



Introduction to Object Oriented Programming

Chapter 10



Objectives

You will be able to:

1. Write a simple class definition.
2. Control access to the methods and data in a class.
3. Create instances of a class.
4. Write and use class constructors.



What is a class?

Essentially a struct with built-in functions

```
class Circle
{
    double radius;

    // Constructor
    Circle(double r);

    // Return the area of this circle.
    double Area();
};
```

A class declaration



Encapsulation

- By default the class definition *encapsulates*, or hides, the data inside it.
- This is a key concept of object oriented programming.
- The outside world can see and use the data only by calling the class functions.
 - Called *methods*.



Class Members

Methods and variables declared inside a class are called *members* of that class.

In order to be visible outside the class definition, a member must be declared *public*.

As written, the previous class would be unusable, because all members are private by default.



Making Methods Visible

To make the methods visible outside we would write the class declaration as:

```
class Circle
{
private:
    double radius;

public:
    // Constructor
    Circle(double r);

    // Return the area of this circle.
    double Area();
};
```

File Circle.h



Constructor

The function

`Circle(double r)`

is called a *constructor*.

Invoked to initialize object when an object of this class is created.

Note: Name same as class name.

No return type



Implementing Member Functions

```
#include "Circle.h"
```

```
Circle::Circle(double r)
{
    radius = r;
}
```

```
double Circle::Area()
{
    return 3.141592 * radius * radius;
}
```

File Circle.cpp



Creating Objects

- The class definition does not *do anything!*
 - Doesn't allocate memory.
 - Doesn't store data.
 - Doesn't perform actions.
- It just defines a type.
- To *do* anything, we have to create an *instance* of the class.
 - Call its methods to perform actions.



Objects

An instance of a class is called an *object*.

You can create any number of instances of a given class.

Each object has its own identity and its own lifetime.

Each object has its own copy of the data associated with the class.

When you call a class method, you call it through a particular object.

The method sees the data in *that object*.



Using Classes and Objects

- Classes and objects are used much like traditional types and variables:
 - Local variables
 - `Circle c1(5);`
 - Can be member variables in other classes
 - Assignment
 - `c2 = c1;`
 - Function arguments
 - `picture1.crop (c1);`



Using Classes and Objects

- In C++ we can create objects by simply *declaring* them.
- Same as ints, doubles, etc.
 - Except that we may need to provide parameter values for the constructor.
- Lifetime of an object declared as a local variable is the same as the lifetime of the function invocation.



Example: Using Class Circle

```
#include <iostream>
#include "Circle.h"

using namespace std;

int main()
{
    Circle c1(5);

    double c1_area = c1.Area();

    cout << "Area of circle c1 is " << c1_area << endl;

    cin.get();
    return 0;
}
```

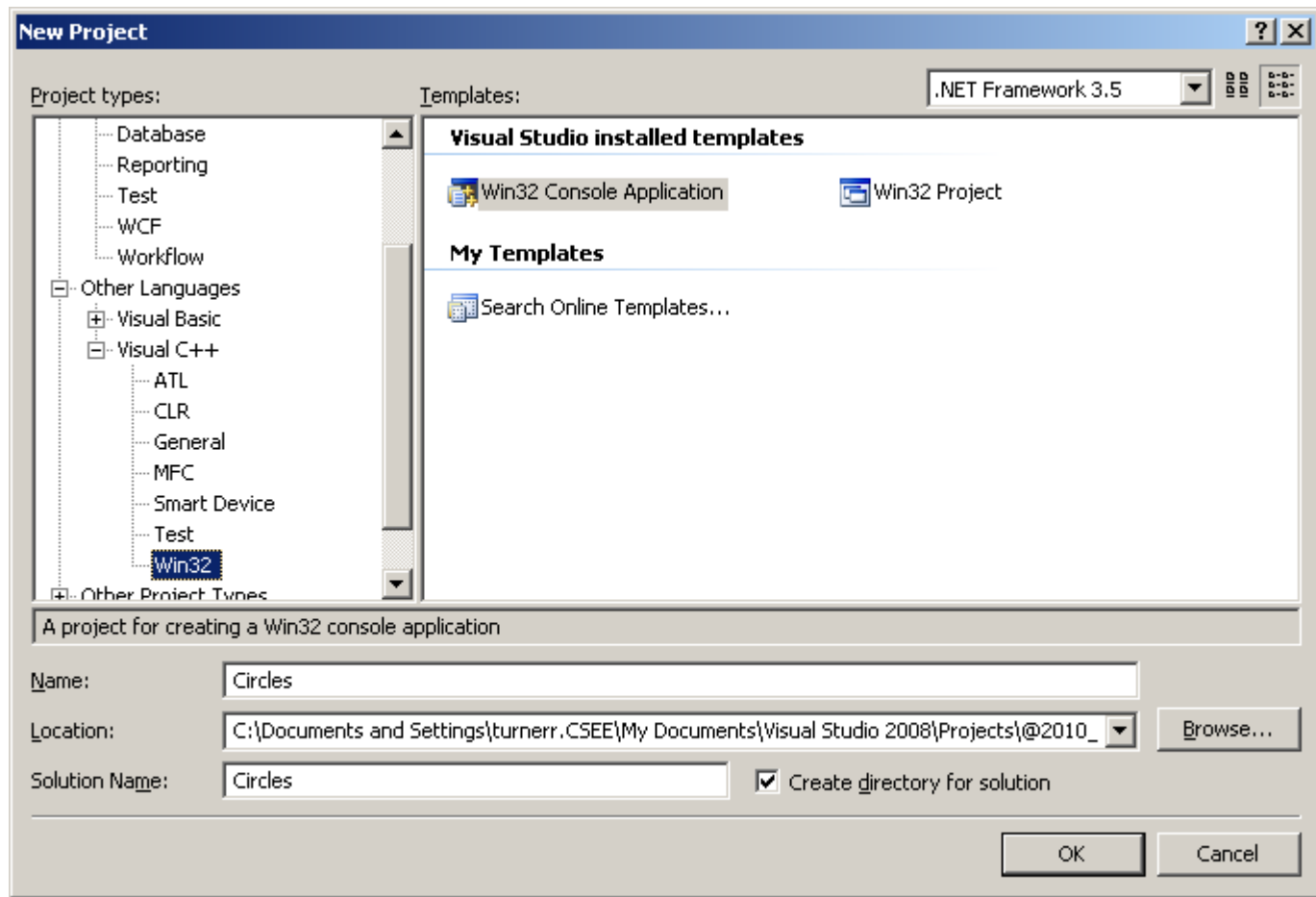
File Circle_Program.cpp



Try it!

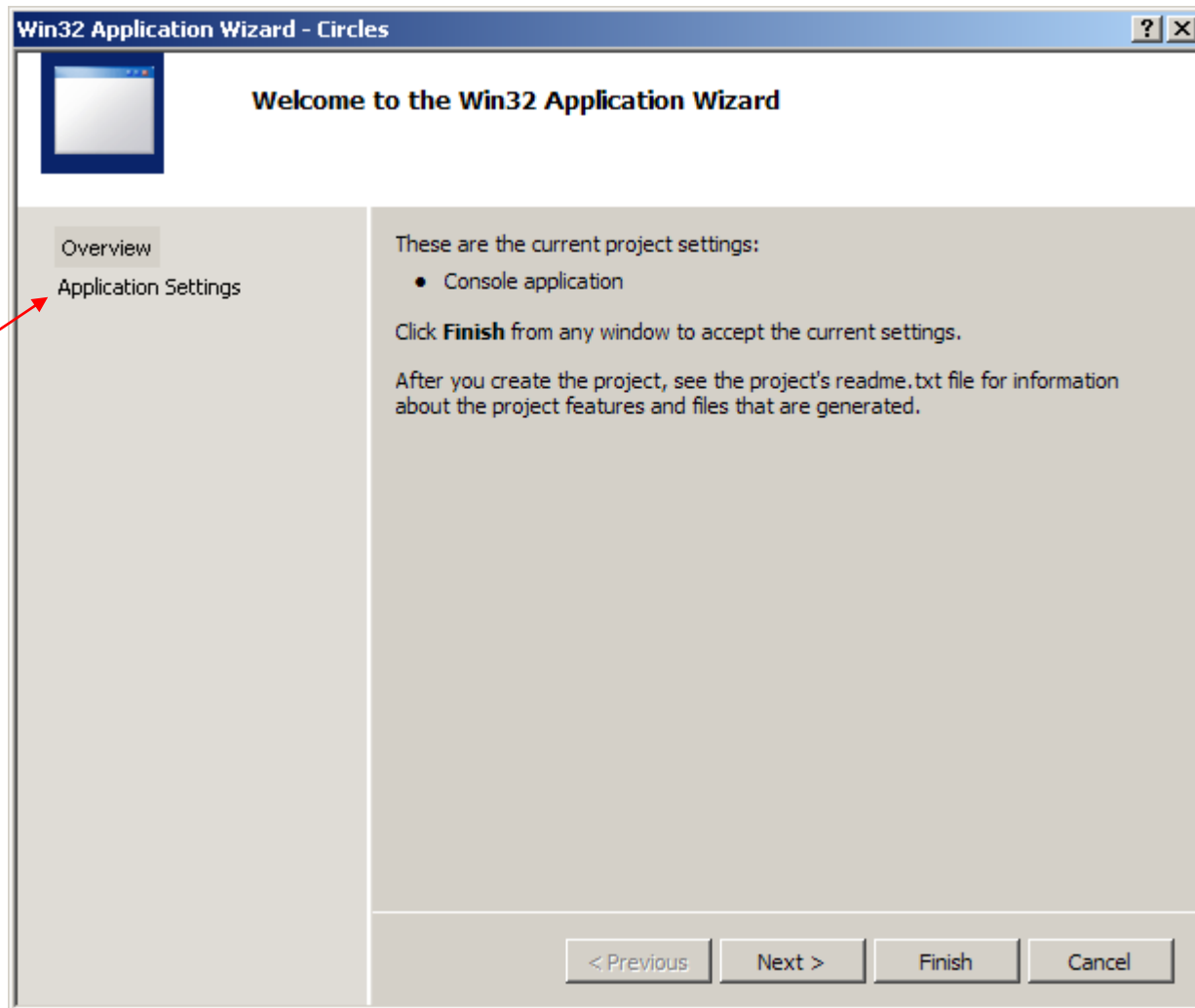
- Start up Visual Studio.
- Create a new project called "Circles"
 - Visual C++
 - Win32
 - Console Project

Creating the Project

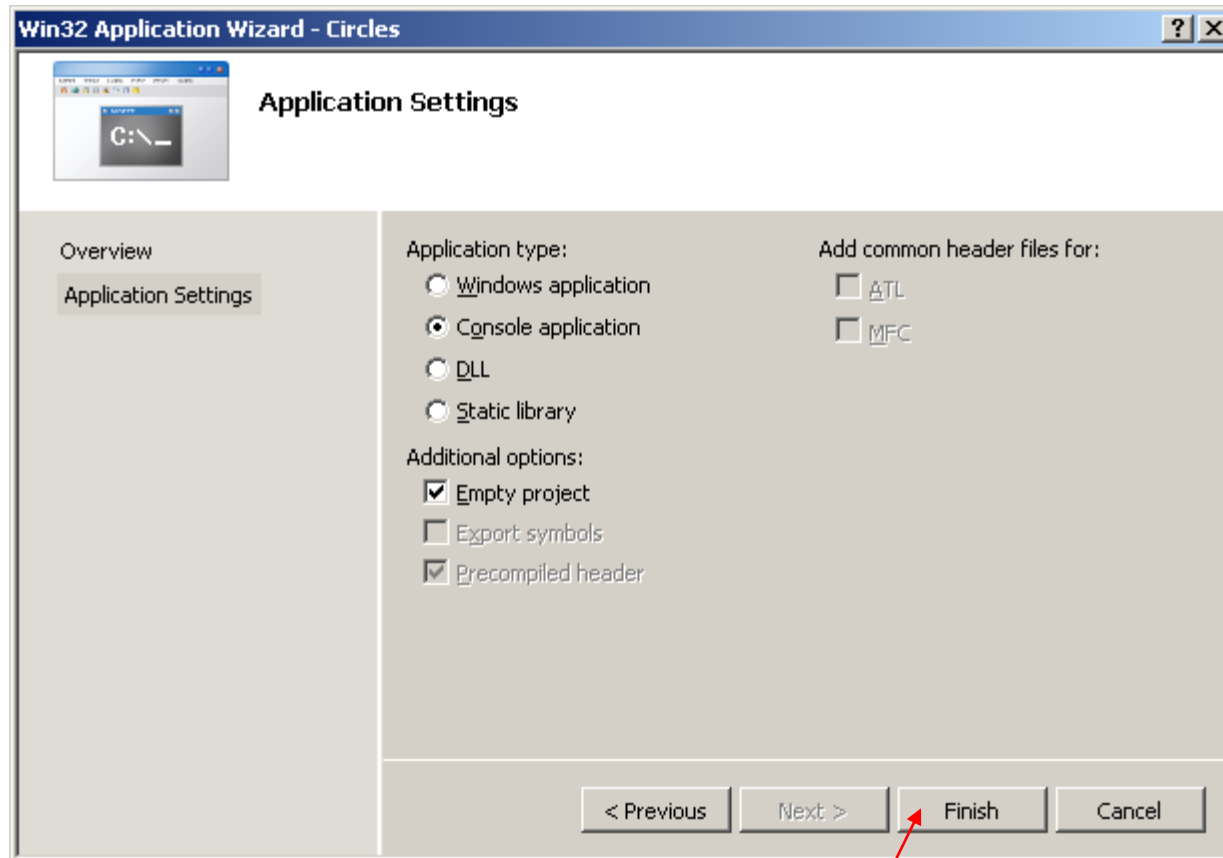


Creating the Project

Click here

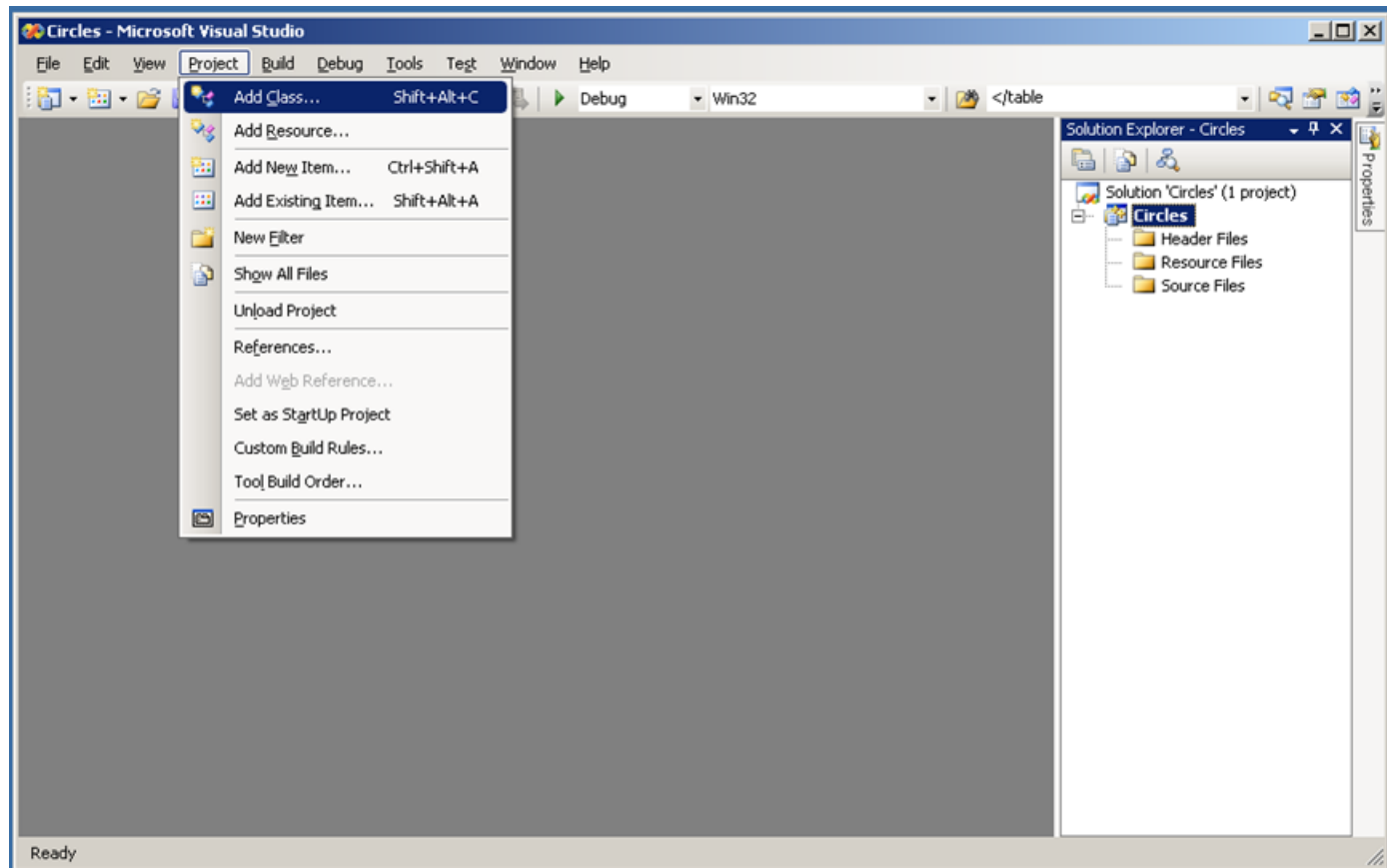


Start with an Empty Project

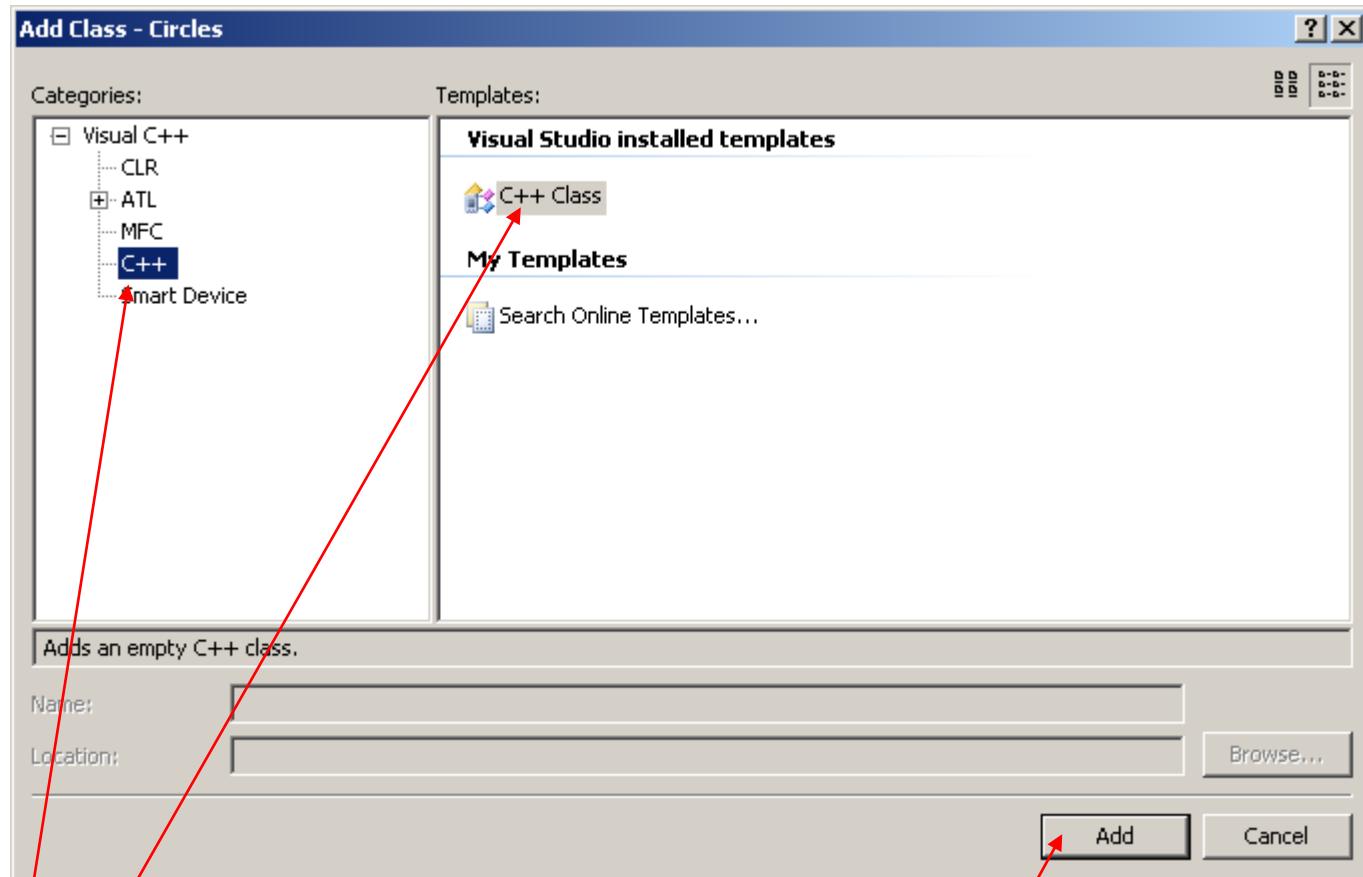


Click here

Add a Class to the Project



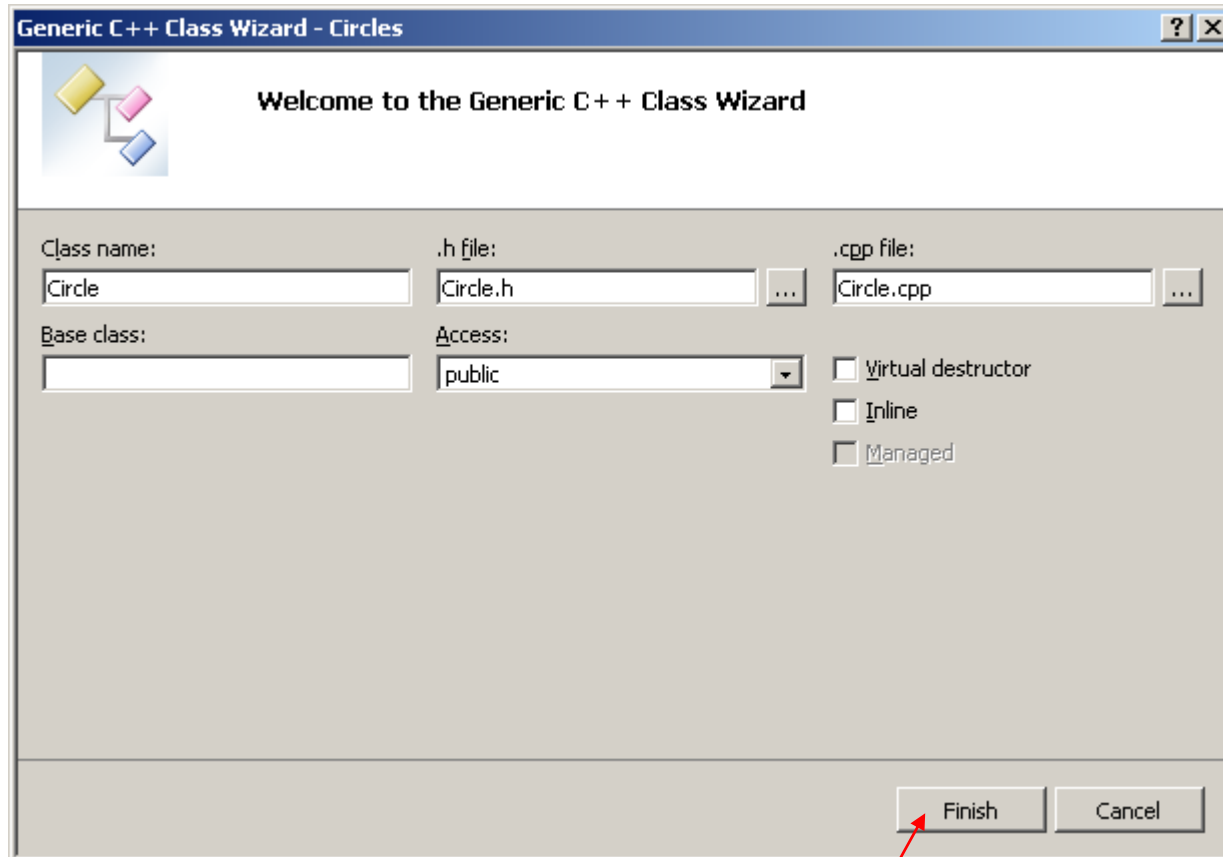
Add a Class to the Project



Be sure to select C++

Click here

Fill in the Class Name



The image shows a Windows-style dialog box titled "Generic C++ Class Wizard - Circles". It has a standard title bar with a question mark and a close button. The main area has a light gray background. At the top left is a small icon of three overlapping squares (yellow, pink, blue). To its right is the text "Welcome to the Generic C++ Class Wizard". Below this, there are several input fields and checkboxes. The "Class name:" field contains "Circle". The ".h file:" field contains "Circle.h" and has a browse button "...". The ".cpp file:" field contains "Circle.cpp" and has a browse button "...". Below these, the "Base class:" field is empty. The "Access:" field is a dropdown menu showing "public". To the right of these fields are three checkboxes: "Virtual destructor" (unchecked), "Inline" (unchecked), and "Managed" (unchecked). At the bottom right, there are two buttons: "Finish" and "Cancel". A red arrow points from the text "Click here" to the "Finish" button.

Generic C++ Class Wizard - Circles

Welcome to the Generic C++ Class Wizard

Class name: Circle .h file: Circle.hcpp file: Circle.cpp ...

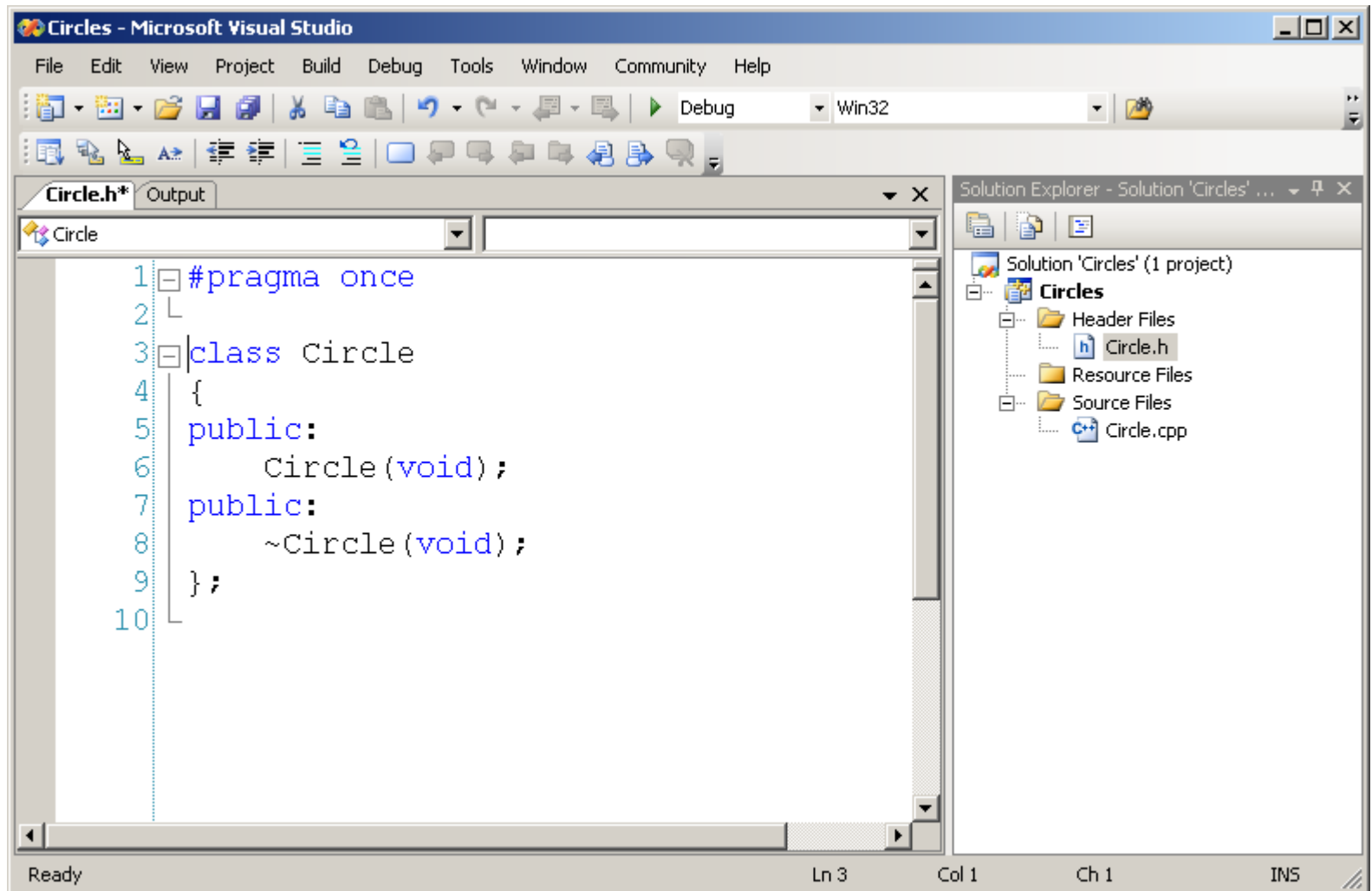
Base class: Access: public

☐ Virtual destructor
☐ Inline
☐ Managed

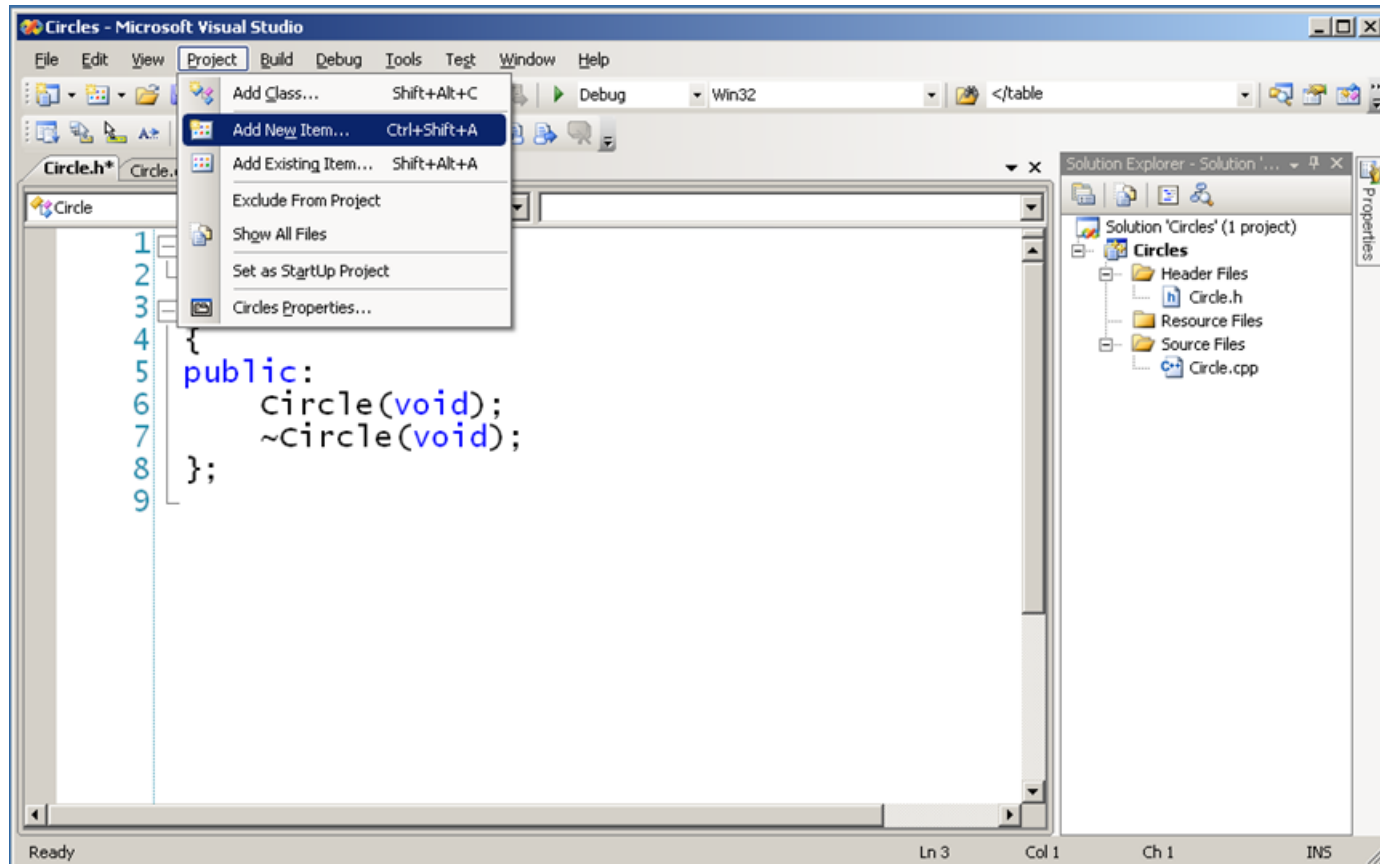
Finish Cancel

Click here

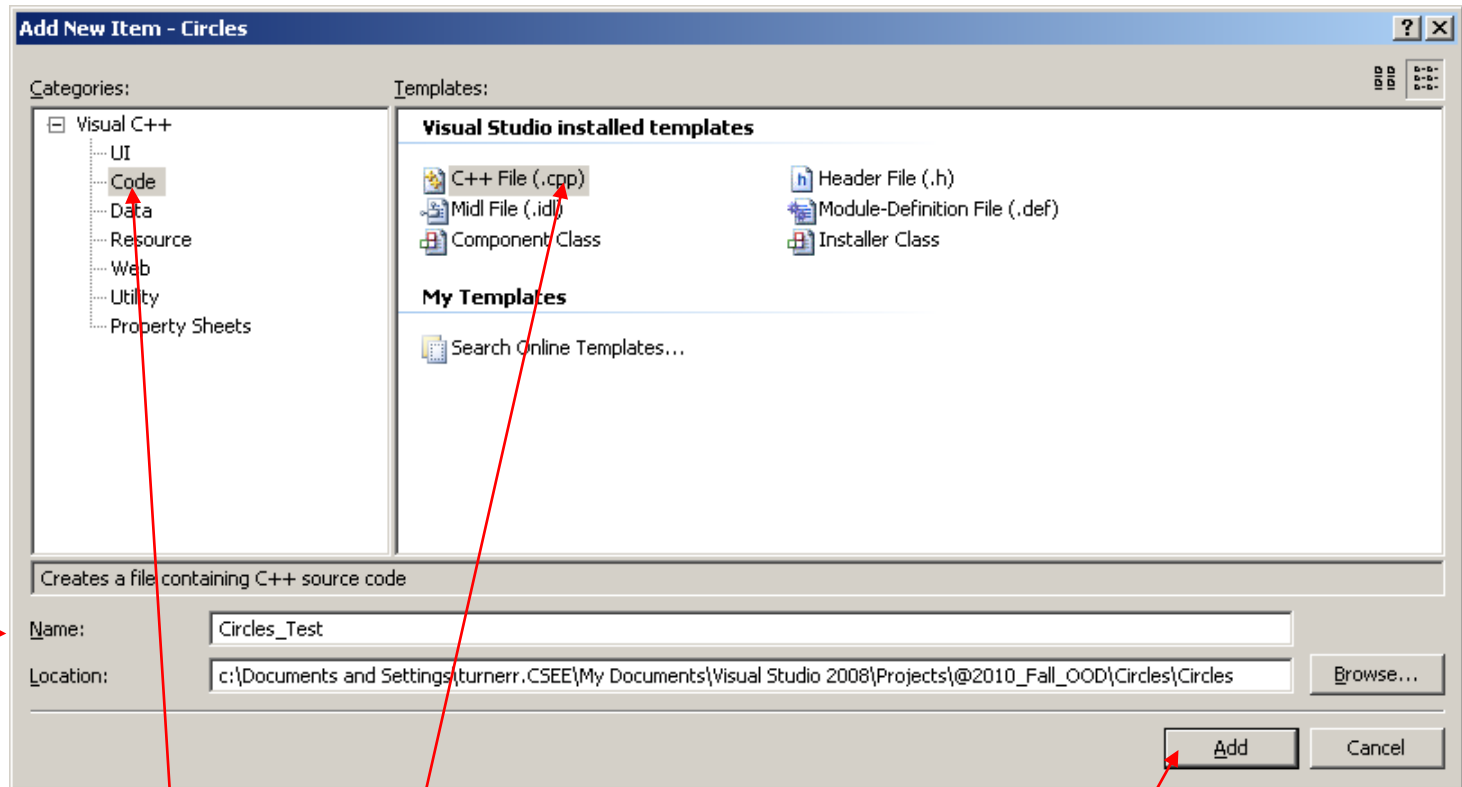
Initial Class Files are Provided



Add a file for main()



Add a file for main()

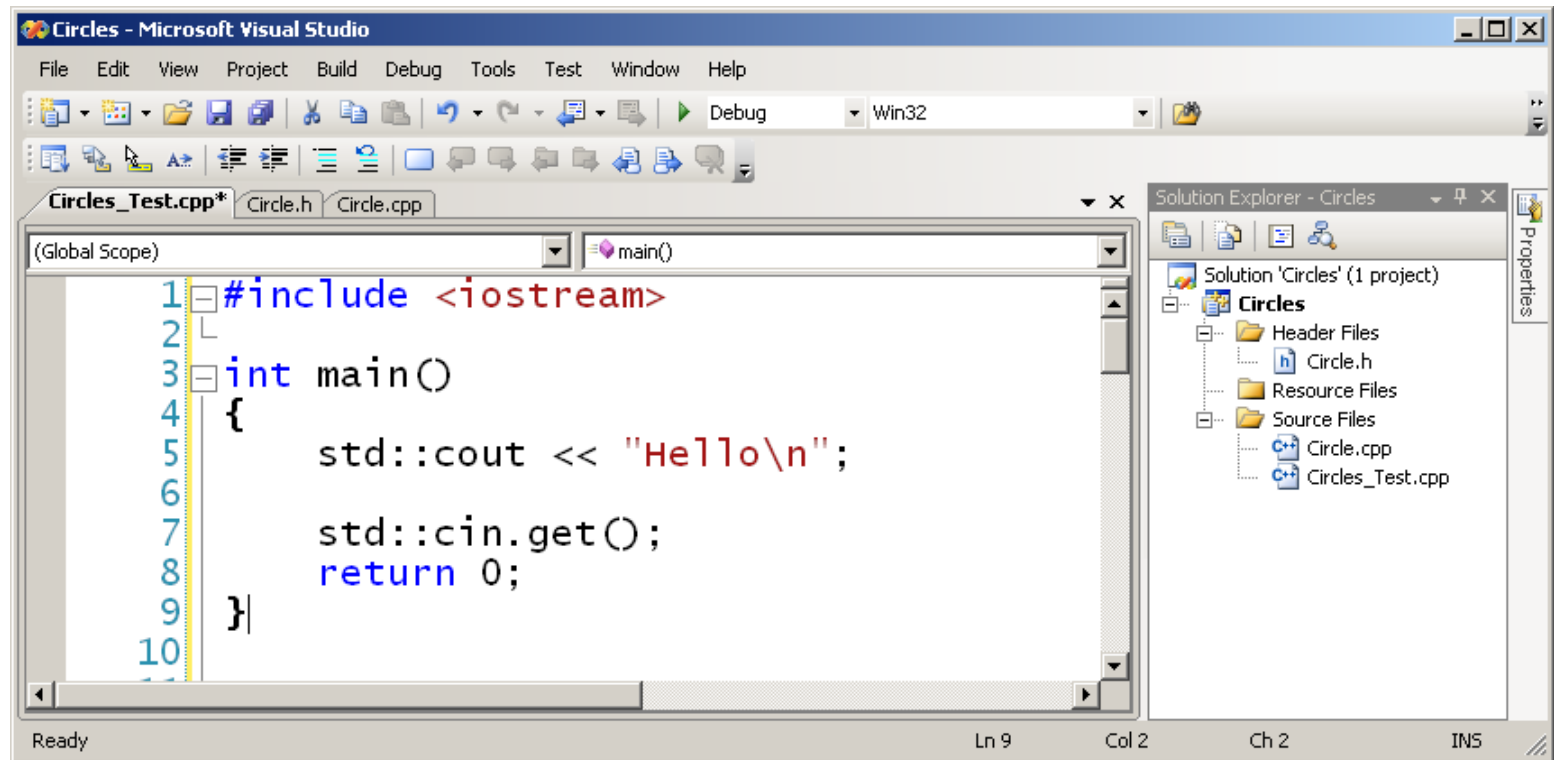


Fill in the
name

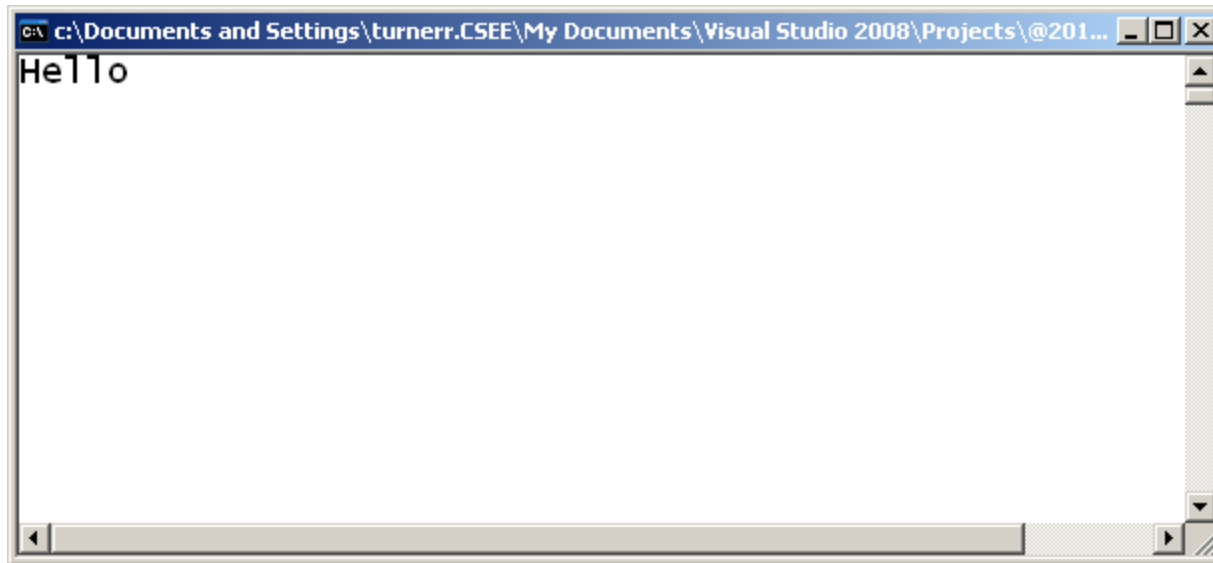
Click here

Be sure to select Code and C++

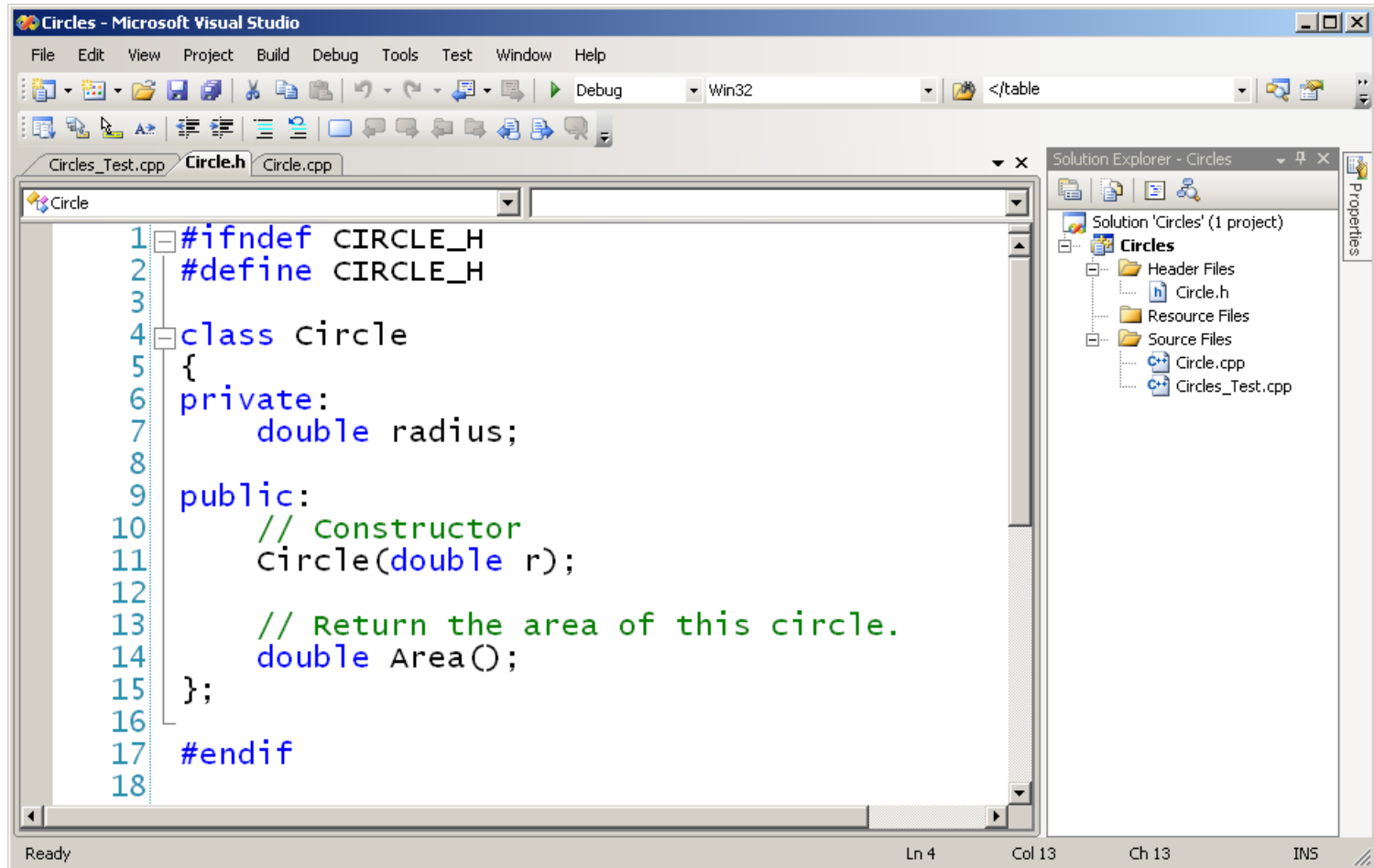
Start with a Stub



Build and Run



Class Circle

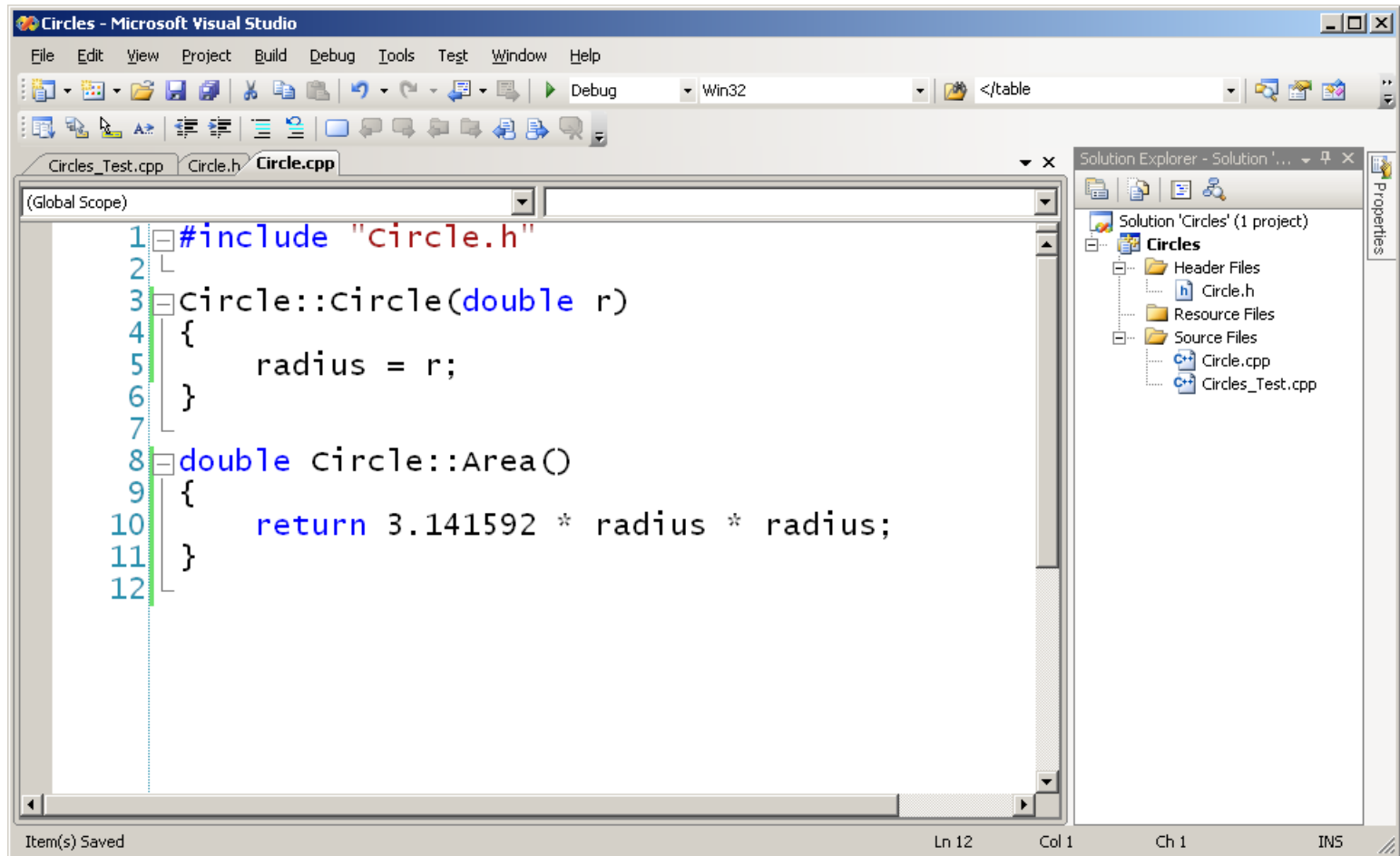


The screenshot shows the Microsoft Visual Studio IDE with the 'Circles' project open. The 'Circle.h' header file is selected in the Solution Explorer and is displayed in the main editor window. The code defines a 'Circle' class with a private 'radius' member and public methods for construction and area calculation. The status bar at the bottom indicates the cursor is at line 4, column 13.

```
1 #ifndef CIRCLE_H
2 #define CIRCLE_H
3
4 class circle
5 {
6 private:
7     double radius;
8
9 public:
10     // constructor
11     circle(double r);
12
13     // Return the area of this circle.
14     double Area();
15 };
16
17 #endif
18
```

Ready Ln 4 Col 13 Ch 13 INS

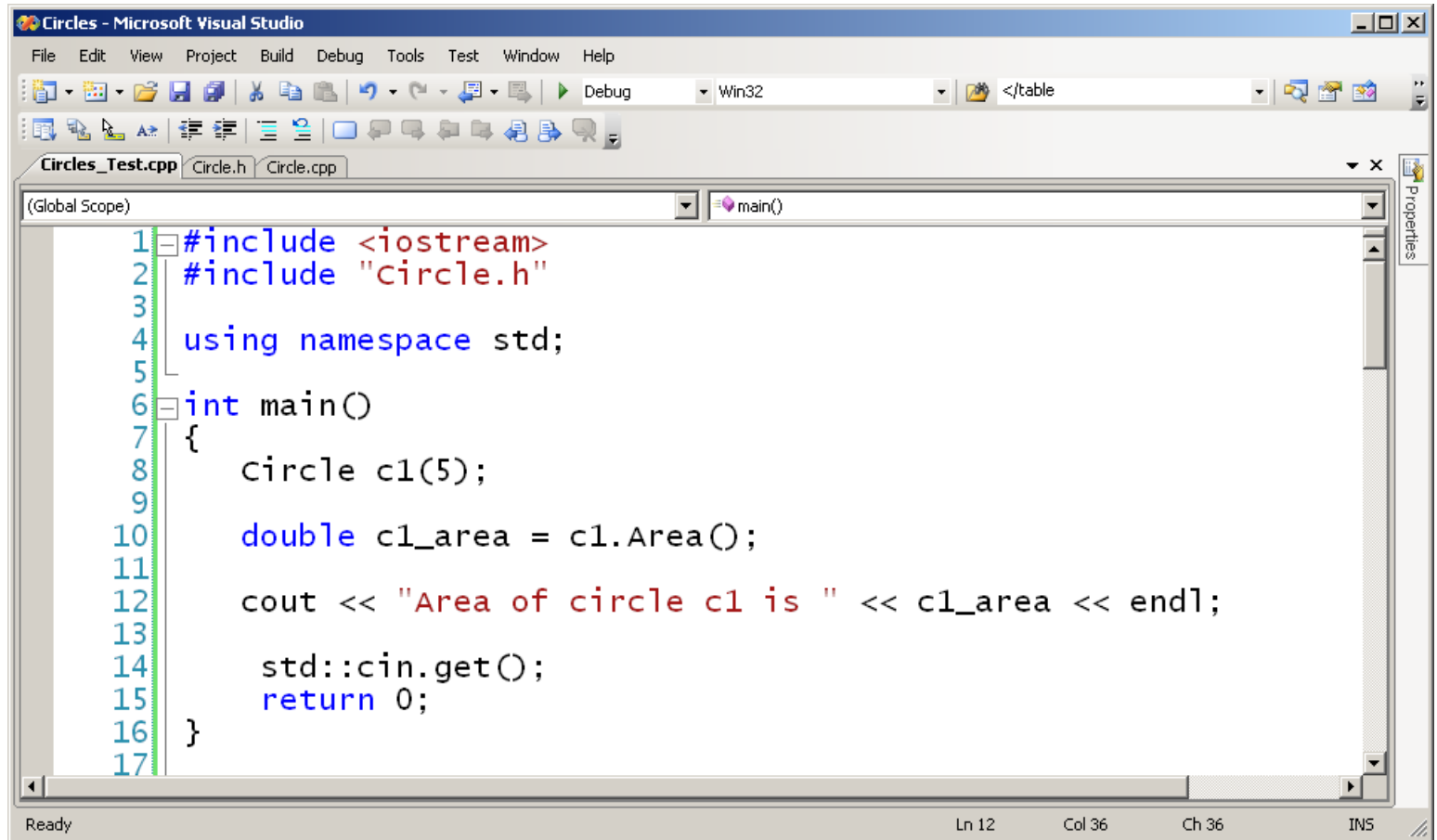
Implementation of Circle



The screenshot displays the Microsoft Visual Studio IDE with a project named 'Circles'. The main editor window shows the implementation of the 'Circle' class in 'Circle.cpp'. The code includes a header file 'Circle.h' and defines two methods: a constructor and an 'Area()' method. The 'Solution Explorer' on the right shows the project structure with 'Header Files' (Circle.h) and 'Source Files' (Circle.cpp, Circles_Test.cpp). The status bar at the bottom indicates 'Item(s) Saved', 'Ln 12', 'Col 1', 'Ch 1', and 'INS'.

```
1 #include "Circle.h"
2
3 Circle::Circle(double r)
4 {
5     radius = r;
6 }
7
8 double Circle::Area()
9 {
10     return 3.141592 * radius * radius;
11 }
12
```

The Main Program

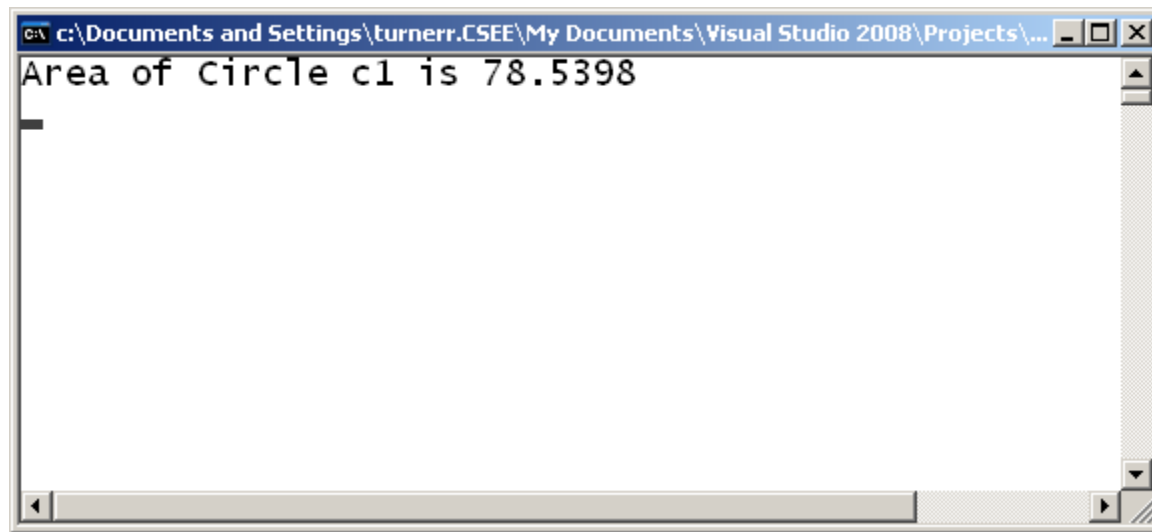


The screenshot shows the Microsoft Visual Studio IDE with a project named 'Circles'. The 'Circles_Test.cpp' file is open, displaying the following C++ code:

```
1 #include <iostream>
2 #include "circle.h"
3
4 using namespace std;
5
6 int main()
7 {
8     circle c1(5);
9
10    double c1_area = c1.Area();
11
12    cout << "Area of circle c1 is " << c1_area << endl;
13
14    std::cin.get();
15    return 0;
16 }
17
```

The code is color-coded: keywords are blue, standard library names are red, and string literals are red. The IDE interface includes a menu bar (File, Edit, View, Project, Build, Debug, Tools, Test, Window, Help), a toolbar, and a status bar at the bottom showing 'Ready', 'Ln 12', 'Col 36', 'Ch 36', and 'INS'.

Build and Run





Example: Creating multiple objects of the same type

```
#include <iostream>
#include "Circle.h"

using namespace std;

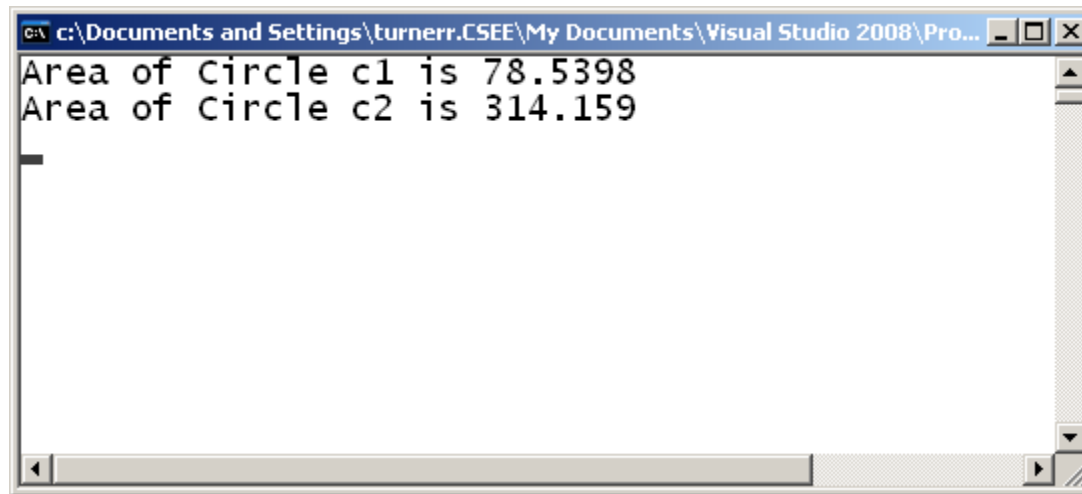
int main()
{
    Circle c1(5);
    Circle c2(10);

    double c1_area = c1.Area();
    cout << "Area of circle c1 is " << c1_area << endl;

    cout << "Area of circle c2 is " << c2.Area() << endl;

    cin.get();          // Keep the window open.
    return 0;
}
```

Circles Program Running



A screenshot of a Windows command prompt window. The title bar shows the path: `c:\Documents and Settings\turnerr.CSEE\My Documents\Visual Studio 2008\Pro...`. The window contains two lines of text: `Area of Circle c1 is 78.5398` and `Area of Circle c2 is 314.159`. The cursor is positioned at the start of a new line below the second line of text.

```
c:\Documents and Settings\turnerr.CSEE\My Documents\Visual Studio 2008\Pro...
Area of Circle c1 is 78.5398
Area of Circle c2 is 314.159

```



Constructor

When function `main()` is entered

- The Circle objects, `c1` and `c2`, are created.
 - Space allocated on the stack.
- The constructor is invoked for `c1` with the argument 5.
 - Initializes the object's data
- The constructor is invoked for `c2` with the argument 10.
 - Initializes the object's data



Multiple Constructors

A class can have any number of constructors.

All must have different *signatures*.

(The pattern of types used as parameters.)

This is called *overloading* a method.

Applies to *all* methods in C++. Not just constructors.

Different *names* for parameters don't matter,
Only the types.



Default Constructor

If you don't write a constructor for a class, the compiler creates a default constructor.

The default constructor is public and has no arguments.

```
c = new Circle();
```

Essentially does nothing.

Class member variables will be uninitialized.



Destructor

- Each class can also have a special method called a *destructor*.
- Invoked when the object is deallocated.
- Free any resources used by the object.
- Name is class name preceded by ~
 - ~Circle
- No return value.
- No parameters.



Assignment

- Do today's examples for yourself if you have not done them in class.
- Read Chapter 10 through page 529.