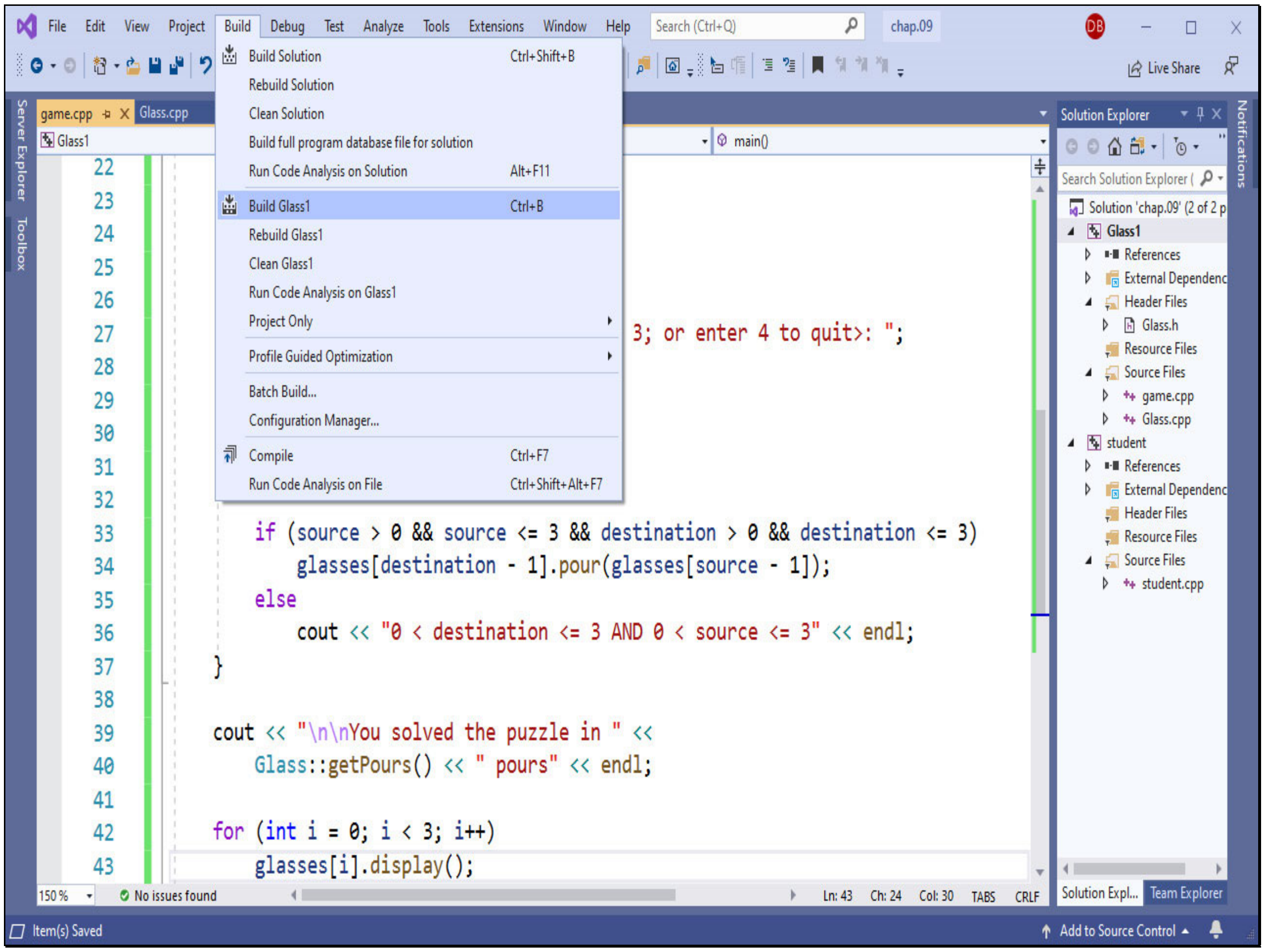


```
            if (destination == 4)
                exit(0);

            int source;

            cout << "Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: ";
            cin >> source;

            if (source == 4)
                exit(0);

            if (source > 0 && source <= 3 && destination > 0 && destination <= 3)
                glasses[destination - 1].pour(glasses[source - 1]);
            else
                cout << "0 < destination <= 3 AND 0 < source <= 3" << endl;
        }

        cout << "\n\nYou solved the puzzle in " <<
            Glass::getPours() << " pours" << endl;

        for (int i = 0; i < 3; i++)
            glasses[i].display();
```

Text Captions

Save all the files by clicking the button with the picture of two floppy disks.

Slide 47



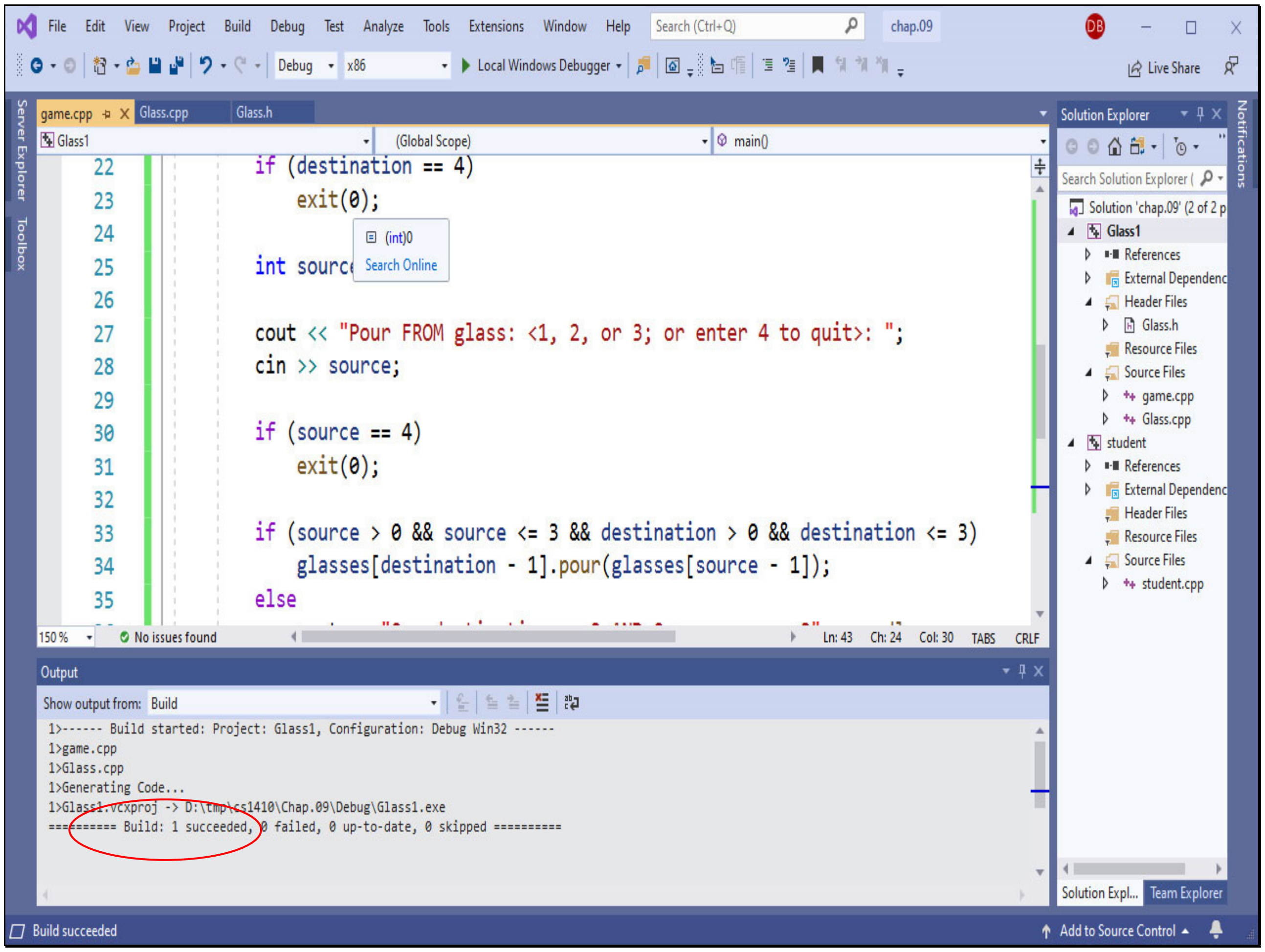
```
3; or enter 4 to quit>: ";
```

```
        if (source > 0 && source <= 3 && destination > 0 && destination <= 3)
            glasses[destination - 1].pour(glasses[source - 1]);
        else
            cout << "0 < destination <= 3 AND 0 < source <= 3" << endl;
    }

    cout << "\n\nYou solved the puzzle in " <<
        Glass::getPours() << " pours" << endl;

    for (int i = 0; i < 3; i++)
        glasses[i].display();
```

Text Captions

Build the project.

Slide 48



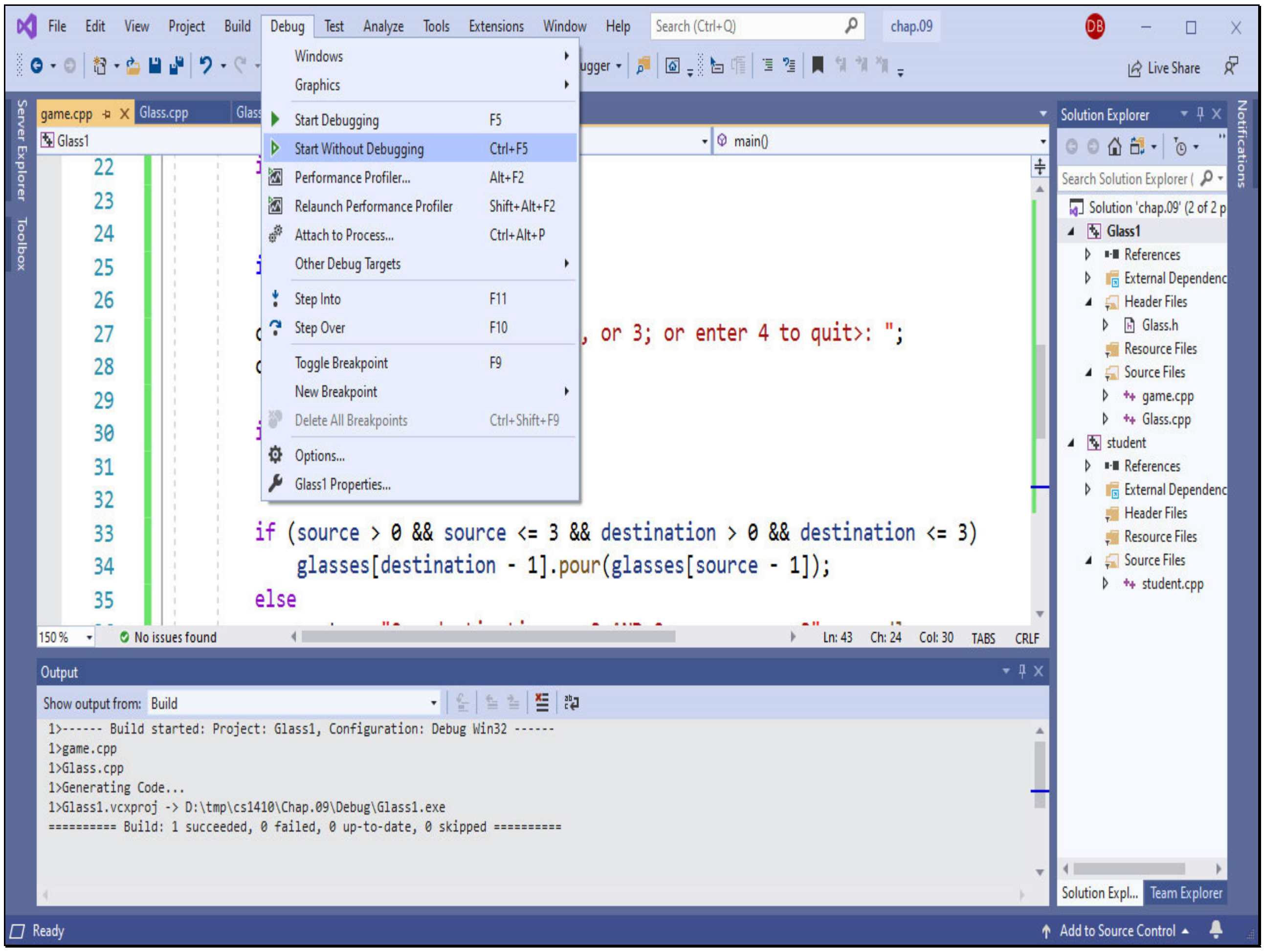
```
        if (destination == 4)
            exit(0);

        int sourc

        cout << "Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: ";
        cin >> source;

        if (source == 4)
            exit(0);

        if (source > 0 && source <= 3 && destination > 0 && destination <= 3)
            glasses[destination - 1].pour(glasses[source - 1]);
        else
```

```
1>------ Build started: Project: Glass1, Configuration: Debug Win32 ------
1>game.cpp
1>Glass.cpp
1>Generating Code...
1>Glass1.vcxproj -> D:\tmp\cs1410\Chap.09\Debug\Glass1.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Text Captions

The program builds without any errors.

Slide 49



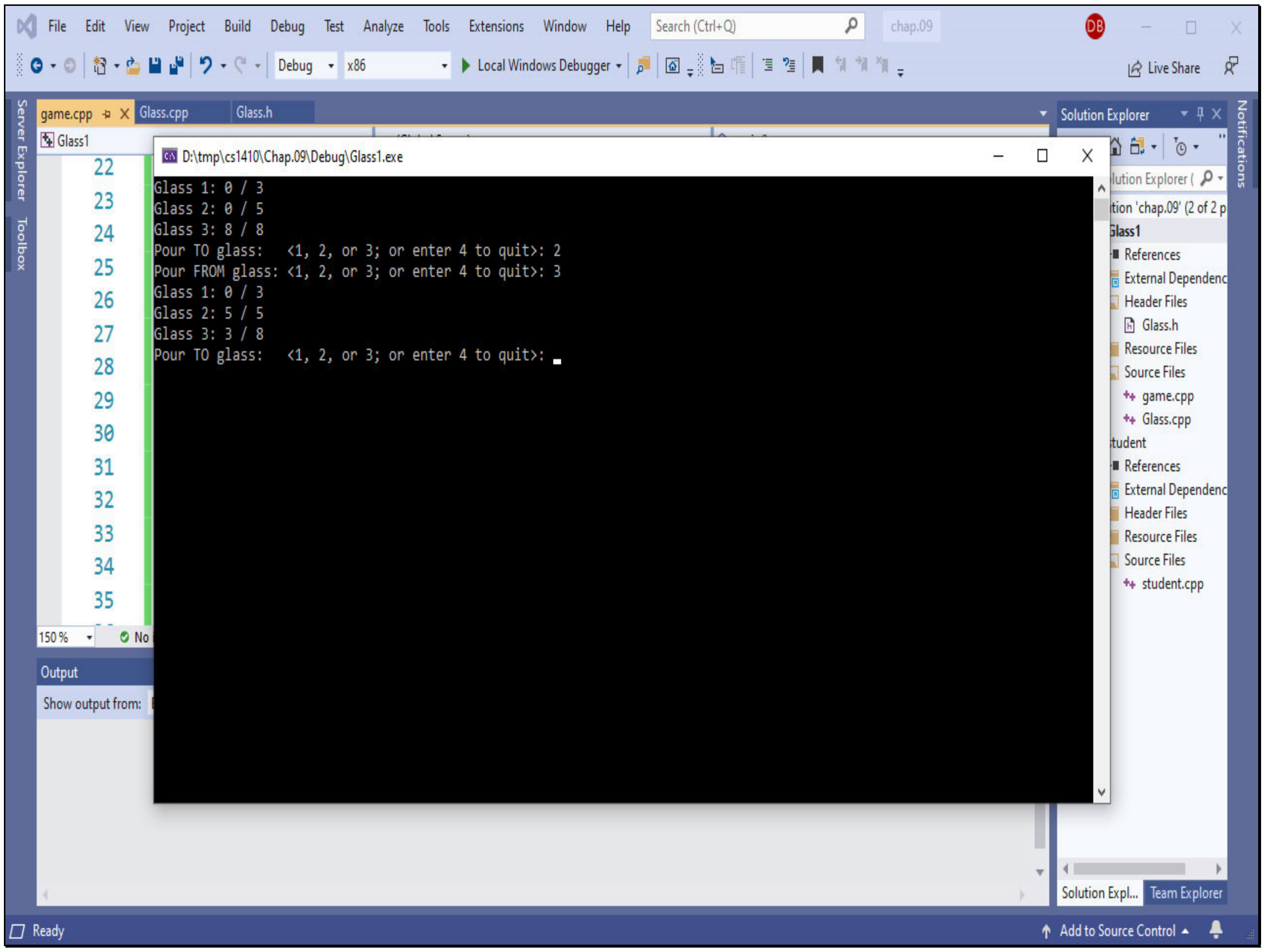
```
, or 3; or enter 4 to quit>: ";

if (source > 0 && source <= 3 && destination > 0 && destination <= 3)
    glasses[destination - 1].pour(glasses[source - 1]);
else
```

```
1>------ Build started: Project: Glass1, Configuration: Debug Win32 ------
1>game.cpp
1>Glass.cpp
1>Generating Code...
1>Glass1.vcxproj -> D:\tmp\cs1410\Chap.09\Debug\Glass1.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Text Captions

Start the program running.

Slide 50



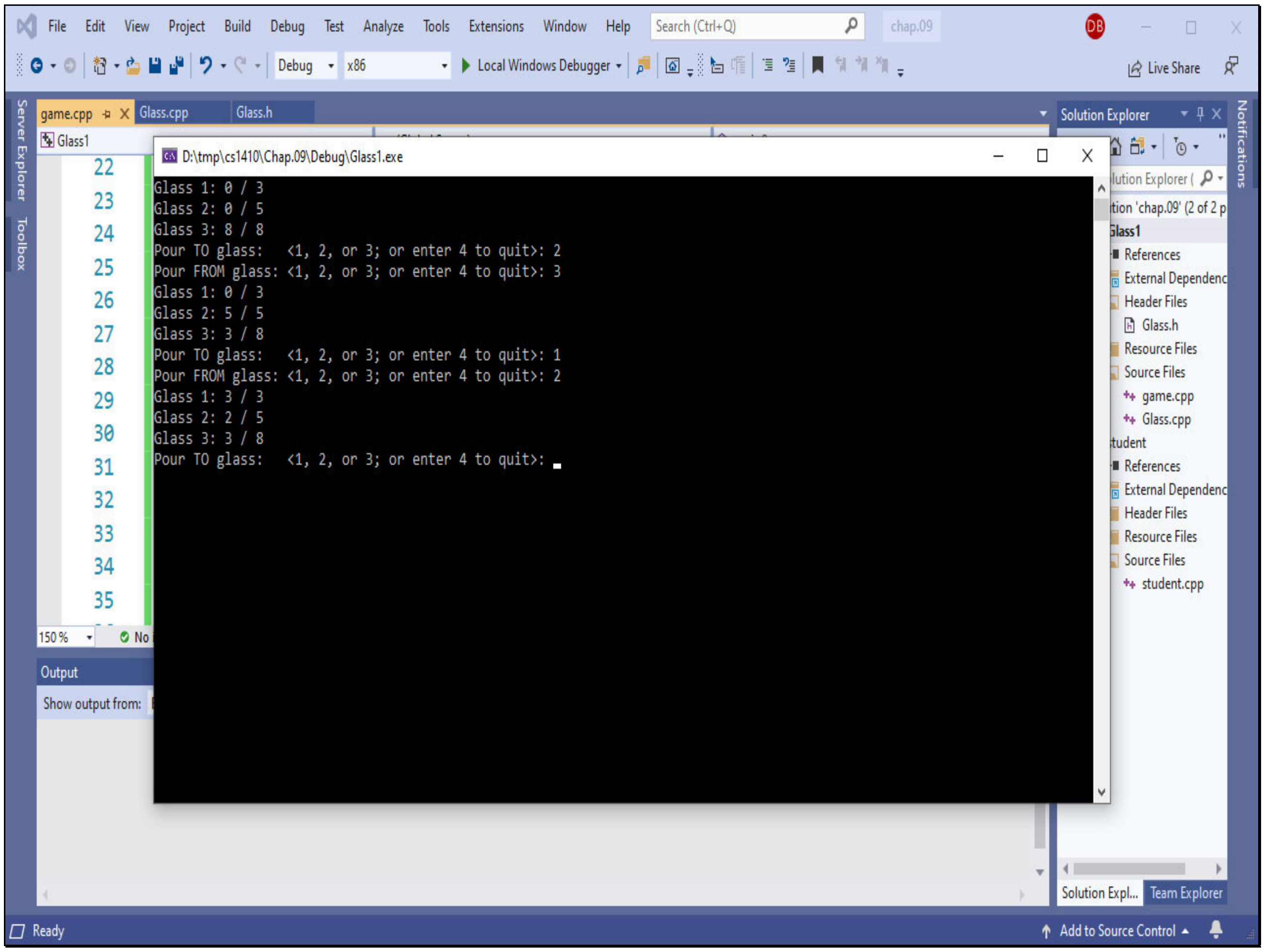
```
Glass 1: 0 / 3
Glass 2: 0 / 5
Glass 3: 8 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 2
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 3
Glass 1: 0 / 3
Glass 2: 5 / 5
Glass 3: 3 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: _
```

Text Captions

The program prints the game's initial state: the two smaller glasses are empty and the largest, 8-ounc glass, is full.

I choose glass 2 to receive water from glass 3. Glass 2, which has a volume of 5 ounces, is filled with water, leaving 3 ounces of water in glass 3. Glass 1 wasn't used and remains empty.

Slide 51

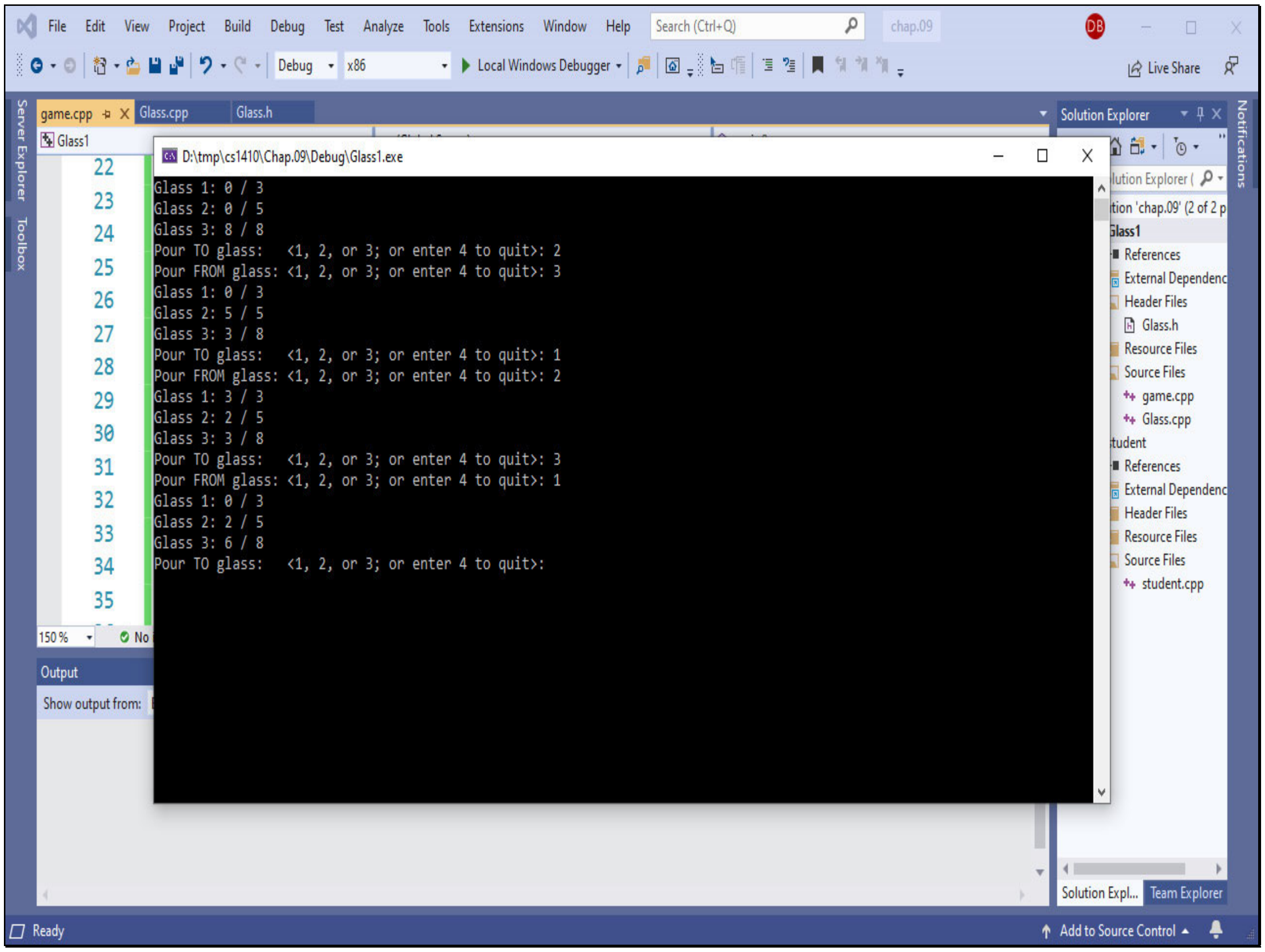


```
Glass 1: 0 / 3
Glass 2: 0 / 5
Glass 3: 8 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 2
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 3
Glass 1: 0 / 3
Glass 2: 5 / 5
Glass 3: 3 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 1
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 2
Glass 1: 3 / 3
Glass 2: 2 / 5
Glass 3: 3 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>:
```

Text Captions

My next move pours water from glass 2 to glass 1, which fills glass 1 with 3 ounces and leaves 2 ounces in glass 2. Glass 3 wasn't used this time, so it still holds 3 ounces.

Slide 52

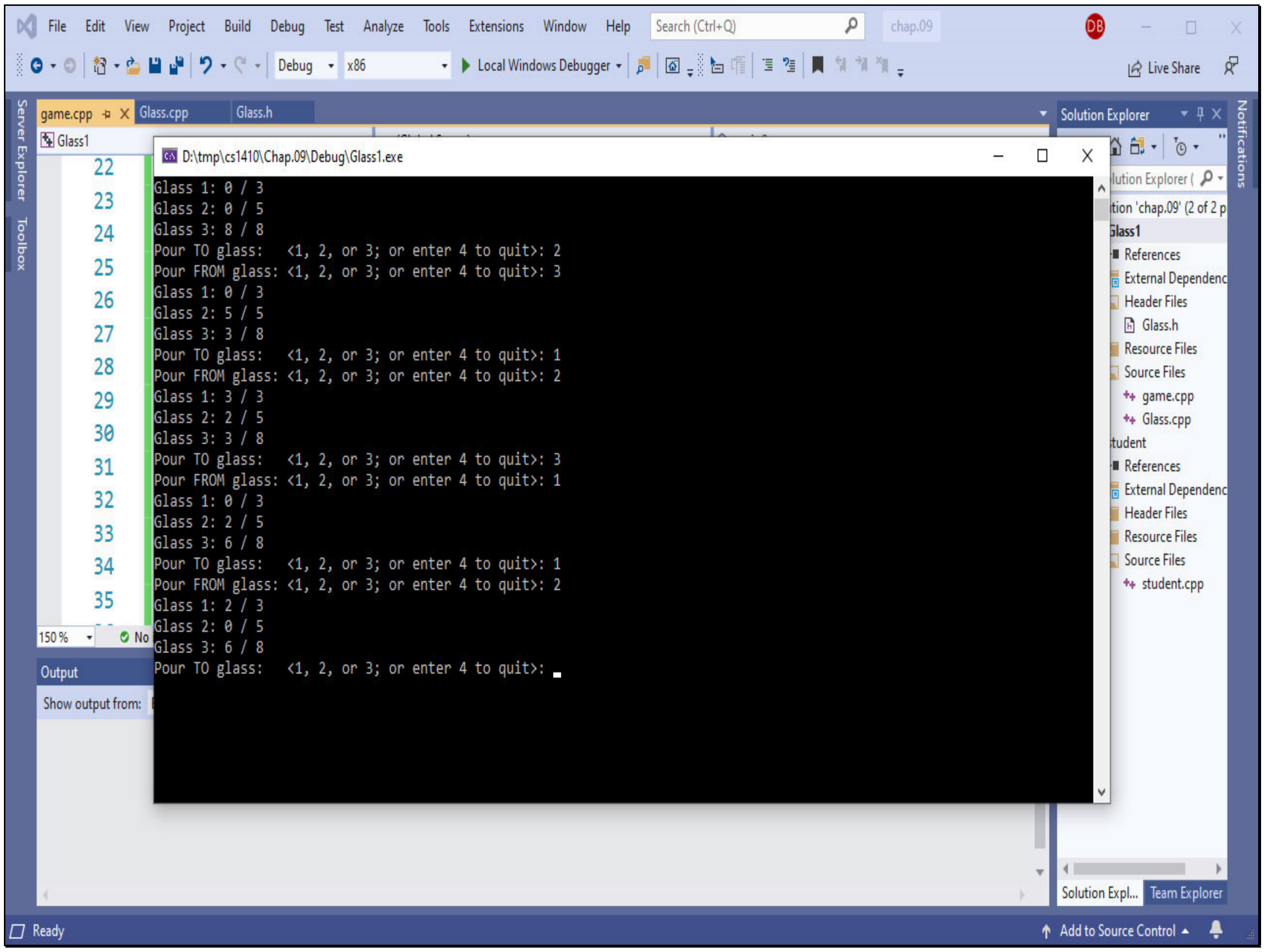


```
Glass 1: 0 / 3
Glass 2: 0 / 5
Glass 3: 8 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 2
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 3
Glass 1: 0 / 3
Glass 2: 5 / 5
Glass 3: 3 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 1
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 2
Glass 1: 3 / 3
Glass 2: 2 / 5
Glass 3: 3 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 3
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 1
Glass 1: 0 / 3
Glass 2: 2 / 5
Glass 3: 6 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>:
```

Text Captions

Next, pour water from glass 1 to glass 3.

Slide 53

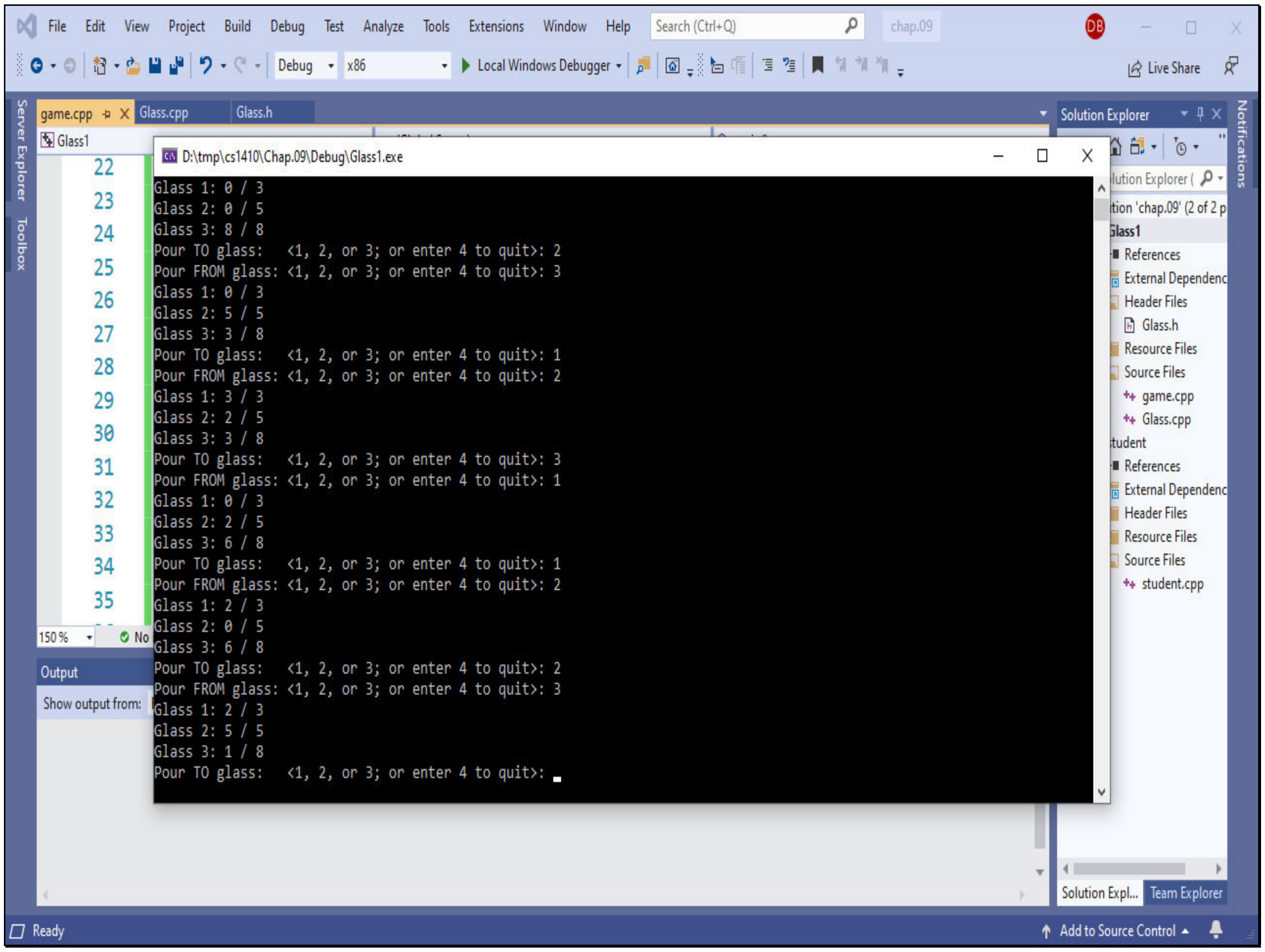


```
Glass 1: 0 / 3
Glass 2: 0 / 5
Glass 3: 8 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 2
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 3
Glass 1: 0 / 3
Glass 2: 5 / 5
Glass 3: 3 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 1
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 2
Glass 1: 3 / 3
Glass 2: 2 / 5
Glass 3: 3 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 3
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 1
Glass 1: 0 / 3
Glass 2: 2 / 5
Glass 3: 6 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 1
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 2
Glass 1: 2 / 3
Glass 2: 0 / 5
Glass 3: 6 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>:
```

Text Captions

The next move repeats the second move: pour water from glass 2 to glass 1.

Slide 54

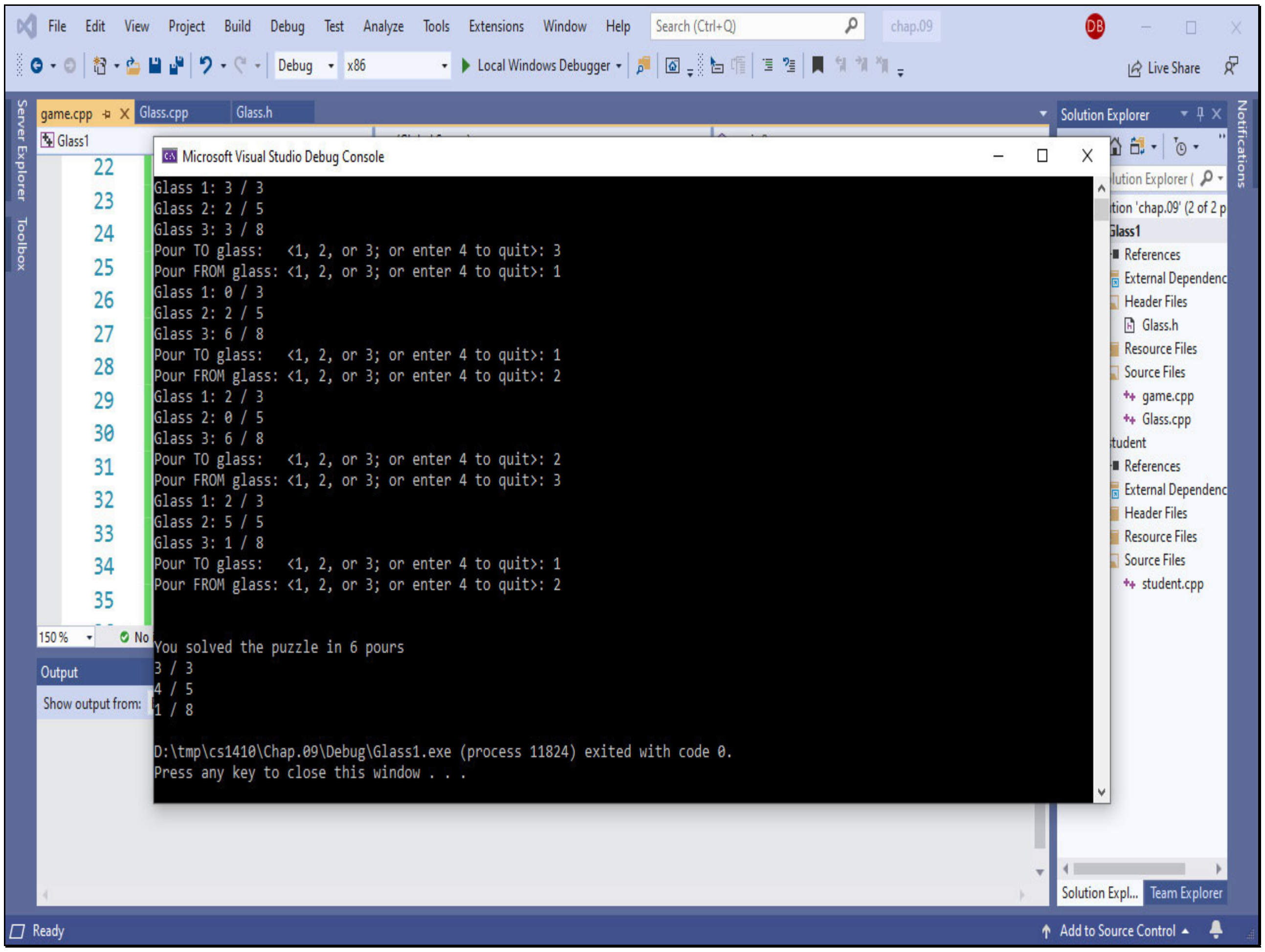


```
Glass 1: 0 / 3
Glass 2: 0 / 5
Glass 3: 8 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 2
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 3
Glass 1: 0 / 3
Glass 2: 5 / 5
Glass 3: 3 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 1
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 2
Glass 1: 3 / 3
Glass 2: 2 / 5
Glass 3: 3 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 3
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 1
Glass 1: 0 / 3
Glass 2: 2 / 5
Glass 3: 6 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 1
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 2
Glass 1: 2 / 3
Glass 2: 0 / 5
Glass 3: 6 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 2
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 3
Glass 1: 2 / 3
Glass 2: 5 / 5
Glass 3: 1 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>:
```

Text Captions

Pouring water from glass 3 to glass 2 also repeats an earlier move – the first one – but this time it sets the game up for the final, winning move.

Slide 55

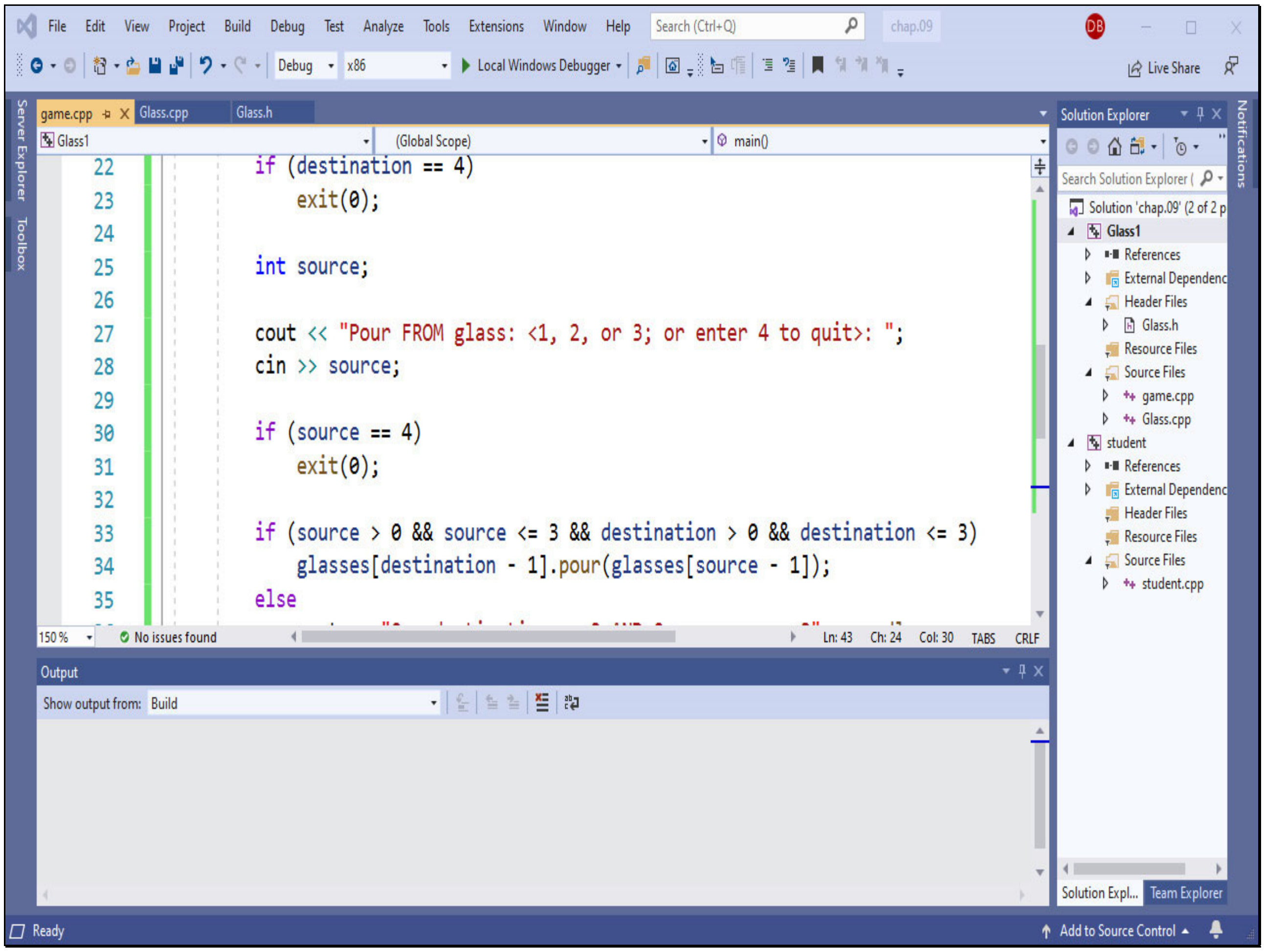


```
Glass 1: 3 / 3
Glass 2: 2 / 5
Glass 3: 3 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 3
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 1
Glass 1: 0 / 3
Glass 2: 2 / 5
Glass 3: 6 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 1
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 2
Glass 1: 2 / 3
Glass 2: 0 / 5
Glass 3: 6 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 2
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 3
Glass 1: 2 / 3
Glass 2: 5 / 5
Glass 3: 1 / 8
Pour TO glass:   <1, 2, or 3; or enter 4 to quit>: 1
Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: 2


You solved the puzzle in 6 pours
3 / 3
4 / 5
1 / 8

D:\tmp\cs1410\Chap.09\Debug\Glass1.exe (process 11824) exited with code 0.
Press any key to close this window . . .
```

Text Captions

Pouring from glass 2 to glass 1 leaves 4 ounces of water in glass 2, which solves the problem and ends the while loop. Below and outside the loop, the program prints the total number of pour operations taken to solve the puzzle and the final state of the game. I've not been able to solve the puzzle with fewer than 6 pouring operations.

Slide 56



```
        if (destination == 4)
            exit(0);

        int source;

        cout << "Pour FROM glass: <1, 2, or 3; or enter 4 to quit>: ";
        cin >> source;

        if (source == 4)
            exit(0);

        if (source > 0 && source <= 3 && destination > 0 && destination <= 3)
            glasses[destination - 1].pour(glasses[source - 1]);
        else
```

Text Captions