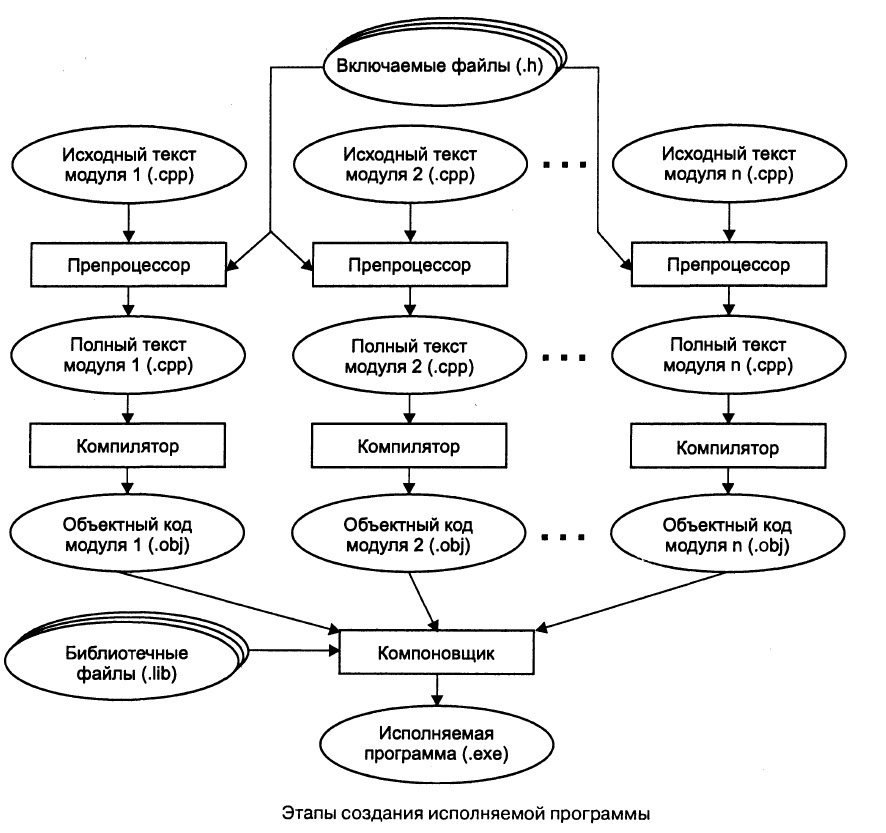
## Литература

1. Г. Шилдт. Самоучитель С++. 3-е изд. – СПб. «БХВ», 2008.
2. Г. Шилдт. С++ Базовый курс. 3-е изд. – М.: Вильямс, 2006.
3. Х. М. Дейтл, П. Дж. Дейтл. Как программировать на С++. М.: Бином 2007.
4. Эккель Б. Философия С++. Введение в стандартный С++. 2-е изд. – Спб.: Питер, 2004.
5. Эккель Б., Эллисон Ч. Философия С++. Практическое программирование. – СПб.: Питер, 2004.
6. С. Прата. Язык программирования С++. Лекции и упражнения. 6-е изд. М.: Вильямс, 2012.

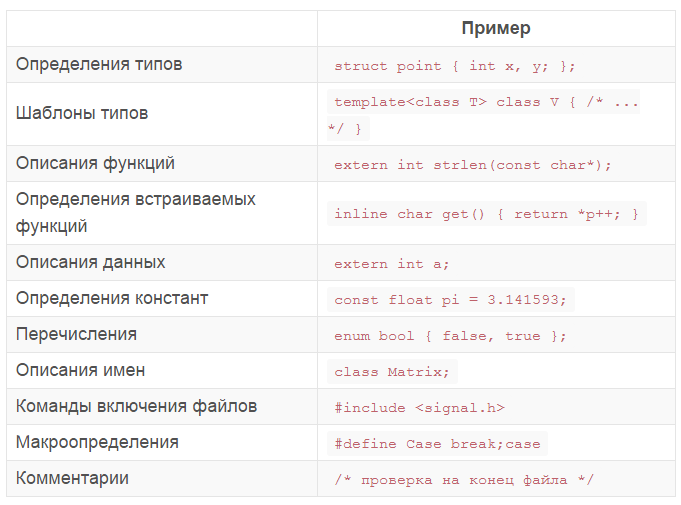
## Раздельная компиляция

**Крупные программы обычно состоят из нескольких файлов исходного кода, которые могут совместно использовать определенные данные. В таких программах применяется раздельная компиляция файлов.**

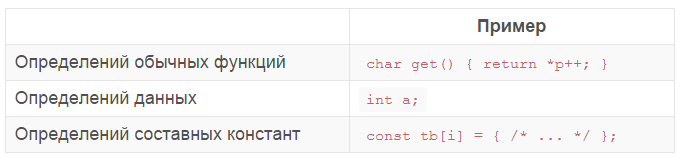
**Проект, состоящий из нескольких файлов**

****

# Содержимое заголовочных файлов

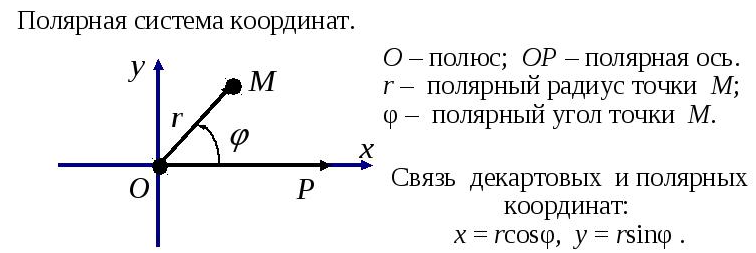


# В заголовочном файле никогда не должно быть



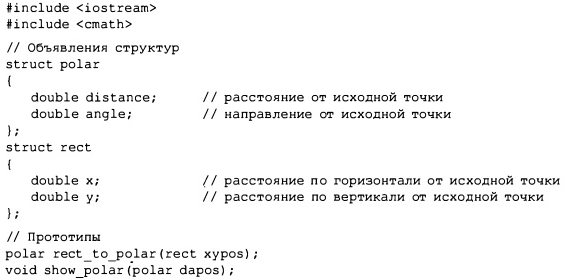
Язык C++, как и С, позволяет и даже поощряет размещение функций программы в отдельных файлах. Файлы можно компилировать раздельно, а затем связывать их с конечным продуктом — исполняемой программой. Как правило, компилятор C++ не только компилирует программы, но и управляет работой компоновщика. При изменении только одного файла можно перекомпилировать лишь этот файл и затем связать его с ранее скомпилированными версиями других файлов. Этот механизм облегчает работу с крупными программами. Большинство интегрированных сред разработки (integrated development environment — IDE), например, Microsoft Visual Studio и др. (Embarcadero C++ Builder, Apple Xcode, Freescale CodeWarrior), предоставляют аналогичные средства, доступ к которым осуществляется с помощью меню Project (Проект).

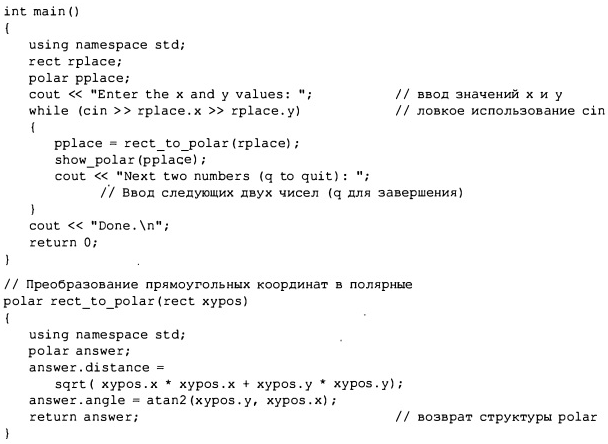
**Рассмотрим пример 1. В нем представлена программа, которая преобразует прямоугольные координаты в полярные. В частности, определены две структуры, представляющие два разных способа описания координат на плоскости, и описаны функции для преобразования одной формы в другую и отображения результата.**

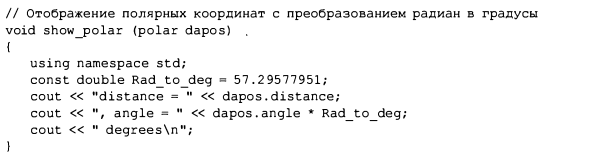




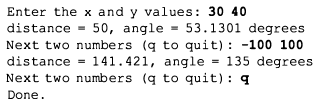
Пример 1.







Результат выполнения программы



Предположим, что принято решение разделить программу из примера 1 на части и поместить используемые ею функции в отдельный файл. Нельзя просто вырезать из исходного файла часть кода после окончания функции main (). Дело в том, что main () и другие две функции используют одни и те же объявления структур, поэтому необходимо поместить эти объявления в оба файла. При простом наборе объявлений в коде можно допустить ошибку. Но даже если объявления скопированы безошибочно, при последующих модификациях нужно будет не забыть внести изменения в оба файла.

**Для решения подобных проблем была предоставлена директива #include. Вместо того чтобы помещать объявления структур в каждый файл, их можно разместить в заголовочном файле, а затем включать его в каждый файл исходного кода.** Таким образом, изменения в объявление структуры будут вноситься только один раз в заголовочный файл. Кроме того, в заголовочный файл можно помещать прототипы функций.

Итак, исходную программу можно разбить на три части:

**1) заголовочный файл**, содержащий объявления структур и прототипы функций, которые используют эти структуры;

**2) файл исходного кода, содержащий определение функций, которые работают со структурами;**

**3) файл исходного кода, содержащий главную функцию main(), который вызывает функции работы со структурами.**

Такая стратегия может успешно применяться для организации программы. Если, например, создается другая программа, которая пользуется теми же самыми функциями, достаточно включить в нее заголовочный файл и добавить файл с функциями в проект. К тому же такая организация программы соответствует принципам объектно-ориентированного программирования (ООП). Первый файл — заголовочный — содержит определения пользовательских типов. Второй файл содержит код функций для манипулирования типами, определенными пользователем. Вместе оба файла формируют пакет, который можно использовать в различных программах.

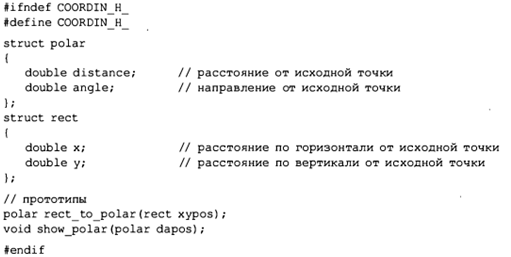
В заголовочный файл не следует помещать определения функций или объявления переменных. Хотя в простейших проектах такой подход может работать, обычно он приводит к проблемам. Например, если в заголовочном файле содержится определение функции, и этот заголовочный файл включен в два других файла, которые являются частью одной программы, в этой программе окажется два определения одной и той же функции, что вызовет ошибку, если только функция не является встроенной.

Объявления структур, классов можно помещать в заголовочные файлы, поскольку они не создают переменные, а только указывают компилятору, как создавать структурную переменную, когда она объявляется в файле исходного кода. Подобно этому объявления шаблонов — это не код, который нужно компилировать, а инструкции для компилятора, указывающие, каким образом генерировать определения функций, чтобы они соответствовали вызовам функций, встречающимся в исходном коде. Данные, объявленные как const, и встроенные функции имеют специальные свойства связывания, которые позволяют размещать их в заголовочных файлах, не вызывая при этом каких-либо проблем.

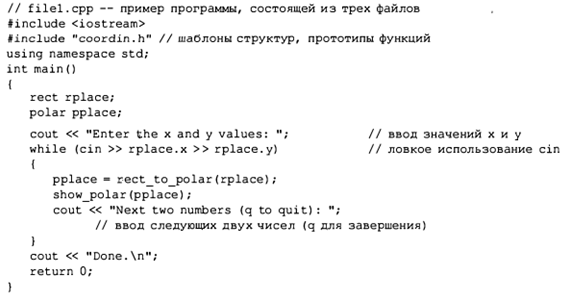
**В примере 2.1(coordin.h), примере 2.2(file1.cpp), примере 2.3(file2.cpp) показан результат разделения программы из примера 1 на отдельные части.**

Обратите внимание, что при включении заголовочного файла используется запись "coordin.h", а не <coordin.h>. Если имя файла помещено в угловые скобки, компилятор C++ ищет его в той части базовой файловой системы, где расположены стандартные заголовочные файлы. Но когда имя файла представлено в двойных кавычках, компилятор сначала ищет файл в текущем рабочем каталоге или в каталоге с исходным кодом (либо в другом аналогичном месте, которое зависит от версии компилятора). Не обнаружив заголовочный файл там, он ищет его в стандартном местоположении. Таким образом, при включении собственных заголовочных файлов должны использоваться двойные кавычки, а не угловые скобки.

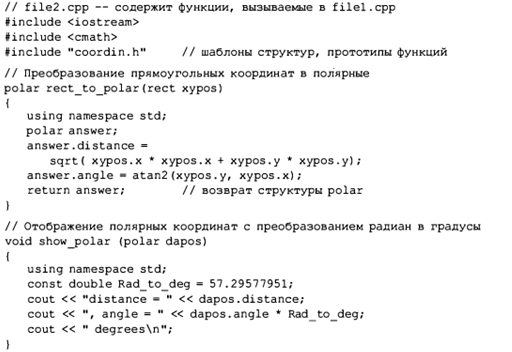
Пример 2.1(coordin.h)



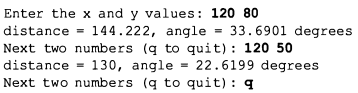
Пример 2.2(file1.cpp)



Пример 2.3(file2.cpp)



Результат работы программы:



## Управление заголовочными файлами

**Заголовочный файл должен включаться в файл только один раз.**

Это кажется простым требованием, которое легко запомнить и придерживаться, тем не менее, можно непреднамеренно включить заголовочный файл несколько раз, даже не подозревая об этом. Например, предположим, что используется заголовочный файл, который включает другой заголовочный файл.

**В C/C++ существует стандартный прием, позволяющий избежать многократных включений заголовочных файлов. Он основан на использовании директивы препроцессора #ifndef (if not defined — если не определено).**

Показанный ниже фрагмент кода обеспечивает обработку операторов, находящихся между директивами #ifndef и #endif, только в случае, если имя coordin\_h\_ не было определено ранее с помощью директивы препроцессора #defіnе:



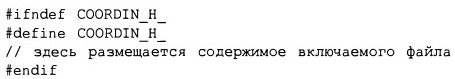
Обычно директива #define используется для создания символических констант, как в следующем примере:

#define MAXIMUM 4096

Однако для определения имени достаточно просто указать директиву #define с этим именем:

#define COORDIN\_H\_

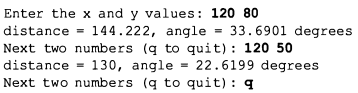
Прием, используемый в примере 2.1(coordin.h), предусматривает помещение содержимого файла внутрь #ifndef:



Когда компилятор впервые сталкивается с этим файлом, имя coordin\_h\_ должно быть неопределенным. Во избежание совпадения с существующими именами, имя строится на основе имени включаемого файла и нескольких символов подчеркивания.

В этом случае компилятор будет обрабатывать код между директивами #ifndef и #endif, что, собственно, и требуется. Во время обработки компилятор читает строку с директивой, определяющей имя coordin\_h\_. Если затем компилятор обнаруживает второе включение coordin.h в том же самом файле, он замечает, что имя coordin\_h\_ уже определено, и переходит к строке, следующей после #endif. Обратите внимание, что данный прием не предотвращает повторного включения файла. Вместо этого он заставляет компилятор игнорировать содержимое всех включений кроме первого. Такая методика защиты используется в большинстве стандартных заголовочных файлов С и C++. Если ее не применять, одна и та же структура, например, окажется объявленной в файле дважды, что приведет к ошибке компиляции.

В результате компиляции и компоновки этих двух файлов исходного кода и нового заголовочного файла получается исполняемая программа. Ниже приведен пример ее выполнения:



=======================================================