Компиляция библиотеки тестирования Google Test

Google C++ Testing Framework (или Google Test) – это библиотека для модульного тестирования (unit testing) на языке C++.

Чтобы использовать данную библиотеку, необходимо, прежде всего, **один раз** скомпилировать эту библиотеку, а затем **подключать ее к каждому проекту**, в котором предполагается выполнение модульного тестирования.

Чтобы скомпилировать библиотеку Google Test для использования в среде Visual Studio, нужно выполнить следующие действия.

1. Скачать пакет.

Перейти по ссылке https://code.google.com/p/googletest на стартовую страницу github.com/google/googletest.

На этой страничке нажать на зеленую кнопку **Clone or download** и в открывшемся окне выбрать **Download ZIP**. Распаковать полученный файл в папку, где будет размещаться проект.

2. Перейти в папку …\googletest\googletest\msvc. В этой папке для файла gtest.sln снять, при необходимости, в свойствах файла атрибут «только чтение», после чего выполнить файл gtest.sln. Запустится соответствующая Visual Studio, в ней появится окно One Way Upgrade – обновление версии проекта; нажать OK. Может появиться окно Security Warnings; в нем снять галочку у Ask me… и нажать OK.

После выполнения Upgrade'а в браузере откроется страница с отчетом о результатах Upgrade'а, в котором не должно быть ошибок (errors), но могут быть предупреждения (warnings). Закрыть эту страницу.

3. В окне Solution Explorer откроются 4 проекта; скомпилировать все решение (Solution 'gtest') для режимов Debug (илл. 1) и Release.

Иллюстрация 1

После успешной компиляции закрыть проект.

Будут созданы необходимые библиотеки, которые потребуются в дальнейшем для отладки. В папке …\Projects\gtest-1.7.0 будут использоваться следующие папки и файлы:

…\Projects\gtest-1.7.0**include** – содержит необходимые файлы-заголовки,

…\Projects\gtest-1.7.0\msvc\gtest\Debug**gtestd.lib** – библиотека для тестирования в режиме Debug,

…\Projects\gtest-1.7.0\msvc\gtest\Release**gtest.lib** – библиотека для тестирования в режиме Release.

Выполнение лабораторного практикума

При выполнении каждой темы лабораторного практикума необходимо создавать решение (Solution), содержащее несколько проектов. Каждый проект отвечает за определенную часть задания, например, основные классы, определяемые заданием; прикладная программа, использующая разработанные классы; программы, выполняющие тестирование разработанных классов (одна или несколько). При этом чтобы использовать разработанные классы в разных проектах, целесообразно создать статическую библиотеку, содержащую эти классы.

Так, для первого задания в среде Visual Studio в одном решении следует создать три проекта:

1 – статическая библиотека, в которой должны находиться скомпилированные методы класса,

2 – консольное приложение, содержащее прикладную программу (диалоговую программу для тестирования разработанных классов),

3 – консольное приложение, использующее библиотеку тестирования Google Test.

Первый создаваемый проект определяет имя всего решения. Назовем, для определенности, проект создания библиотеки как `Library`; проект, содержащий прикладную программу – как `Application`; проект для тестирования – как `Testing`; тогда после создания всех приложений, при условии, что первым создается проект для библиотеки, в окне Solution Explorer мы получим (илл. 2):

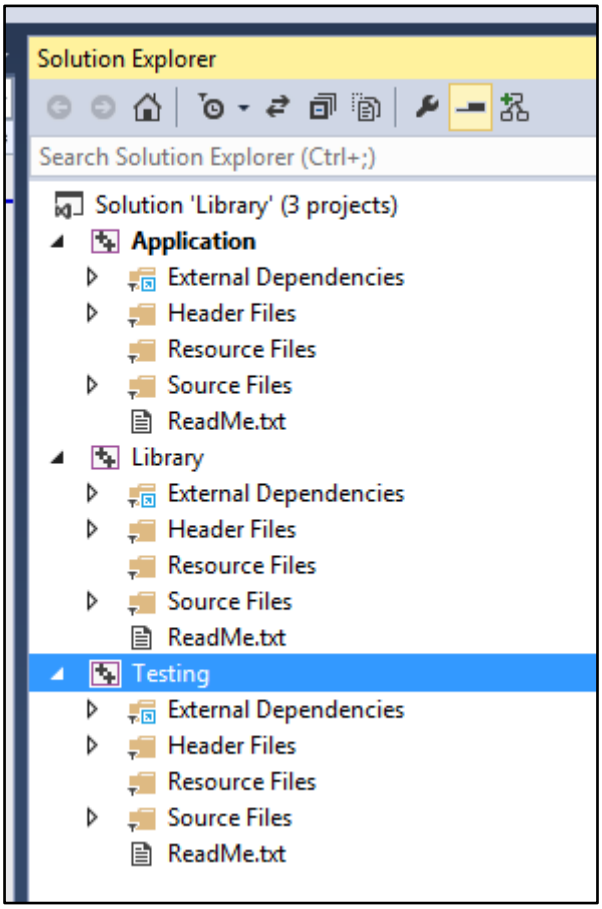

Иллюстрация 2

Создание проекта для статической библиотеки

1. Проект для статической библиотеки создается как обычное консольное приложение для C++, но в окне Application Settings следует выбрать **Static Library** (статическая библиотека) и нажать Finish (илл. 3). Проект будет создан (илл. 4).

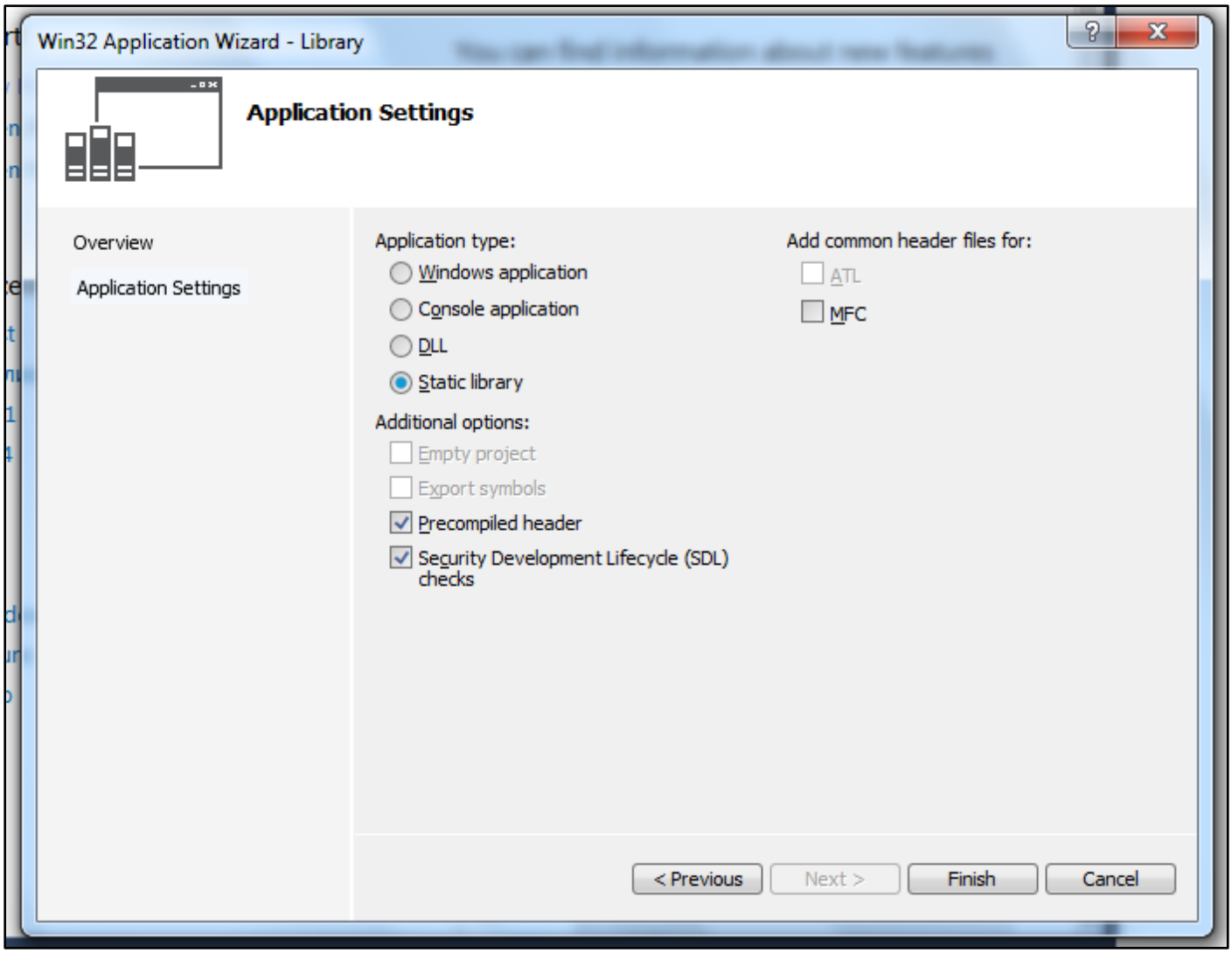

Иллюстрация 3

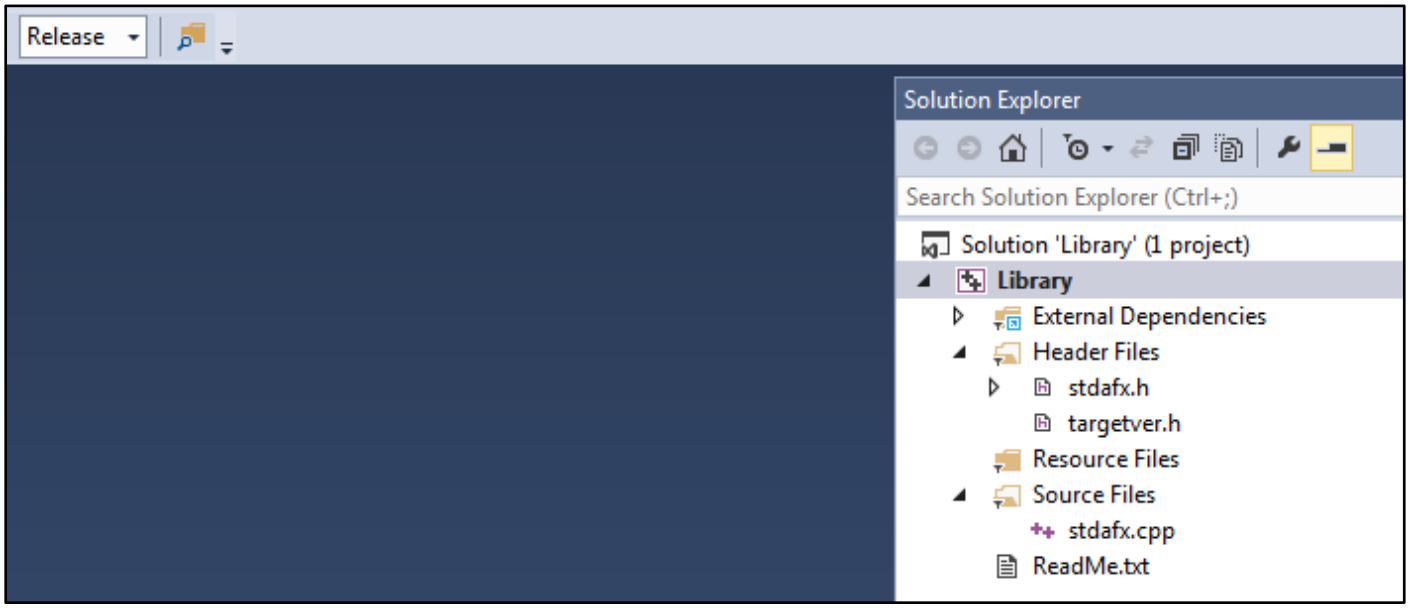

Иллюстрация 4

3. В созданном проекте в окне Solution Explorer на строке с именем проекта (в нашем случае – Library) щелкнуть правой кнопкой мыши → Add → Class; в открывшемся окне (илл. 5) нажать на кнопку Add.

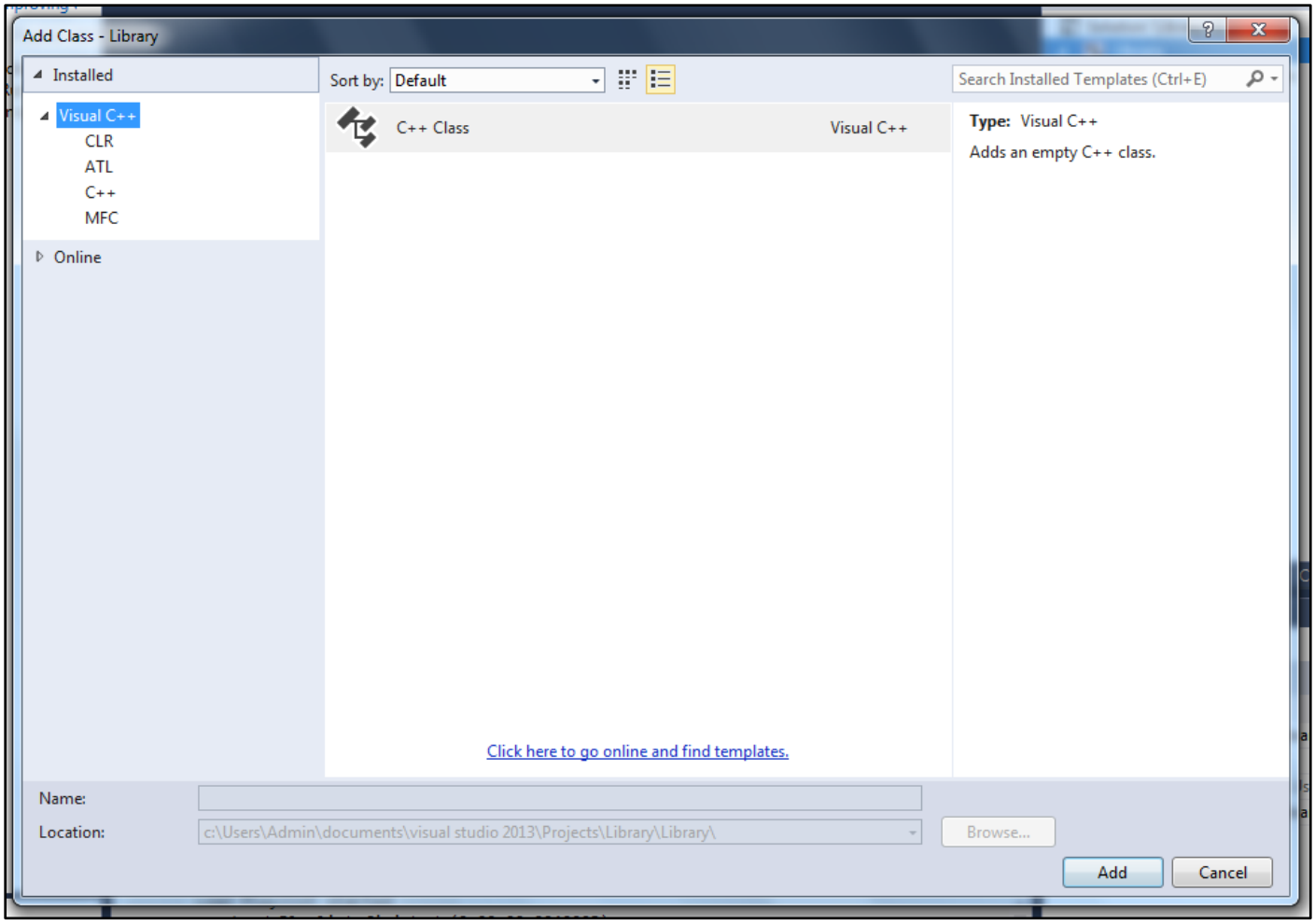

Иллюстрация 5

В новом окне (илл. 6) ввести имя класса и нажать Finish.

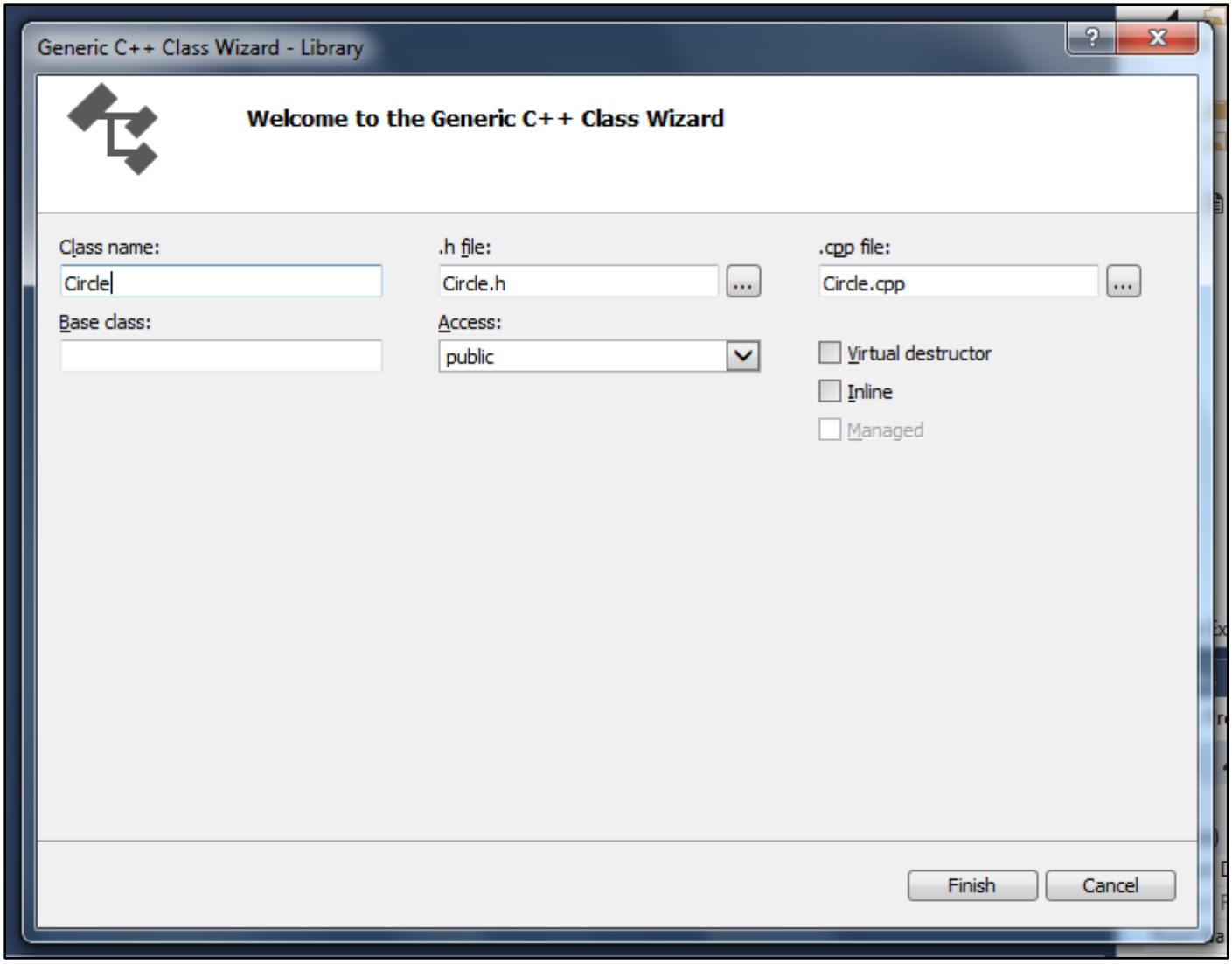

Иллюстрация 6

В проекте будут созданы два файла – в нашем случае, `Circle.cpp` и `Circle.h`, в которых будут находиться некоторые заготовки для класса.

Если файлы с определением класса и реализацией класса уже есть, вместо создания новых файлов можно добавить в проект имеющиеся. Для этого нужно скопировать соответствующие файлы в папку

`...\Library\Library`, затем в окне Solution Explorer выполнить следующие действия:

– щелкнуть правой кнопкой мыши на строке Header Files → Add → Existing Item…, выбрать нужный *файл*.h и нажать Add;

– щелкнуть правой кнопкой мыши на строке Source Files → Add → Existing Item…, выбрать нужный *файл*.cpp и нажать Add.

При этом в файле `.cpp` не должна находиться функция `main()`!

4. Настройка характеристик проекта.

Правой кнопкой мыши на проекте `Library` – выбрать Properties (свойства); в открывшемся окне в левой части раскрыть C/C++, далее найти и выбрать Code generation (генерация кода); в правой части выбрать Runtime Library (библиотека времени исполнения), справа в этой строке нажать на кнопку с черным треугольником и в открывшемся окне выбрать первую строку – Multi-threaded (/MT) для режима Release (илл. 7) или Multi-threaded Debug (/MTd) для режима Debug (илл. 8); нажать Применить и ОК.

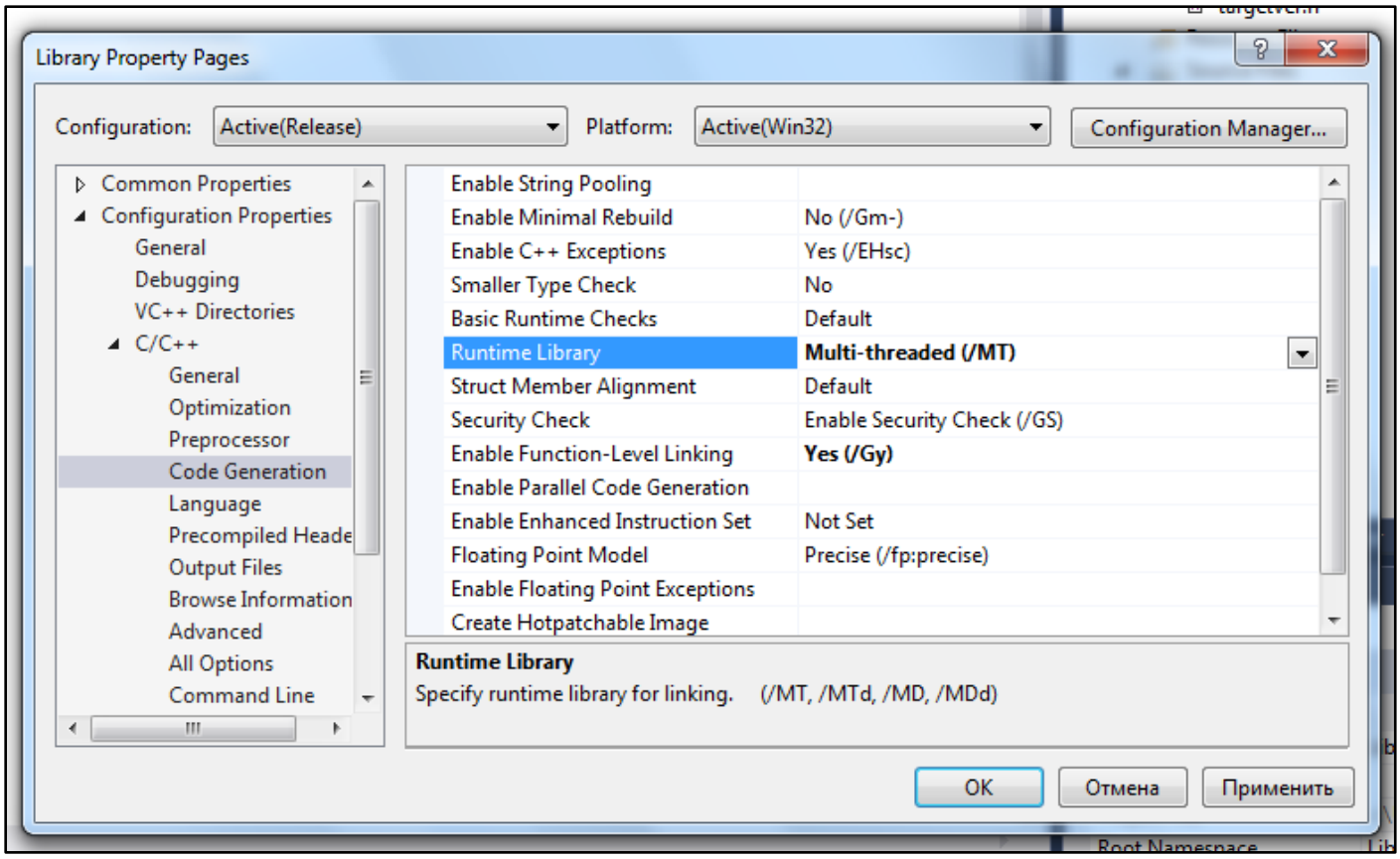

Иллюстрация 7

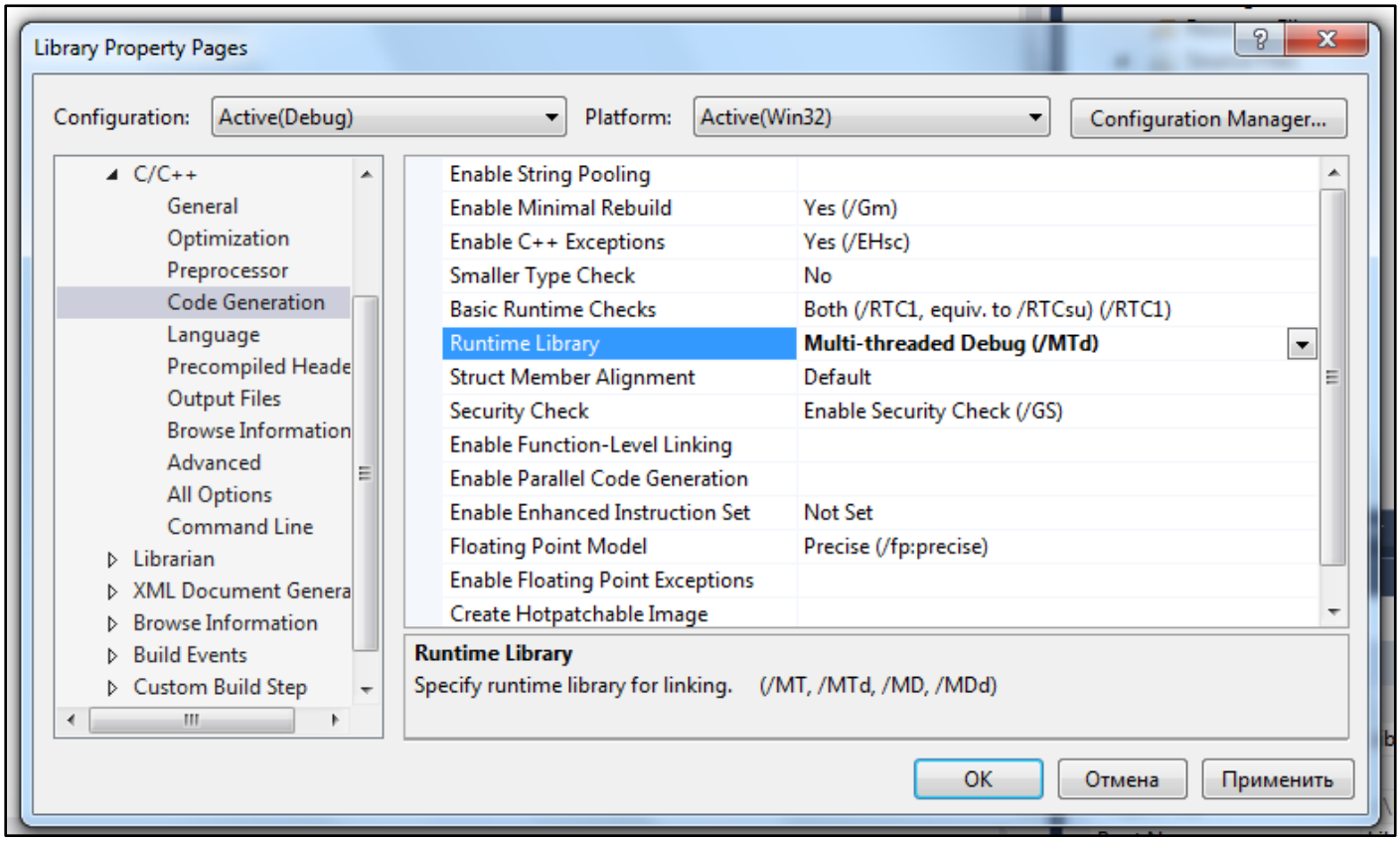

Иллюстрация 8

Характеристики проекта должны настраиваться для двух режимов – Debug и Release, если предполагается использовать оба режима.

!!! Указанное действие должно быть выполнено для всех проектов !!!

5. Чтобы определить, где будет размещаться библиотека, строим проект (BUILD); внизу в окне output будет указано маршрутное имя библиотеки. Указанные действия выполняем в режимах Release и Debug; получим что-то вроде

```
...\Projects\Library\Debug\Library.lib
...\Projects\Library\Release\Library.lib
```

Создание консольного приложения для прикладной программы

1. В окне Solution Explorer на строке Solution 'Library' нажать правой кнопкой мыши, в открывшемся окне выбрать Add → New Project, далее создать обычное консольное приложение C++ (в нашем примере – с именем `Application`).

Для созданного приложения также установить в свойствах проекта C/C++ → Code Generation, Runtime Library → Multi-threaded (/MT) для режима Release и Multi-threaded Debug (/MTd) для режима Debug.

2. В открывшемся файле `Application.cpp` для прикладной программы ввести маршрутное имя файла, содержащего определение класса; в нашем примере, это файл `Circle.h`, находящийся в проекте `Library`; поэтому надо ввести следующее предложение (илл. 9):

```
#include "..\Library\Circle.h"
```

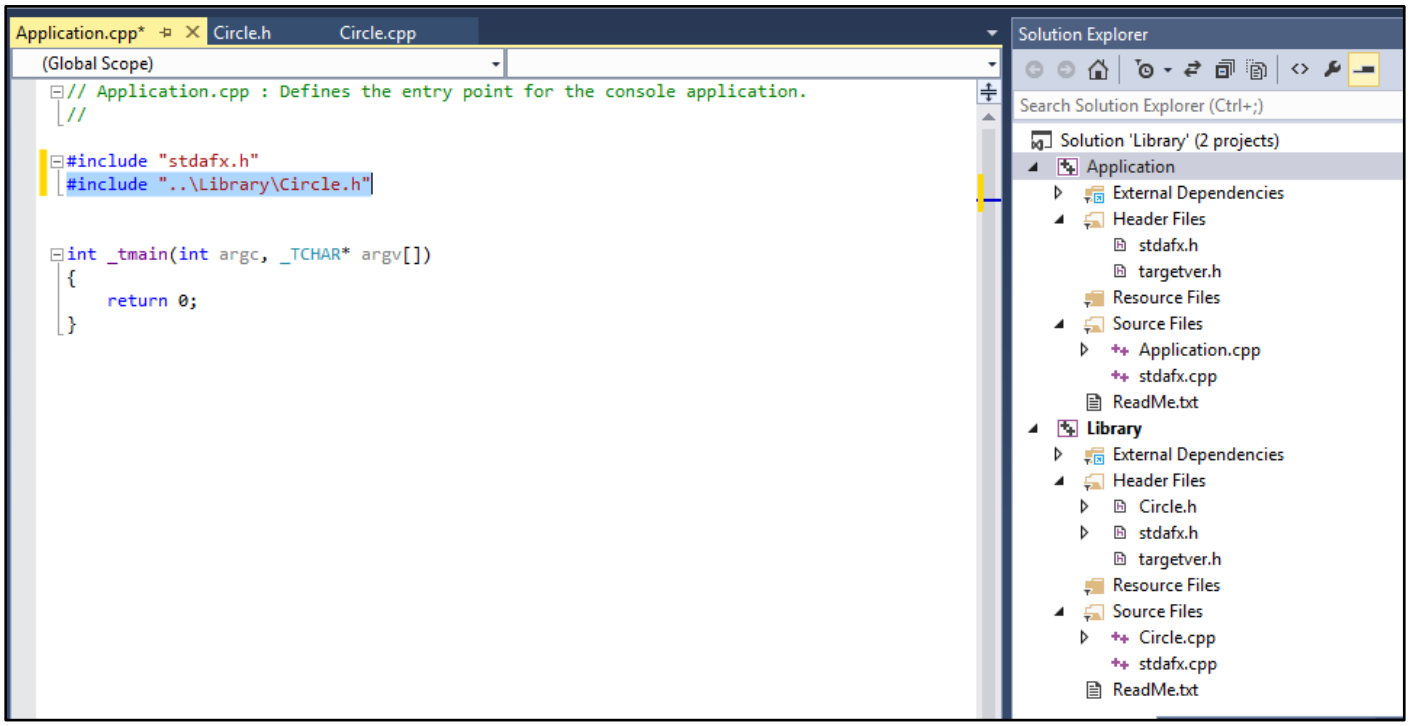

Иллюстрация 9

3. Устанавливаем зависимости между проектами (т.е. порядок компиляции): сначала должен компилироваться проект, содержащий статическую библиотеку (у нас это `Library`), а затем – проект, содержащий прикладную программу (`Application`).

Для этого в окне Solution Explorer на строке Solution 'Library' → правая кнопка мыши → в открывшемся окне выбираем Project dependencies; в открывшемся окне на вкладке Dependencies в окне Projects выбираем имя проекта для прикладной задачи (в нашем примере, `Application`), а в окне Depends on ставим галочку у проекта с библиотекой (в нашем примере, у `Library`) (илл. 10).

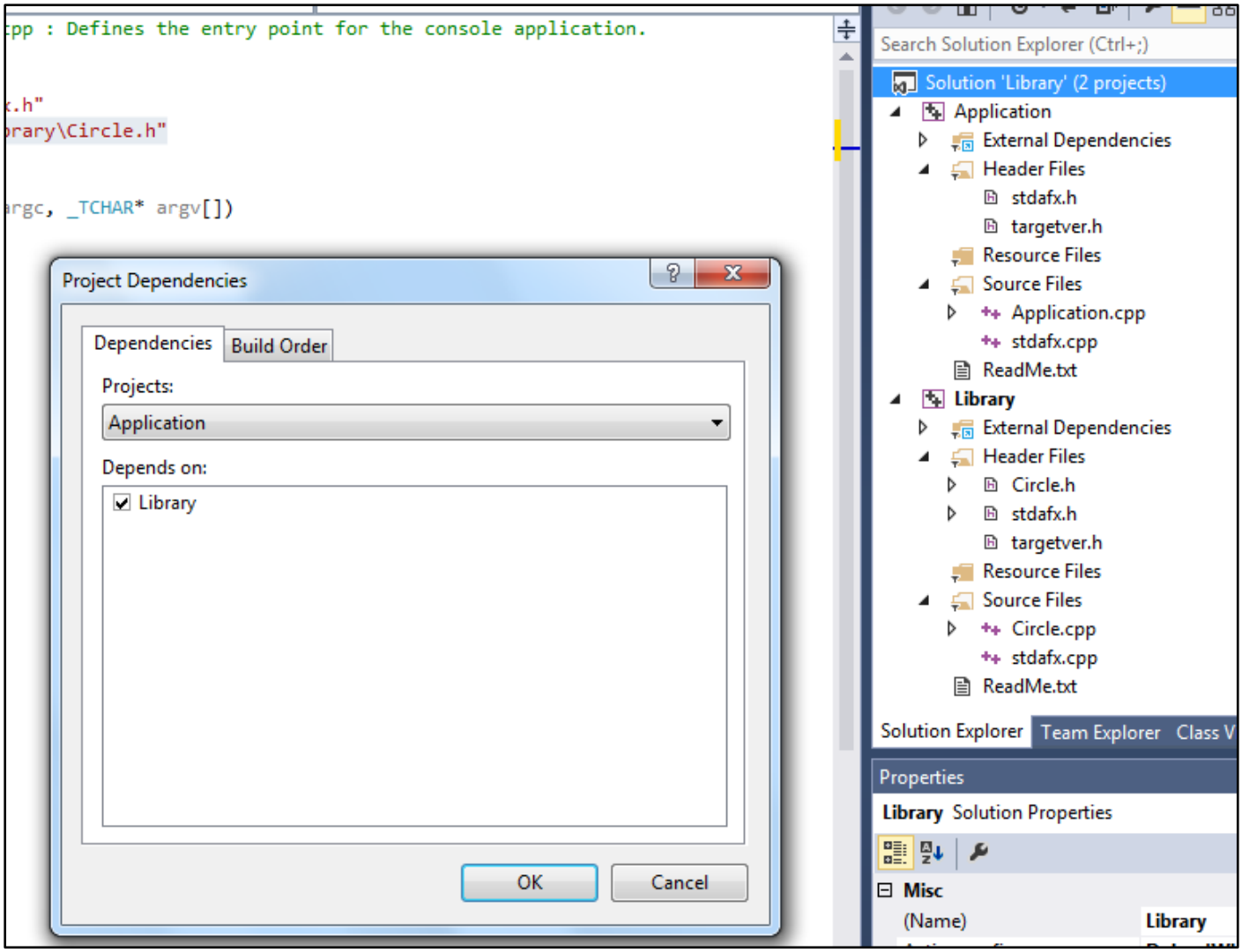

Иллюстрация 10

4. В проекте `Application` указываем место размещения библиотеки с разработанным классом (В окне Solution Explorer на имени проекта с прикладной программой (`Application`) выбираем Properties (свойства); слева выбираем Linker → General; в правом окне выбираем Additional Library Directories (илл. 11);

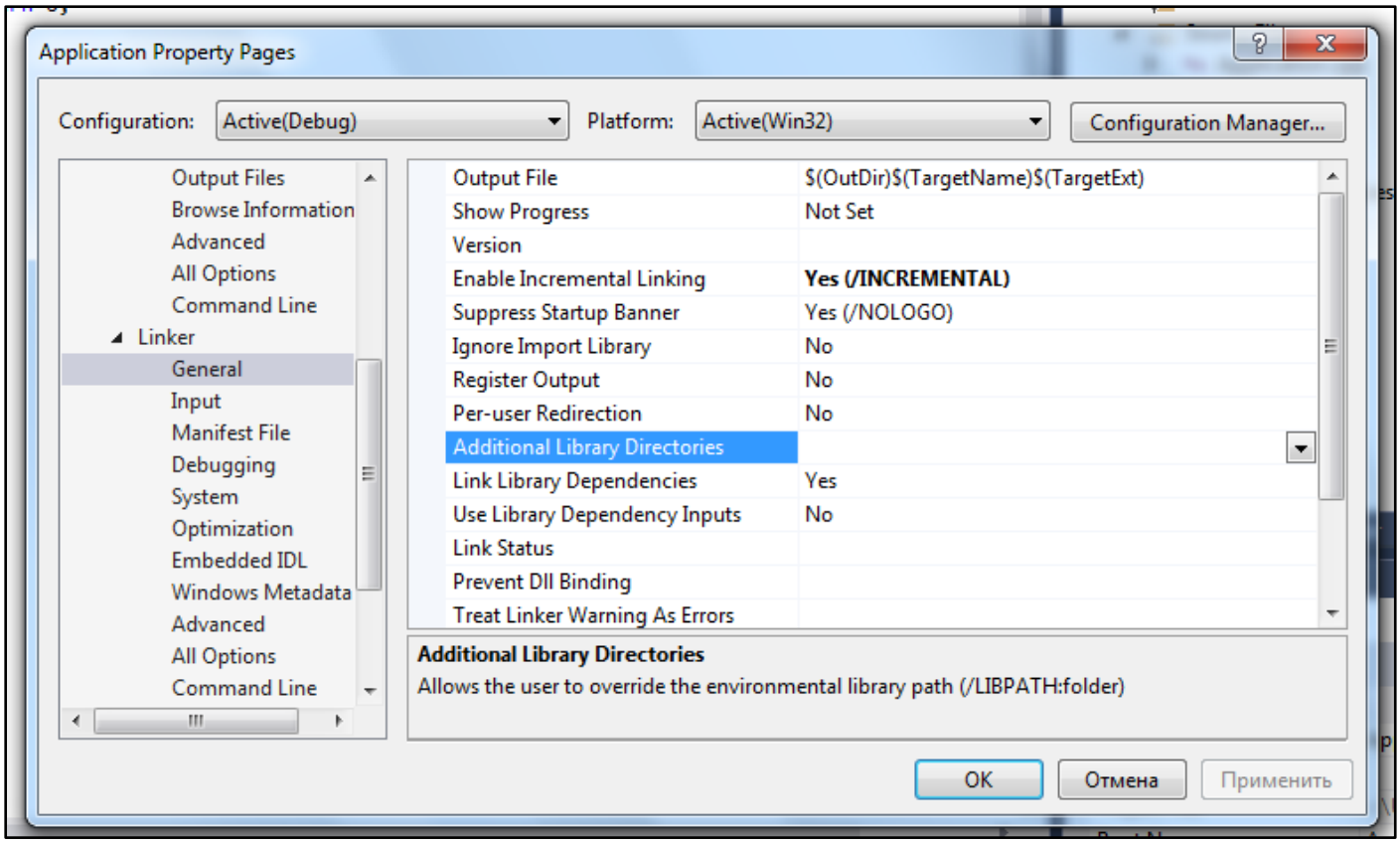

Иллюстрация 11

Нажимаем справа на кнопку с черным треугольником, в открывшемся окне нажимаем <Edit…>; в верхней части открывшегося окна нажимаем на иконку с изображением папки (илл. 12)

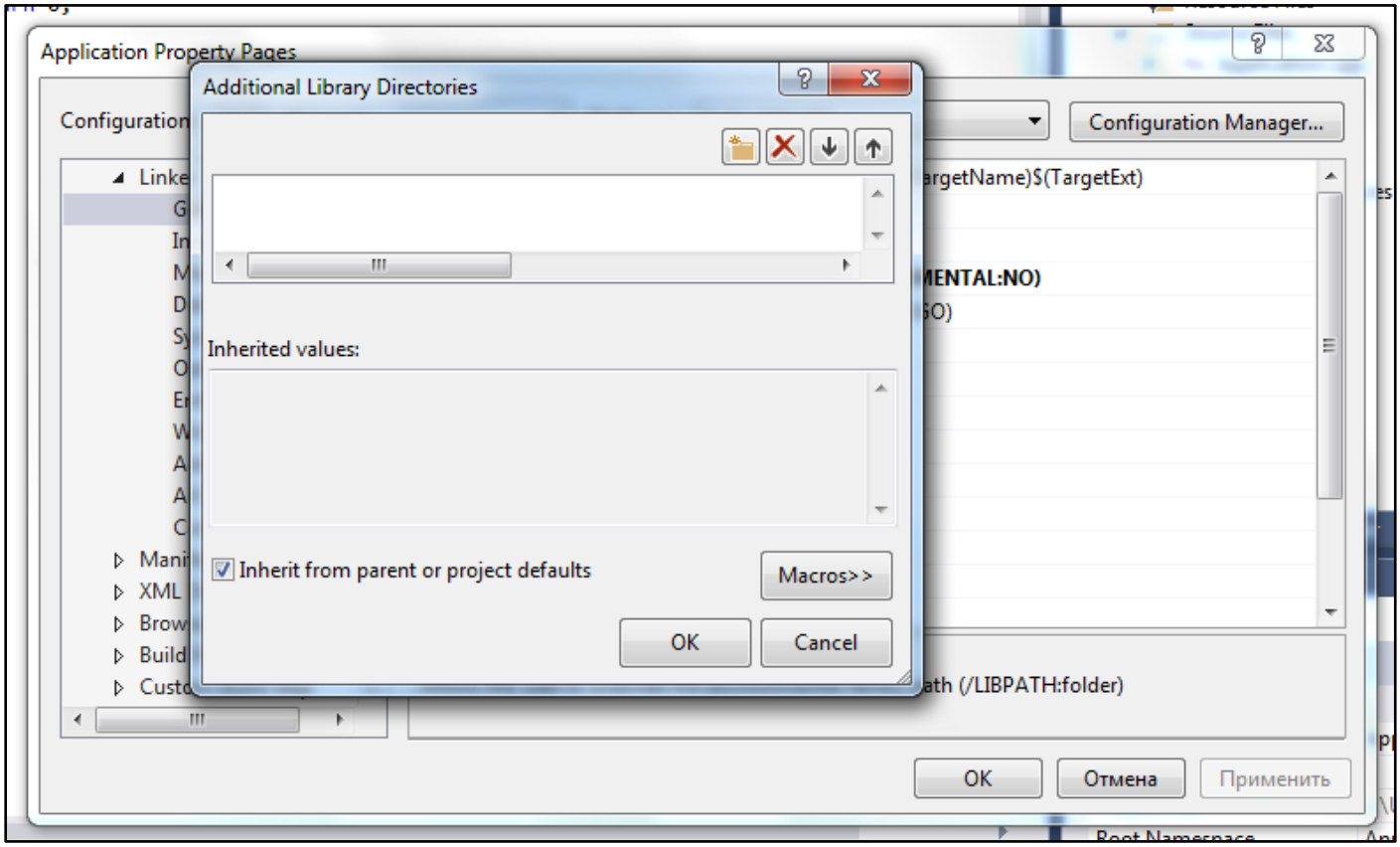

Иллюстрация 12

– откроется окно для ввода, справа – кнопка с многоточием; нажимаем на нее и выбираем папку, где располагается библиотека: для режима Release в нашем примере – это `…\Projects\Library\Release`, для режима Debug – `…\Projects\Library\Debug` (илл. 13):

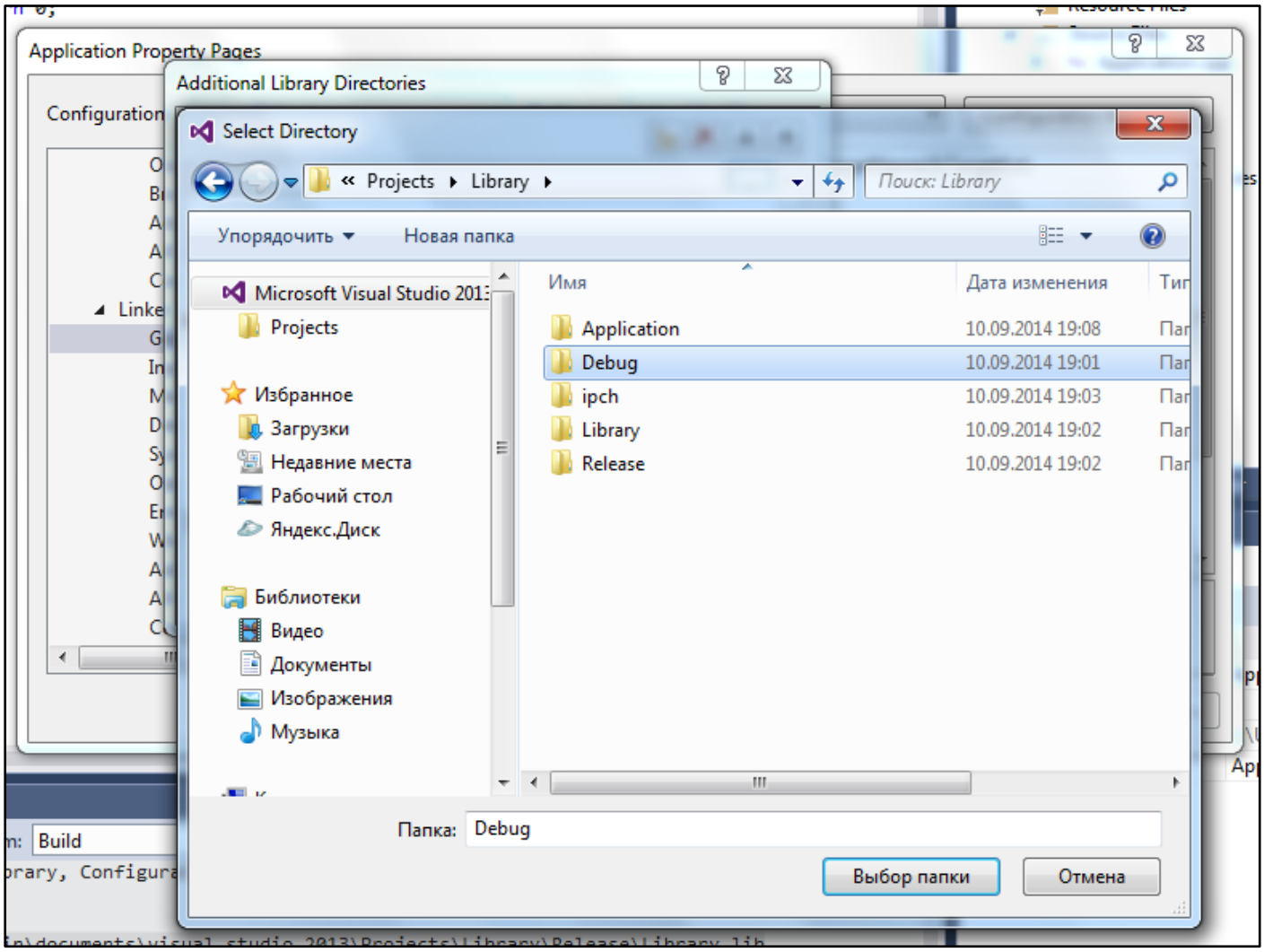

Иллюстрация 13

далее нажимаем Выбор папки, ОК – должны остаться в окне Properties.

5. В окне Properties выбираем Linker → Input → Additional Dependencies (илл. 14)

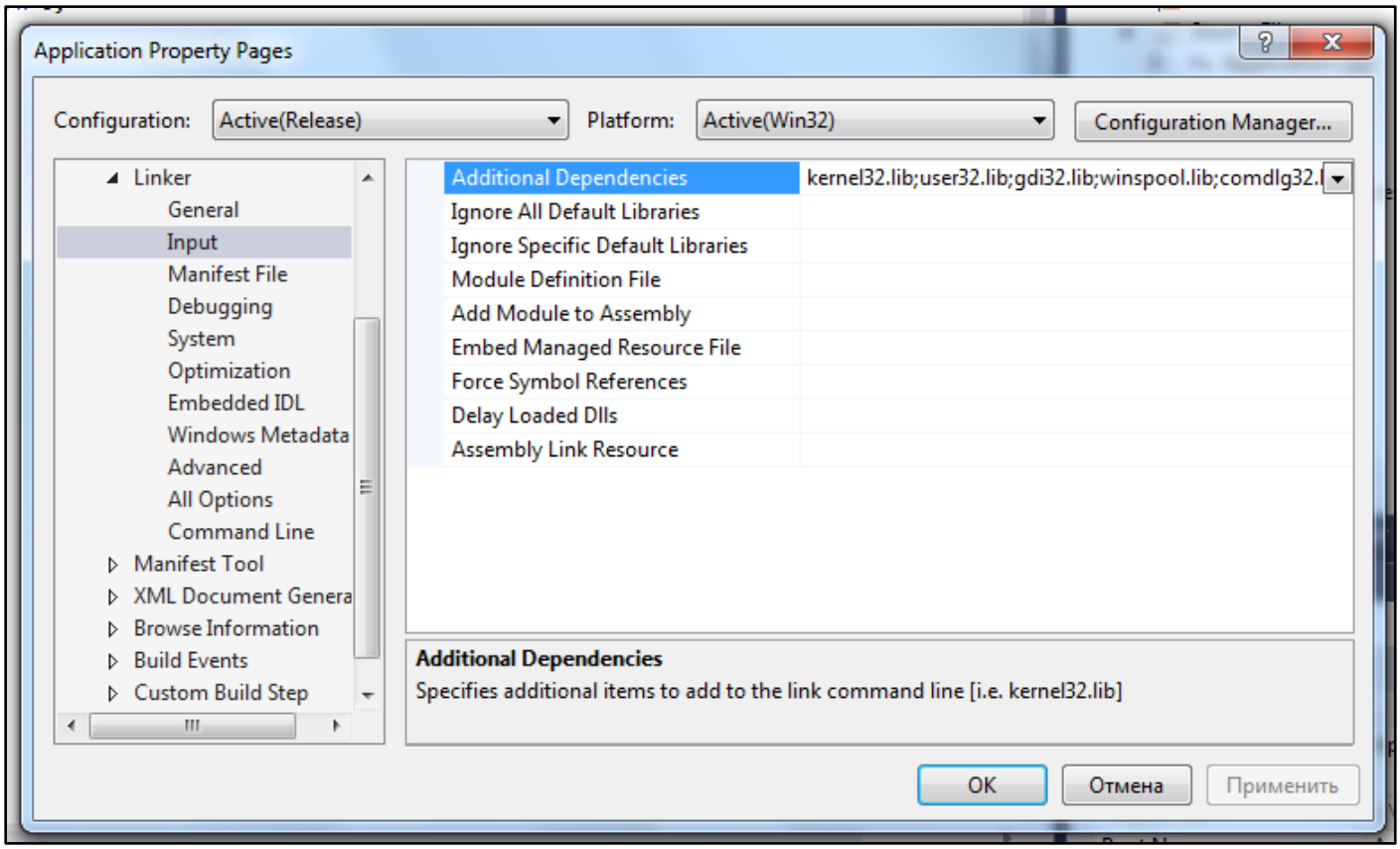

Иллюстрация 14

Нажимаем справа черный треугольник, щелкаем по окну редактирования – открывается дополнительное окно, в котором вводим имя библиотеки (илл. 15).

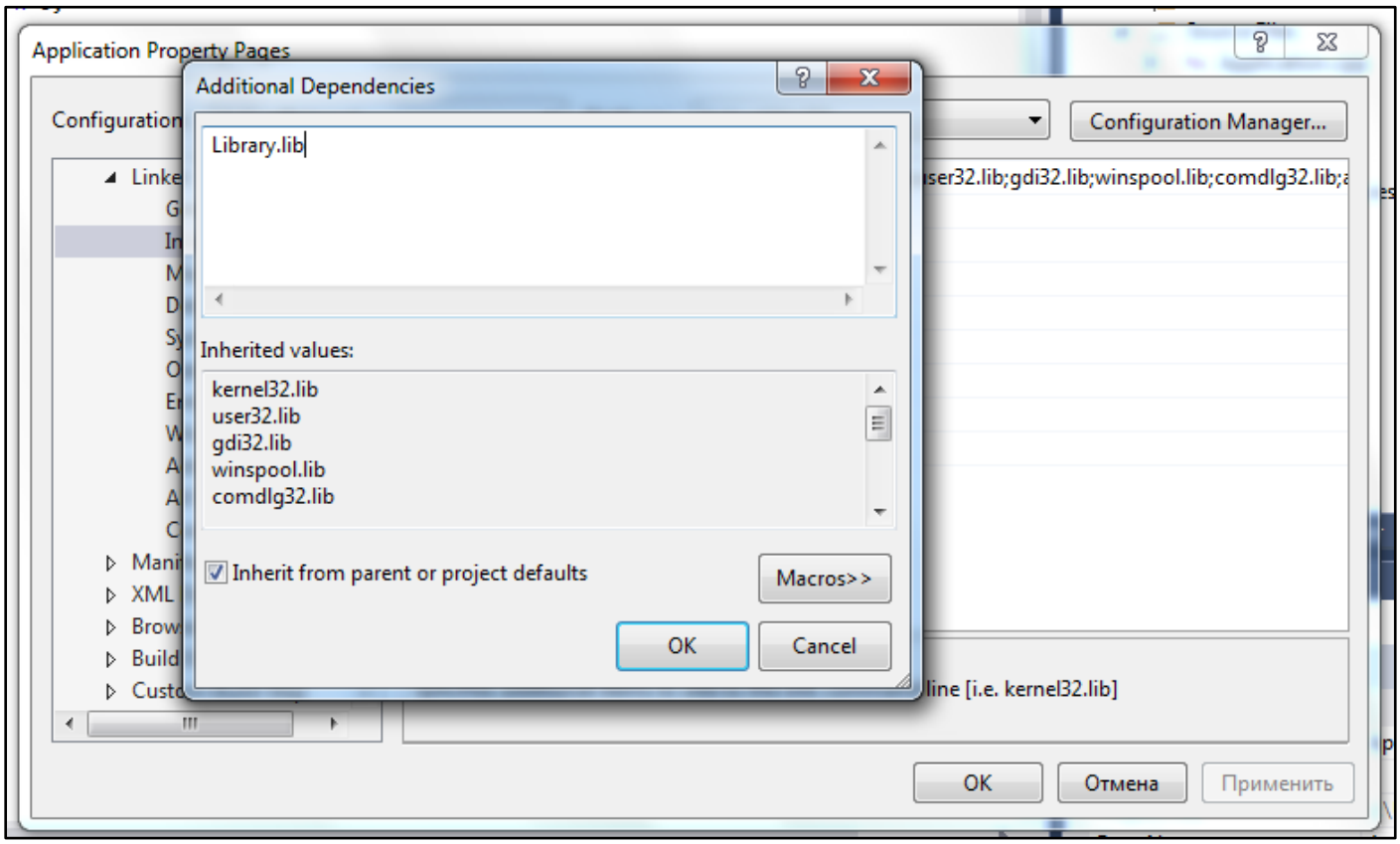

Иллюстрация 15

Далее закрываем все окна (ОК → ОК).

Все необходимые свойства проекта настроены.

6. Далее необходимо указать, какой проект запускается при нажатии клавиш Ctrl + F5 (или просто F5).

Для этого открываем окно properties для соответствующего проекта (в нашем случае, это `Application`), и в нем выбираем Set as StartUp Project.

Теперь можно выполнять программу.

Создание консольного приложения для тестирования

Консольное приложение для тестирования создается так же, как и консольное приложение для прикладной программы (см. п.п. 1 – 5).

Дополнительные настройки для данного проекта.

1. Размещение дополнительных include-файлов.

В окне Properties (Свойства) нового проекта (Testing) выбрать C/C++ → General, Additional Include Directories (илл. 16):

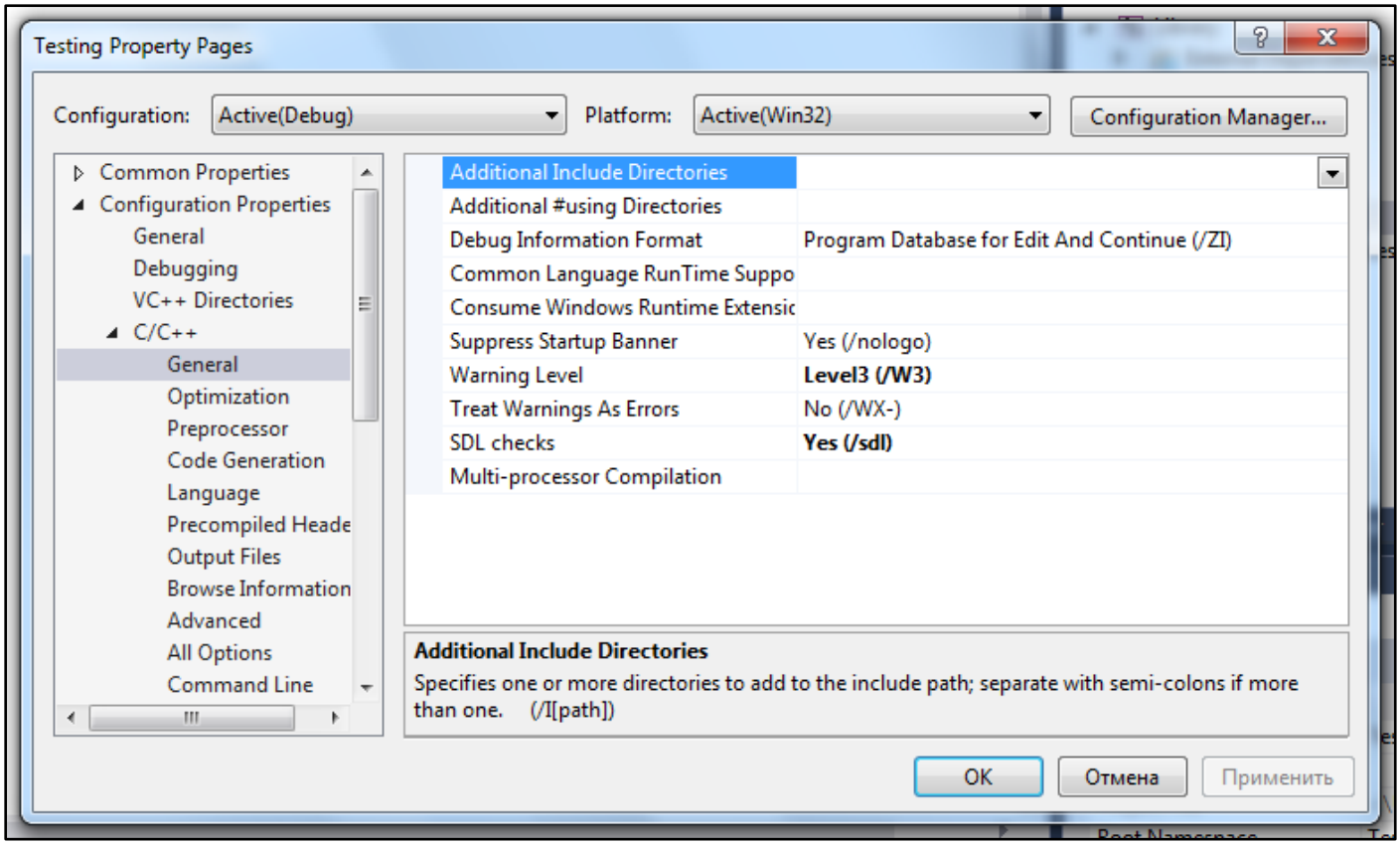

Иллюстрация 16

Нажать на черный треугольник справа, выбрать редактирование текстового поля, и в открывшемся окне ввести (выбором) путь к папке include проекта gtest-1.7.0, размещенного в папке Projects (илл. 17).

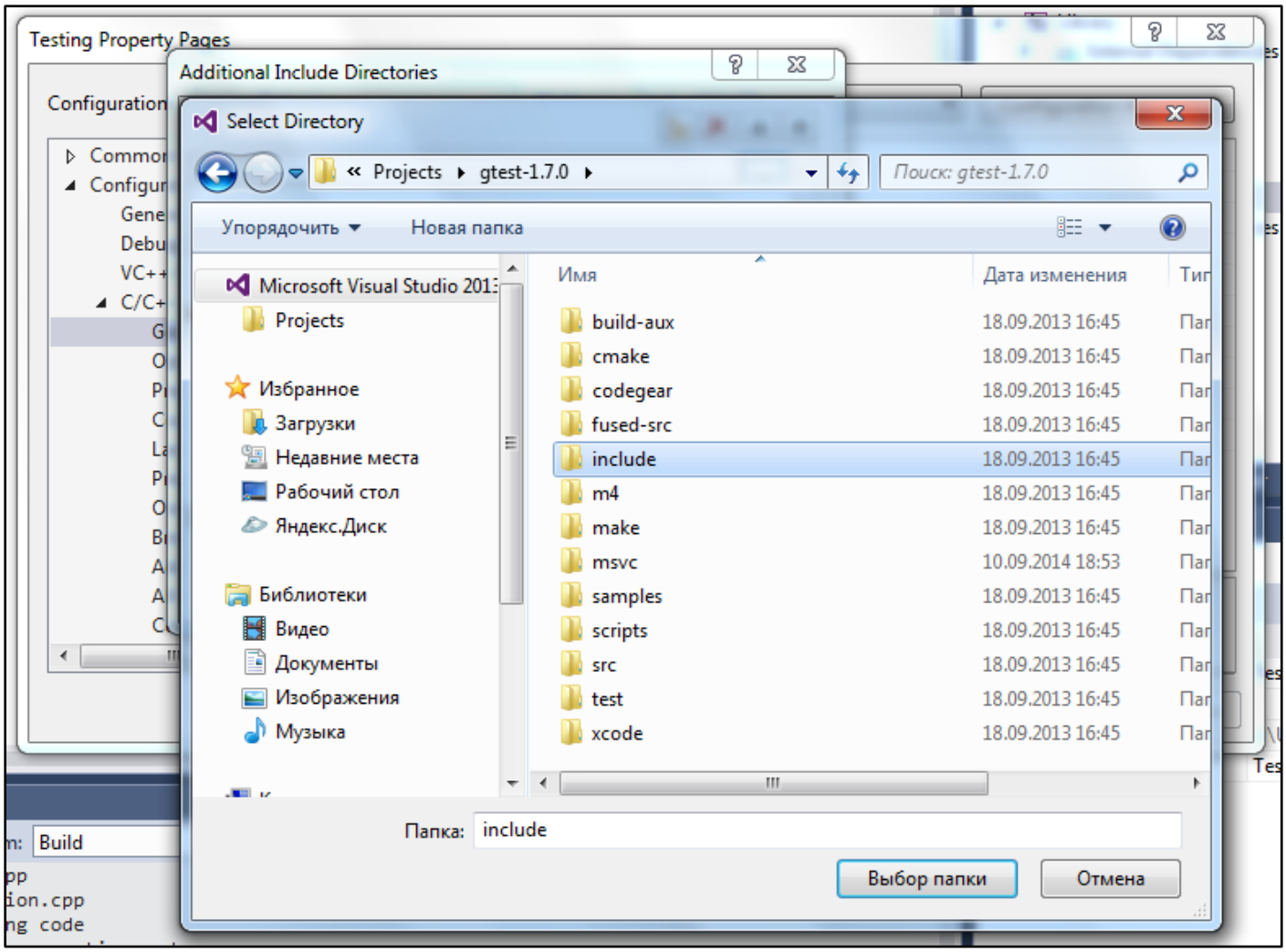

Иллюстрация 17

Далее – Выбор папки и OK; остаемся в окне Properties.

2. Расположение библиотеки тестирования

Выбрать Linker → General, Additional Library Directories и выбираем папку из проекта gtest-1.7.0, в которой размещена библиотека (илл. 18):

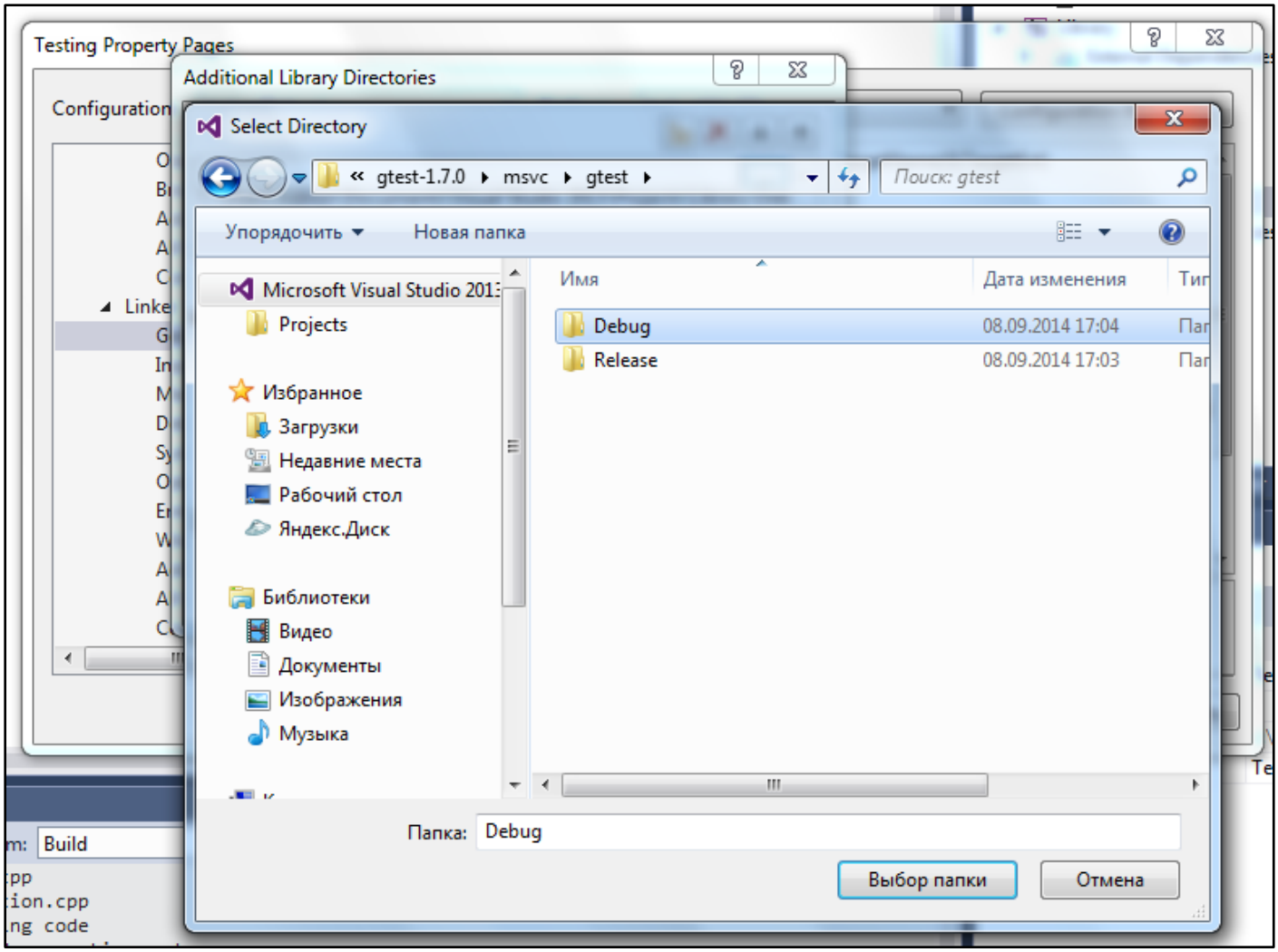

Иллюстрация 18

Далее опять выбор папки и OK. Остаемся в окне Properties.

Наконец, выбираем Linker → Input, Additional Dependencies и вводим имя библиотеки: gtest.lib для режима Release и gtestd.lib для режима Debug (илл. 19).

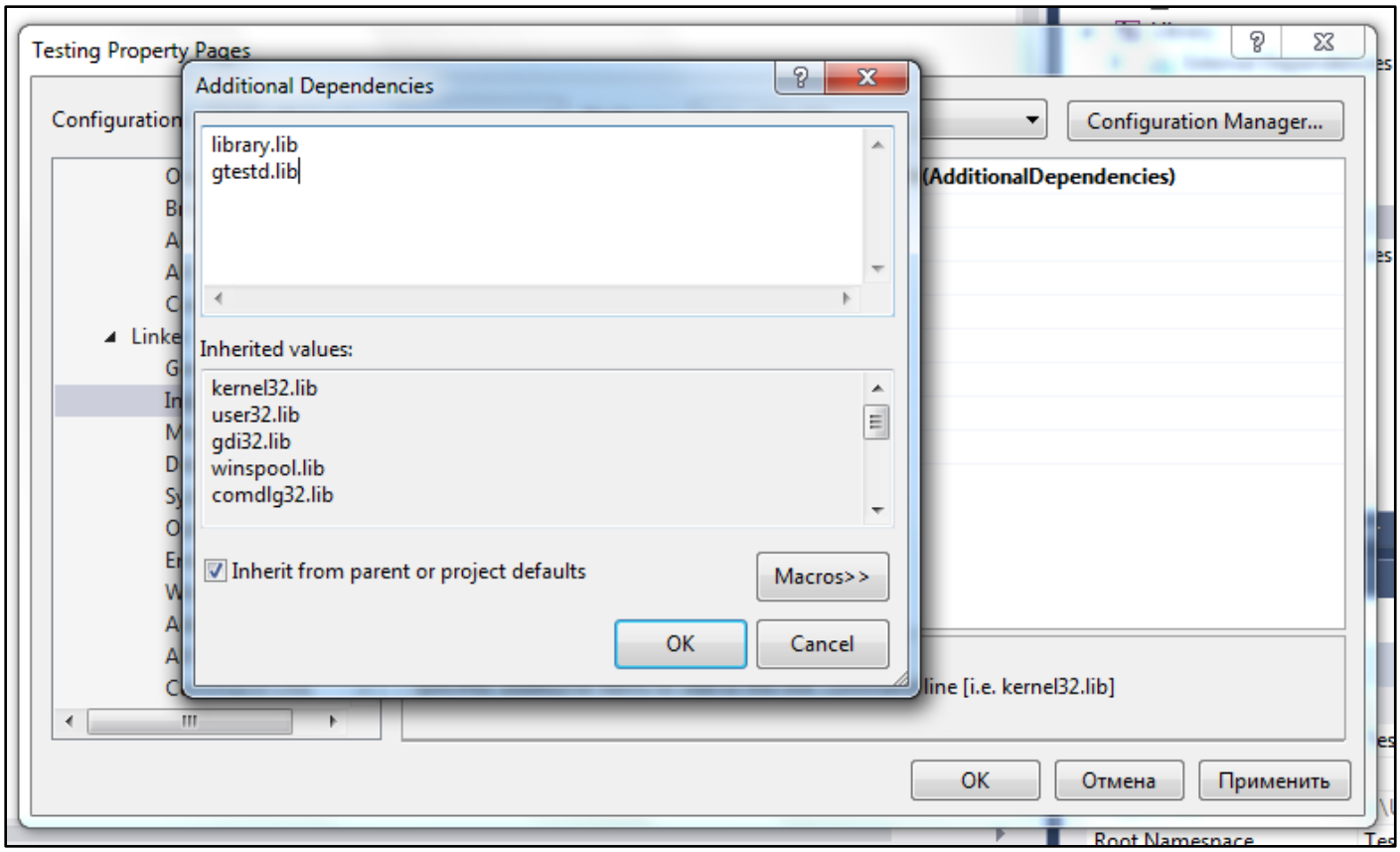

Иллюстрация 19

Далее – OK, OK.

Устанавливаем для нового проекта Project Dependencies (илл. 20):

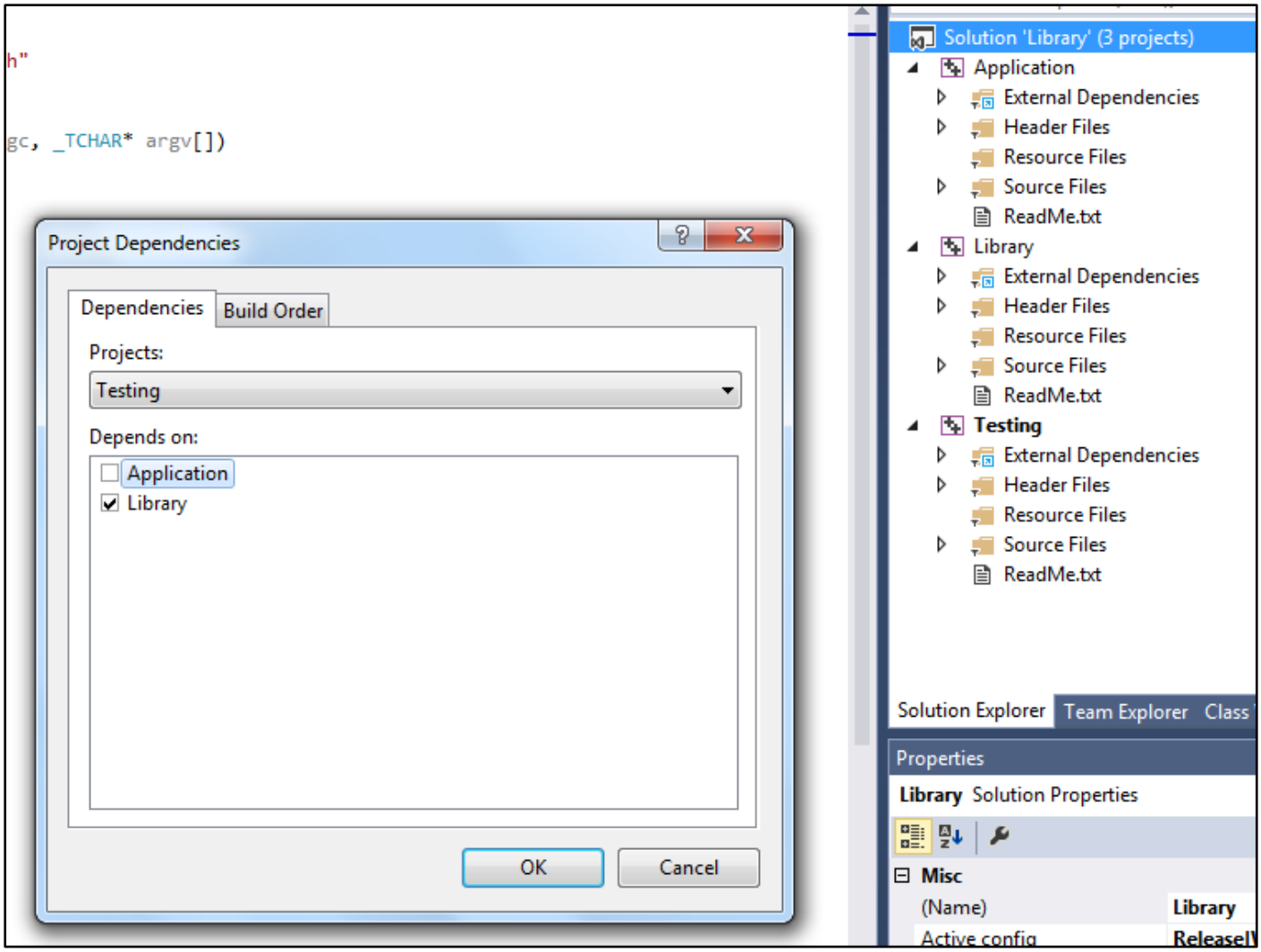

Иллюстрация 20

Все проекты настроены.

Для проекта Testing следует выбрать Set as StartUp Project.

3. Добавляем нужные строки в cpp-файл для тестирующей программы (илл. 21):

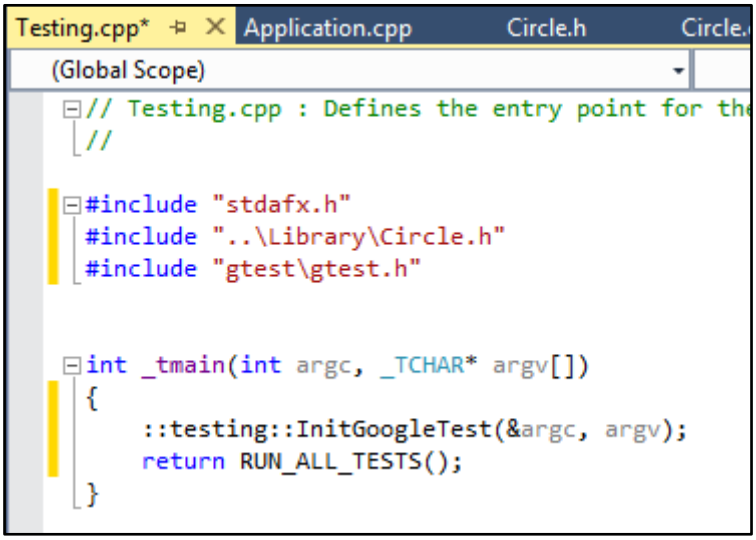

Иллюстрация 21

Теперь настроенный проект можно запускать на исполнение. Поскольку пока в файлах ничего нет, получим следующее (илл. 22):

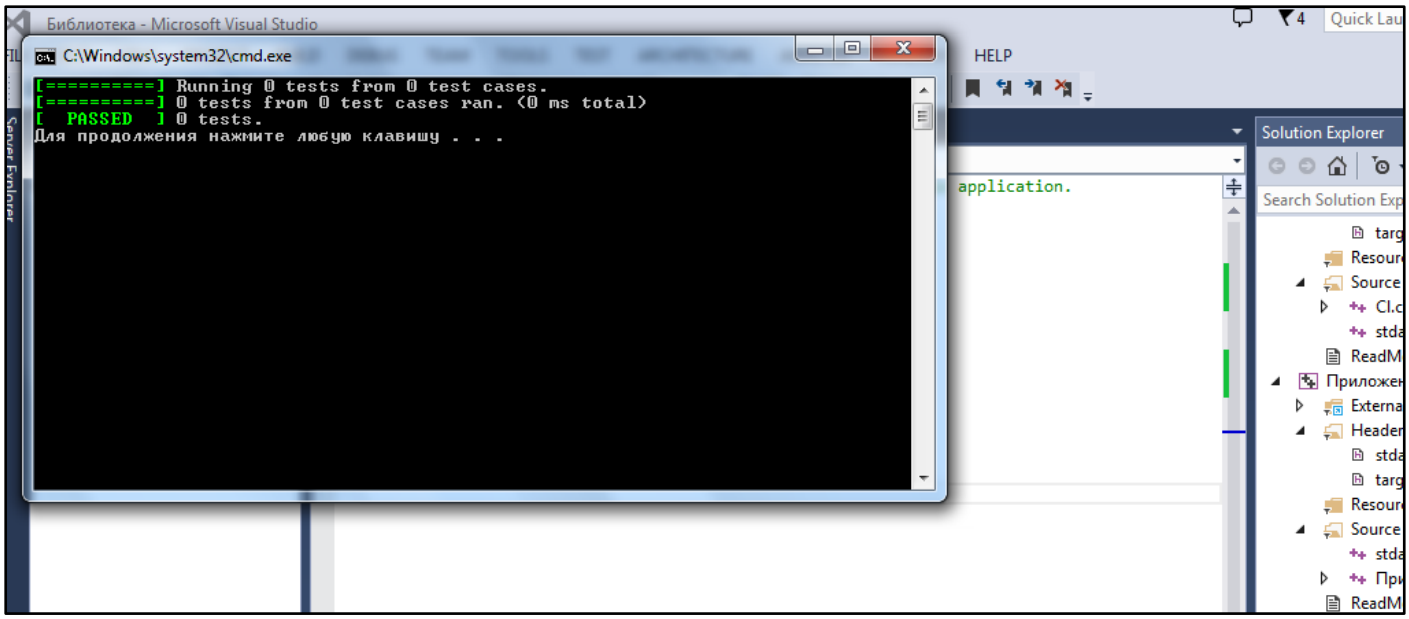

Иллюстрация 22

Дополнительную начальную информацию о задании тестов можно найти во Введении в Google C++ Testing Framework, по ссылке:

https://code.google.com/p/googletest-translations/wiki/GoogleTestPrimerRussian.