```
/**
    Curso de C++ Moderno
    Lista de exercícios Módulo 02: Compilação em detalhes

    @author Arthur Nunes de Paiva Santos Queiroz
    @version 03/2017
*/

*) Respostas dos exercícios do slide (p. 27)

1) O compilador utilizado gera 11846 linhas de código para o programa
"hello" (ver "exe_01/program.ii").
Compilador:
gcc (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4

2)      Modo 1: Definir função "violateODR" como "inline" em "a.h"
                (ver "exe_02/fix1/")
        Modo 1: Definir função "violateODR" em um arquivo separado
"a.cpp"         (ver "exe_02/fix2/")

3) Uma vez que C++ suporta function overloading, o nome de uma função
não pode ser utilizado como um ID único durante a linkagem,
por isso ele modifica o nome da função adicionando a ele informação dos
seus parâmetros (mangling).
A expressão "extern 'C'" faz com que o compilador não realize o
mangling de forma que código em C possa "linkar" uma função C++.
Um outro caso de uso é um programa C++ precisar utilizar uma biblioteca
C, o que pode ser feito declarando as a interface C como
"extern 'C'".
Fonte: http://stackoverflow.com/questions/1041866/in-c-source-what-is-
the-effect-of-extern-c

4) Se um uma diretiva "using" for utilizada em um arquivo header, todos
os arquivos que incluirem o header irão inadvertidamente
importar o namespace, tornando-se susceptíveis a colisão de nomes.
Fonte: http://stackoverflow.com/questions/2232496/is-it-wrong-to-use-c-
using-keyword-in-a-header-file

5) Namespaces anônimos permitem declarar identificadores únicos para
uma translation unit. Isso permite utilizar identificadores
comuns sem correr risco de ocorrer colisões durante a linkagem. O uso
de namespaces apresenta algumas diferenças em relação a utilização
da keyword "static". Por exemplo, identificadores em namespaces
anonymous possuem linkagem externa, enquanto identificadores declarados
"static" possuem linkagem interna.
Não faz sentido utilizar namespaces anônimous em arquivos header porque
os identificadores definidos no header serão interpretados
como nomes únicos para a translation unit que o incluiu, de forma que
definições presentes em outras translation units não serão
encontradas durante a linkagem.
Fonte: http://www.comeaucomputing.com/techtalk/#nostatic

6) A macro interpreta qualquer elemento separado por vírgulas como um
parâmetro, de forma que um template com mais de um tipo
```

é interpretado como dois parâmetros pela macro (uma vez que os tipos do template são separados por vírgulas), gerando um erro de compilação.
Ver "exe_06/exe_06.cpp"
Ex:

```
        $ g++ exe_06.cpp -o exe_06.out
        exe_06.cpp:21:37: error: macro "SETTER" passed 3 arguments, but takes just 2
          SETTER(property, Tuple<int, double>);
                                             ^
        exe_06.cpp:21:2: error: ‘SETTER’ does not name a type
          SETTER(property, Tuple<int, double>);
  ^
```

7) Ver "exe_07/modified/":
        p1.cpp: função inline
        p2.cpp: enum
        p3.cpp: template
        p4.cpp: valor const

==========

*) Dúvidas:
1) Porque compilar múltiplos arquivos da questão 2, mesmo com os fixes, geram erros de compilação se a flag "-save-temps=obj" for usada?
Ex:

```
        $ cd exe_02/fix1/build
        $ gcc -save-temps=obj ../q.cpp ../p.cpp -o p.out
        p.o: In function `stuff()':
        p.cpp:(.text+0x0): multiple definition of `stuff()'
        p.o:p.cpp:(.text+0x0): first defined here
        p.o: In function `main':
        p.cpp:(.text+0xb): multiple definition of `main'
        p.o:p.cpp:(.text+0xb): first defined here
        collect2: error: ld returned 1 exit status
```

2) Porque ao compilar o arquivo p3.cpp da questão 7 utilizando gcc gera erros referentes à classe "string", mas compilando com g++ não há problema?
Fonte: http://stackoverflow.com/questions/3081815/c-errors-while-compiling
                http://stackoverflow.com/questions/19016253/g-undefined-reference-std
Ex.:

```
        gcc ../p3.cpp -o p3.out
        /tmp/cctHNXW0.o: In function `h()':
        p3.cpp:(.text+0x22): undefined reference to `std::allocator<char>::allocator()'
        p3.cpp:(.text+0x37): undefined reference to `std::basic_string<char, std::char_traits<char>, std::allocator<char> >::basic_string(char const*, std::allocator<char> const&)'
        p3.cpp:(.text+0x56): undefined reference to `std::basic_string<char, std::char_traits<char>, std::allocator<char> >::~basic_string()'
        p3.cpp:(.text+0x62): undefined reference to
```

```
`std::allocator<char>::~allocator()'
        p3.cpp:(.text+0x7f): undefined reference to
`std::basic_string<char, std::char_traits<char>, std::allocator<char>
>::~basic_string()'
        p3.cpp:(.text+0xa1): undefined reference to
`std::allocator<char>::~allocator()'
        /tmp/cctHNXW0.o: In function `MyClass<std::string>::~MyClass()':
        p3.cpp:(.text._ZN7MyClassISsED2Ev[_ZN7MyClassISsED5Ev]+0x14):
undefined reference to `std::basic_string<char, std::char_traits<char>,
std::allocator<char> >::~basic_string()'
        /tmp/cctHNXW0.o: In function `MyClass<std::string>::MyClass
(std::string)':
        p3.cpp:(.text._ZN7MyClassISsEC2ESs[_ZN7MyClassISsEC5ESs]+0x18):
undefined reference to `std::basic_string<char, std::char_traits<char>,
std::allocator<char> >::basic_string()'
        /tmp/cctHNXW0.o:(.eh_frame+0x4b): undefined reference to
`__gxx_personality_v0'
        collect2: error: ld returned 1 exit status

=====================================================================
*) Referências
http://www.manpages.info/linux/gcc.1.html
http://stackoverflow.com/questions/5370539/what-is-the-meaning-of-lines-
starting-with-a-hash-sign-and-number-like-1-a-c
[1] http://stackoverflow.com/questions/1041866/in-c-source-what-is-the-
effect-of-extern-c
https://isocpp.org/wiki/faq/mixing-c-and-cpp
http://stackoverflow.com/questions/2232496/is-it-wrong-to-use-c-using-
keyword-in-a-header-file
http://stackoverflow.com/questions/2448242/struct-with-template-
variables-in-c
http://wiki.ros.org/CppStyleGuide
http://stackoverflow.com/questions/3081815/c-errors-while-compiling
http://stackoverflow.com/questions/19016253/g-undefined-reference-std
```