Curso: C++ Moderno
Período: 1º semestre/2017
Aluno: Arthur Nunes de Paiva Santos Queiroz

Lista de exercícios Módulo 02: Compilação em detalhes

Questões

Questão 1

Quantas linhas de codigo seu pré-processador gera para o exemplo hello?

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

Questão 2

Corrija o problema da função violateODR( ) de duas maneiras diferentes.

Modo 1: Definir função "violateODR" como "inline" em "a.h"

a.h

```
#ifndef A_H__
#define A_H__

inline void violateODR(){}

#endif
```

Modo 2: Definir função "violateODR" em um arquivo separado "a.cpp"

a.cpp

```
#include "a.h"

void violateODR(){

}
```

Questão 3

Para que serve extern "C" e qual sua relação com o linker?

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

Questão 4

O que é uma diretiva using namespace e por quê nunca utilizá-la em um header?

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

Questão 5

O que é um namespace anônimo e por quê não faz sentido utilizá-lo em um header?

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

Questão 6

Qual erro de compilação você terá ao utilizar a macro SETTER, passando como argumento para TYPE a instância de um template de classe com mais de 1 parâmetro?

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.

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>);
  ^
```

Questão 7

Substitua todas as macros dos programas abaixo por um enum, por um valor const, por uma função inline ou por um template. Cada um deles só pode ser utilizado uma única vez.

p1.cpp: função inline

```
inline int ADD2(int A, int B) {
	return A+B;
}

void g(){
	int x = ADD2(1,10);
}

int main(){
	void g();
	return 0;
}
```

p2.cpp: enum

```
namespace Numbers{
	enum Number {
		One = 1,
		Two = 2,
	};
}
typedef Numbers::Number Number;

void k(int v){
	switch(v){
		case Numbers::One: ;
		case Numbers::Two: ;
	}
}

int main(){
	k(Numbers::One);
	return 0;
}
```

p3.cpp: template

```cpp
#include <string>

template<typename T>
struct MyClass {
	MyClass(T){}
	T data_;
};

using namespace std;
typedef MyClass<int> MyClass_int;
typedef MyClass<string> MyClass_string;

void h(){
	MyClass_int m(10);
	MyClass_string n("foo");
}

int main(){
	h();
	return 0;
}
```

p4.cpp: valor const

```cpp
const int SIZE = 100;

template<int size>
class MyArray{};

void f(){
	MyArray<SIZE> a;
}

int main(){
	f();
	return 0;
}
```