a)

file	symbol	internal unlinkable symbol	reference of external symbol	weak linkable symbol	strong linkable symbol
a.c:	x		✓		
a.c:	y			✓	
a.c:	f	✓			
a.c:	g				✓
a.c:	h				✓
b.c:	x				✓
b.c:	y	✓			
b.c:	z	✓			
b.c:	f				✓
b.c:	g	✓			

b)

```
output:  b.c: f()
         b.c: g()
         a.c: h()
         a.c: g()
         a.c: f()
```

a) main() will call the extern function f(), which is defined both in a.c and b.c, but since in a.c f() is static it cannot be accessed/referenced from outside of the object file, therefore the linker goes for the strong global symbol f which is defined in b.c. Next f() makes a call for g(), where the statically defined function in b.c will be called (static beats global in the obj file where the static func is defined). Next g() will call h() which is a reference to an external symbol, defined in a.c, so we enter a.c now, and execute h(). In a.c h() calls g(), since it cannot access g() in b.c since it is static it will execute g() from a.c., which will then execute the static f() same argument as before (static vs global). The calls stop here as f() does not call another function so main will return.

c) In order to support features such as overloaded functions, it was necessary to encode type information into the symbols. This is called name mangling. Since C++ is a languages which allows for function overloading, name mangling is necessary for encoding the symbols.

The extern "C" {...} is required in some C++ header files since it declares some functions or libraries to be of "C" linkage, rather than "C++" linkage. The purpose of this is to allow C++ code to interact with C code and to use different libraries written in C without "C++" in mind.