Object-Oriented Programming

Introduction to Microsoft Visual Studio (2012)

Microsoft Visual Studio (MVS)

- is a design environment for creating, compiling, linking, debugiing and testing programs.

- Editor – interactive environment for creating and editing source code (**.cpp*, **.h*).
- Compiler – converts source code into machine language, and detects and reports errors in the comppilation process (**.obj*).
- Linker – combines the various modules generated by the compiler, adds required modules from program libraries and creates an executable file.
- Libraries – extend the C++ language by providing support to operations which are not part of the language.

Projects

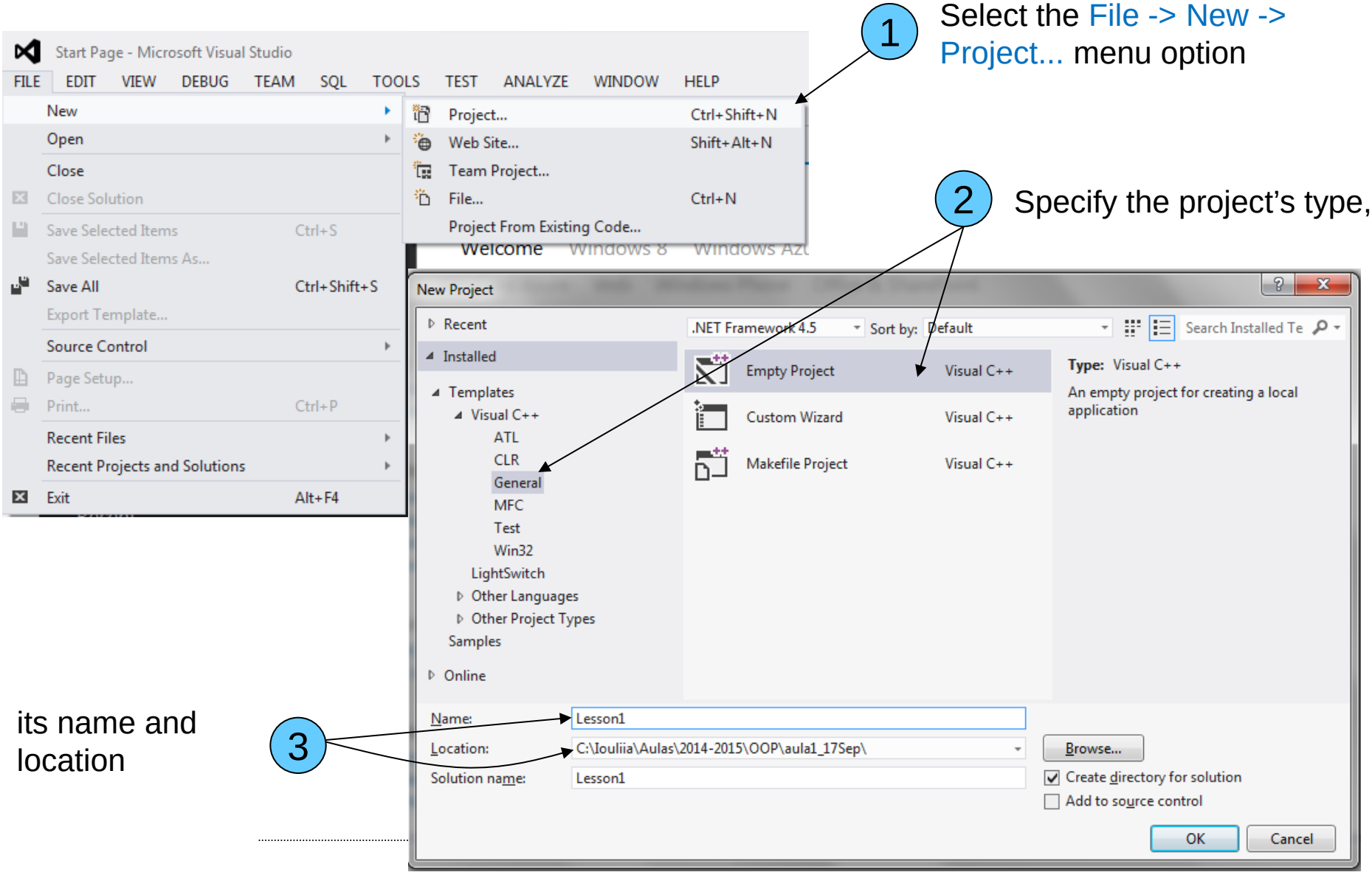
- A project (*.vcxproj*) is a program of some kind.
- A solution (**.sln*) stores all the information relating to a project.
- When you create a project, a solution is created automatically.
- When you have created a project along with its solution, you can add further projects to the same solution. One of those projects is marked as “StartUp”.
- Generally, each of your projects should have its own solution.

Defining a project

A project definition includes:

- A project name
- A list of all the source files
- A definition of what kind of program is to be built from the source files
- The options set for the editor, the compiler, the linker and other components of MVS
- The windows to be displayed when the project is opened

Creating a new project



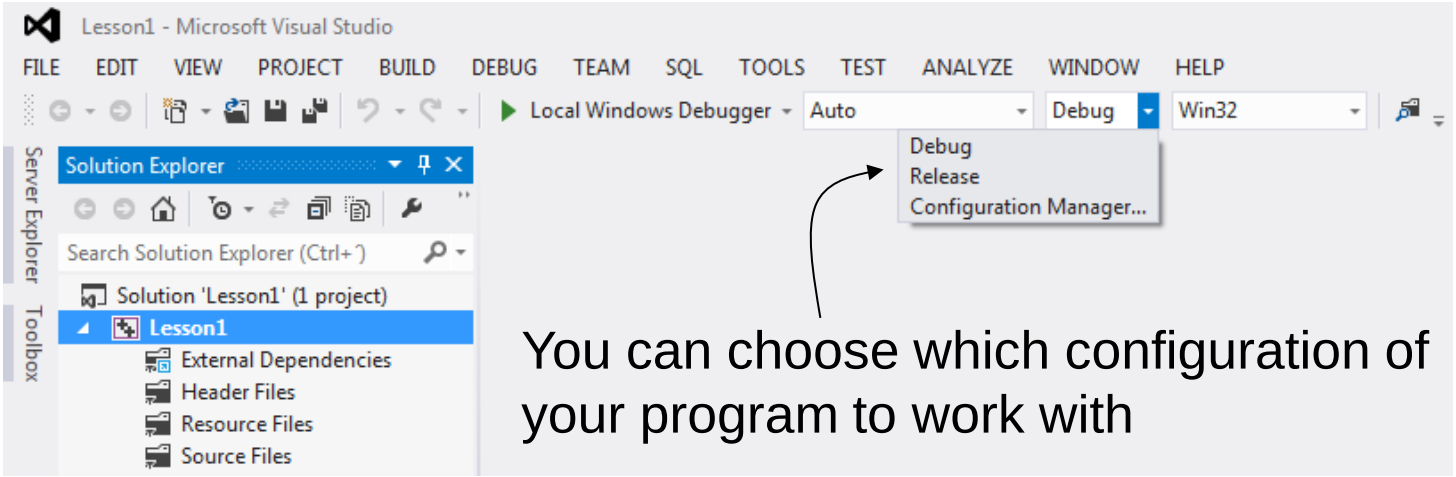
Project configurations

Project configurations determine how your source code is to be processed during the compile and link stages.

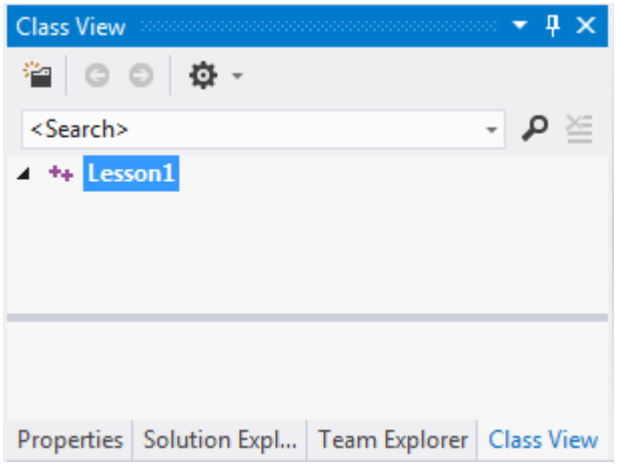
When you create a new solution, MVS will automatically create configurations for producing two versions of your application.

One includes information which will help you to debug the program and is called Debug.

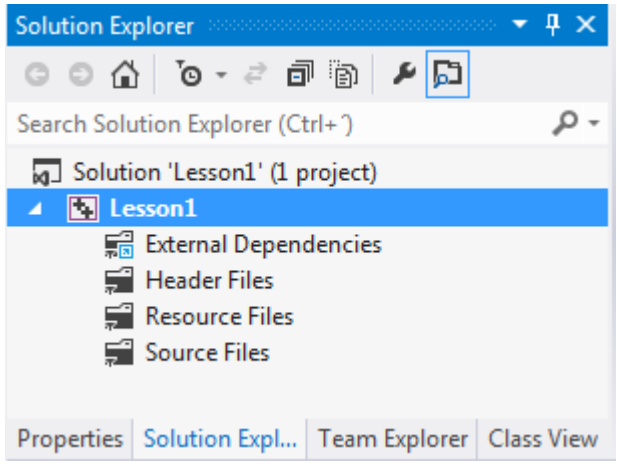
The other, called Release, has no debug information included and has the code optimization options for the compiler turned on to provide you with the most efficient executable module.



Project views

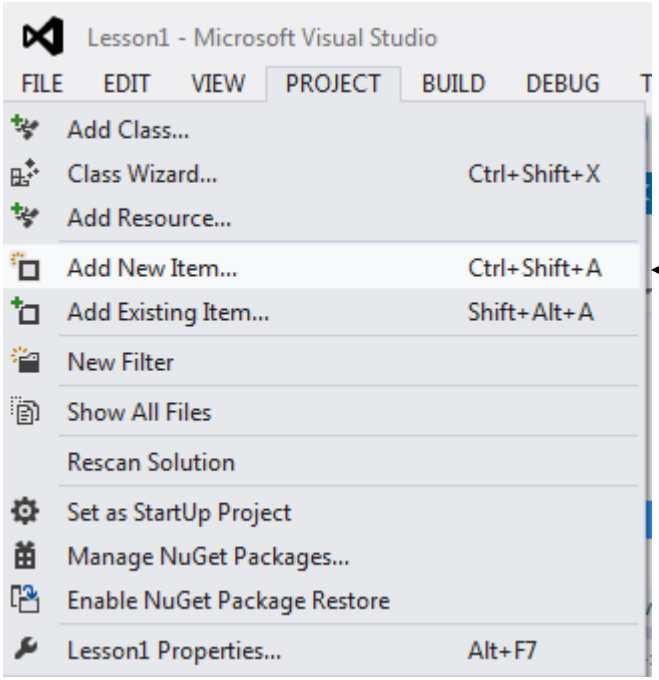


- The Class View displays the classes defined in your project and will also show the contents of each class.
- We don't have any classes in this application, so the view is empty.



- The Solution Explorer shows the source program files that make up your project.
- You can display the contents of any file by clicking on a file name. At the moment we have no source files.
- There are three groups of files: Header Files (definition of class interfaces), Resource Files (menus, icons, etc.) and Source Files (source code)

Entering your first program

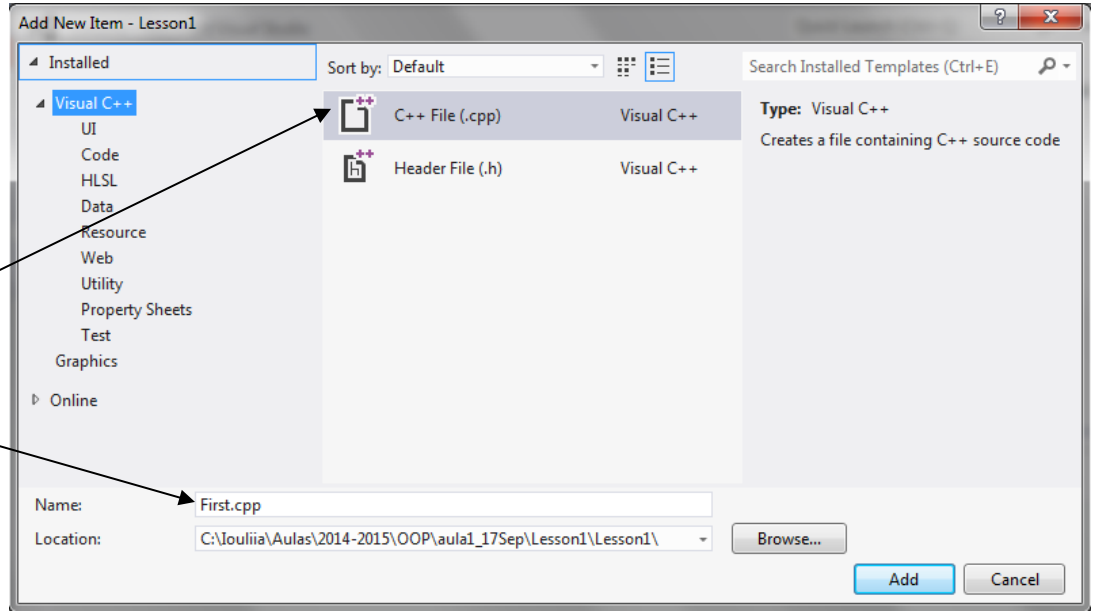


1

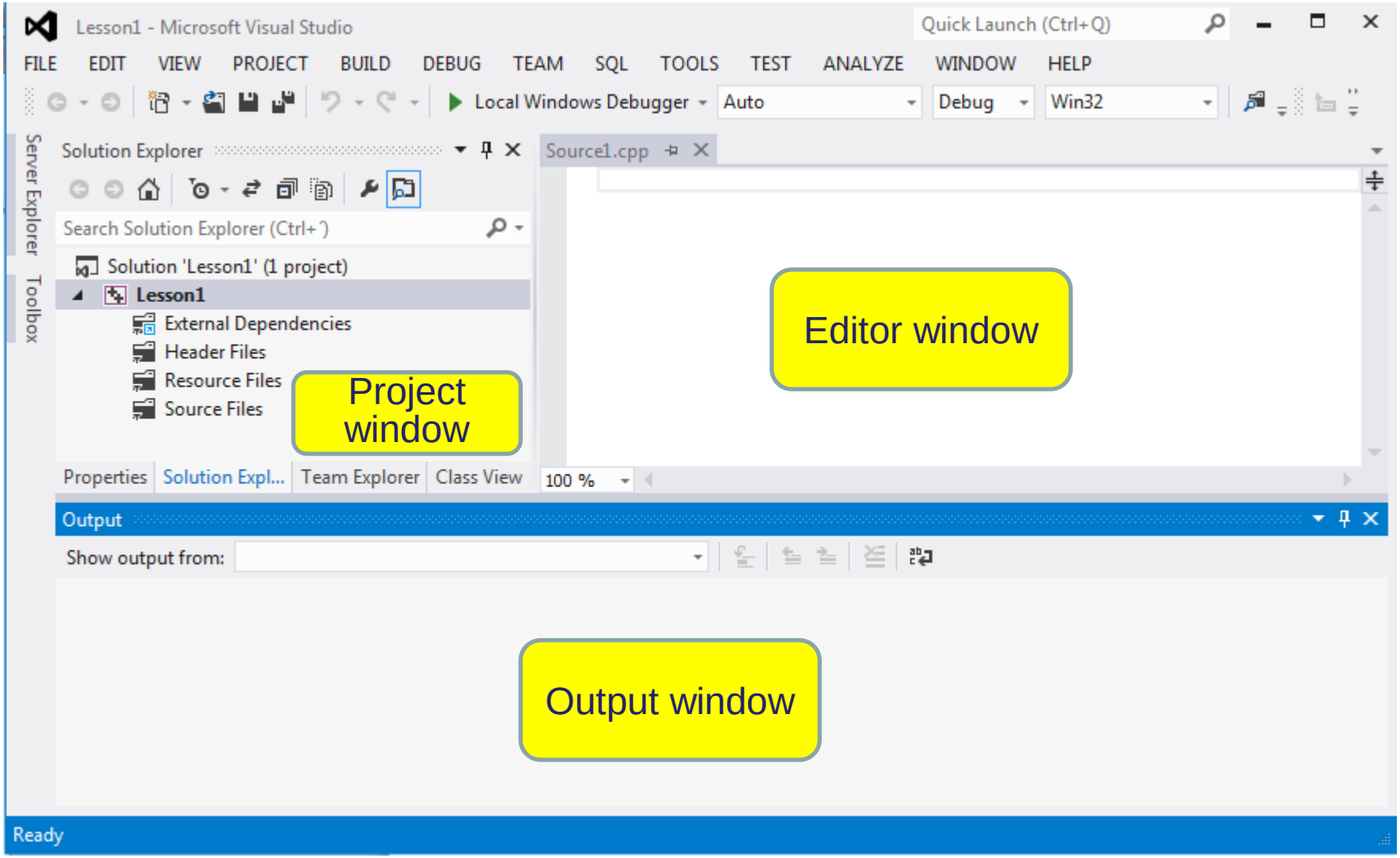
Select the Project -> Add New Item... menu option

Select the file type and enter the file name

2

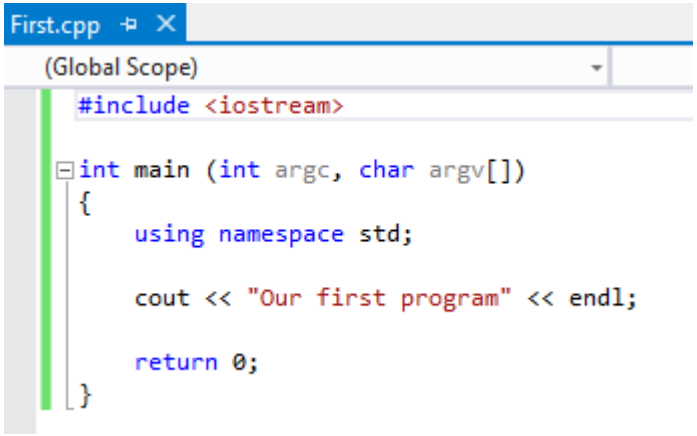


MVS GUI


Lesson1 - Microsoft Visual Studio
Quick Launch (Ctrl+Q)
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ANALYZE WINDOW HELP
Local Windows Debugger
Auto
Debug
Win32
Solution Explorer
Source1.cpp
Server Explorer
Toolbox
Search Solution Explorer (Ctrl+;)
Solution 'Lesson1' (1 project)
Lesson1
External Dependencies
Header Files
Resource Files
Source Files
Project window
Editor window
Properties
Solution Expl...
Team Explorer
Class View
100 %
Output
Show output from:
Output window
Ready

Entering your first program (cont.)

Enter the following code to *First.cpp*:



The line *#include <iostream>* includes the *iostream* header file which declares objects that control data input/output in C++.

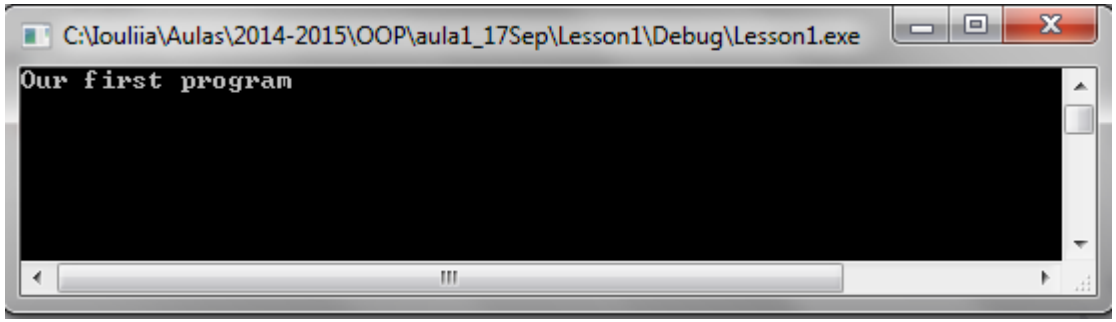
cout (console output) is an object that accepts all data bound for standard output. To send data to standard output, you use the operator <<. With iostream objects, the operator << means “send to.”

endl (end line) is a manipulator that terminates a line and flushes the buffer.

All of the Standard C++ libraries are wrapped in a single namespace, which is *std* (for “standard”). The directive *using namespace std;* opens access to all the elements declared in *std* namespace, including *cout* and *endl*.

Building and debugging your project

To compile, link and execute your program, press Ctrl-F5.



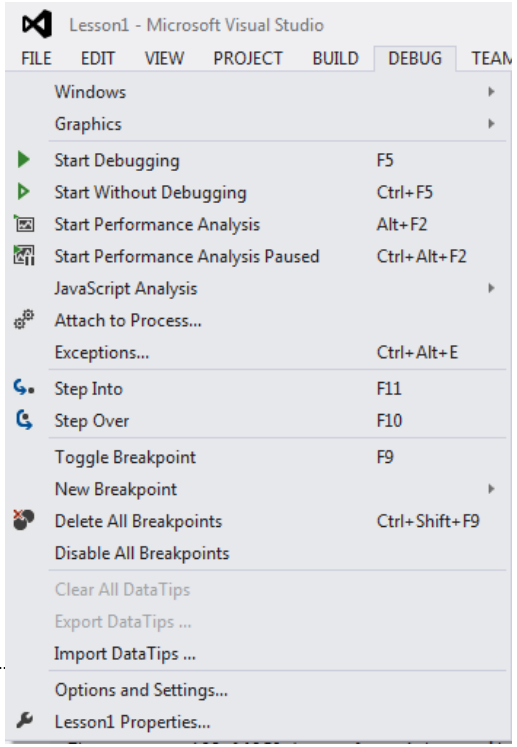
To execute the program step-by-step (not stepping into functions) press F10.

To execute the program step-by-step (stepping into functions) press F11.

```
    cout << "Our first program" << endl;
```

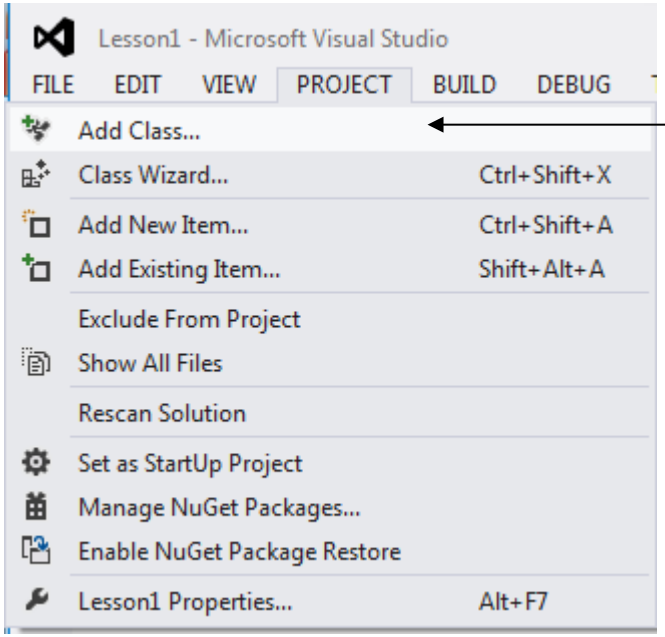
To insert breakpoint select the code line and press F9.

```
    cout << "Our first program" << endl;
```



Various debug options are available in the menu *Debug*.

Creating a class

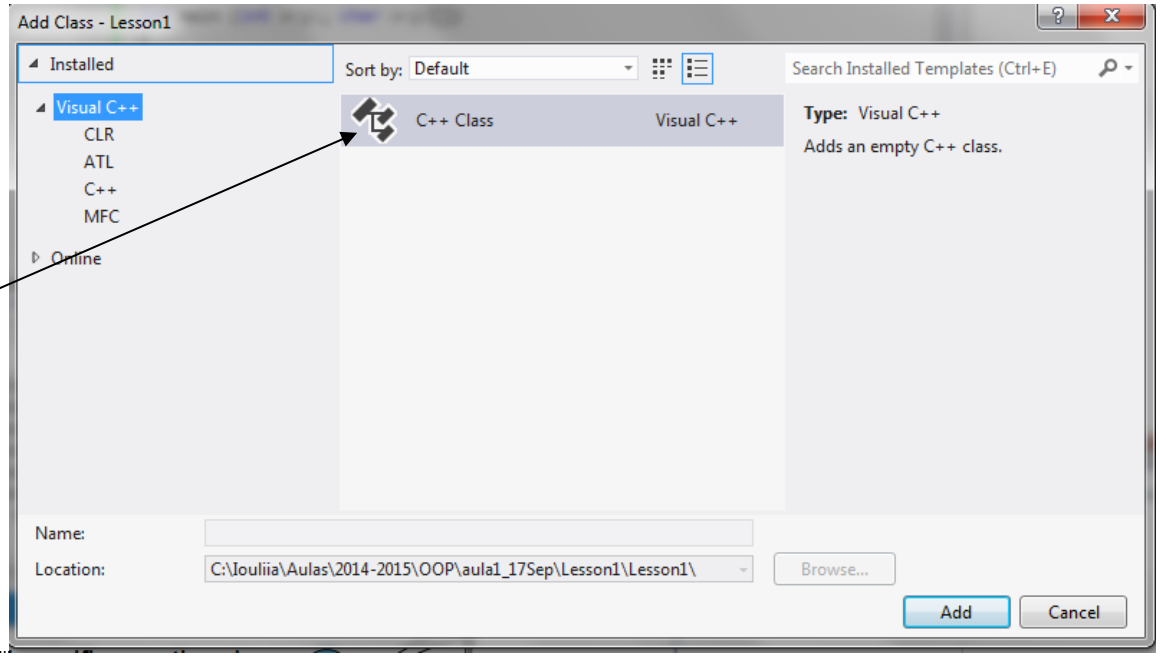


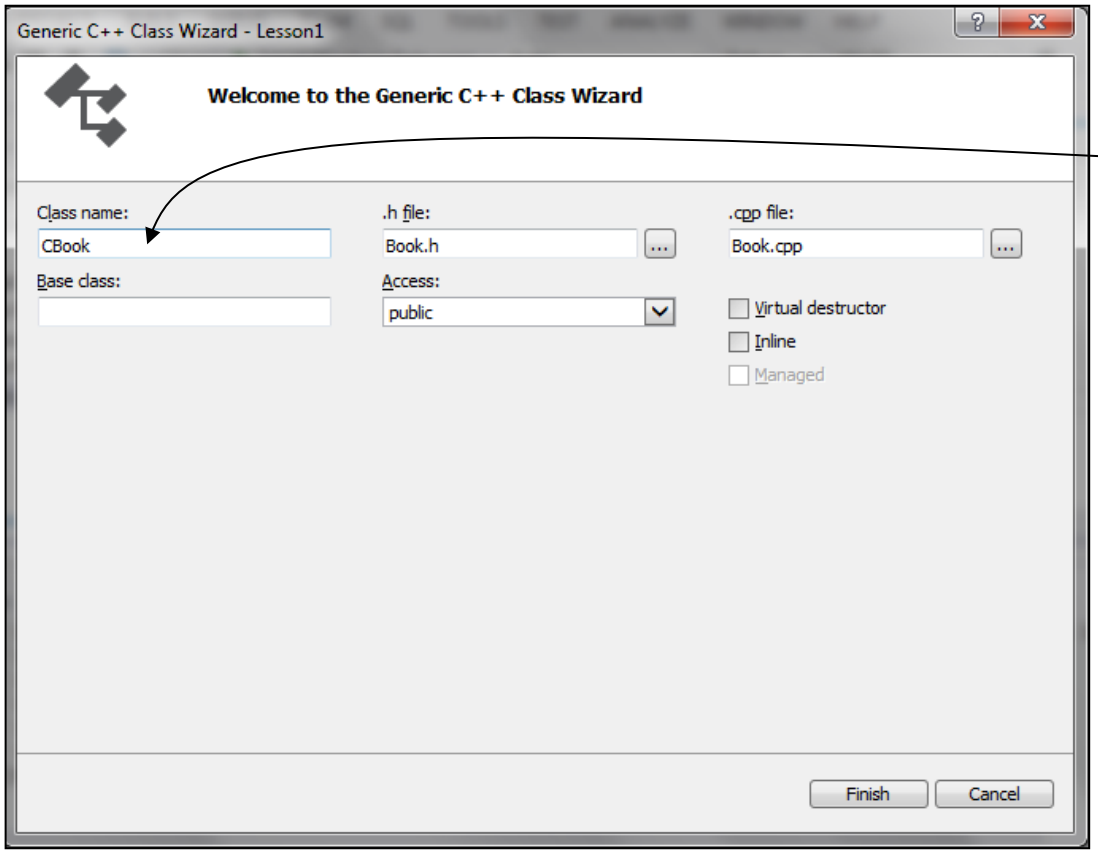
1

Select the Project -> Add Class... menu option

Specify the class type

2





3

Enter the class name

These files declare two functions: the default constructor (CBook::CBook()) and the destructor (CBook::~CBook()) .

Book.h* Book.cpp*

(Global Scope)

```
#pragma once
class CBook
{
public:
    CBook(void);
    ~CBook(void);
};
```

Two files will be created automatically: *Book.h* – with class interface e *Book.cpp* with class implementation.

4

Book.h* Book.cpp*

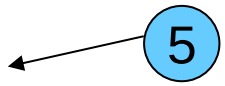
(Global Scope)

```
#include "Book.h"


CBook::CBook(void)
{
}


CBook::~CBook(void)
{
}
```

```
private:
        unsigned m_uYear;
        std::string m_sTitle;
```



Add to the class CBook two data members: the year of publication (m_uYear) and the title (m_sTitle).

The Standard C++ string class is designed to take care of (and hide) all the low-level manipulations of character arrays that were previously required of the C programmer.

To use strings you include the C++ header file <string>. The string class is in the namespace std.

Because of operator overloading, the syntax for using strings is quite intuitive. For instance, you can assign to any string object using '='. This replaces the previous contents of the string with whatever is on the right-hand side, and you don't have to worry about what happens to the previous contents – that's handled automatically for you. To combine strings you simply use the '+' operator, which also allows you to combine character arrays with strings.

```
private:
        unsigned m_uYear;
        std::string m_sTitle;
```

The values of m_uYear and m_sTitle are not defined. These data members must be initialized. The initialization is performed by a special function called the constructor.

If a class has a constructor, the compiler automatically calls that constructor at the point an object is created, before client programmers can get their hands on the object.

The class constructor has the same name as the name of the class. Like any function, the constructor can have arguments to allow you to specify how an object is created and give it initialization values.

Constructors and destructors return nothing.

6 Let us add the following intitialization code to the constructor:

```
CBook::CBook(std::string title, unsigned year)
{
    m_uYear = year;
    m_sTitle = title;
}
```

```
#pragma once
#include <string>

class CBook
{
public:
    CBook(std::string title, unsigned year);
    ~CBook(void);

    void Print();

private:
        unsigned m_uYear;
        std::string m_sTitle;
};
```

7

Let us include a member function to the class CBook:

void Print();

This function can be implemented as follows (do not forget to include the iostream header file):

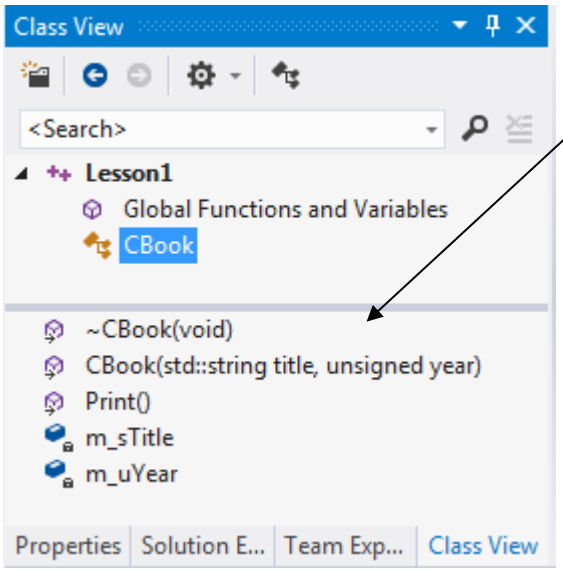
8

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

CBook::CBook(std::string title, unsigned year)
{
    m_uYear = year;
    m_sTitle = title;
}


CBook::~CBook(void)
{
}

void CBook::Print()
{
    using namespace std;
    cout << "Title: " << m_sTitle << "; Year: " << m_uYear << endl;
}
```



9 At the moment the class CBook has the following members:

10 Finally, the main function is modified so as to test the CBook class.

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

int main (int argc, char argv[])
{
    using namespace std;

    cout << "Our first program" << endl;

    CBook* pBook;
    CBook POO ("C++", 2014);

    pBook = &POO;

    pBook->Print();
    POO.Print();

    return 0;
}
```

The results should be the following:

11

```
Our first program
Title: C++; Year: 2014
Title: C++; Year: 2014
```