extreme hello worlding ☺

Zerlegen Sie das Problem - Ausgabe der Zeichenkette „Hello, World!“ - in eine völlig unangemessen hohe Anzahl von C++-Quelltexten, und machen Sie sich anhand der Aufgabe mit der Übersetzungstechnologie von C++ (Referenzsystem g++ 3.3.2 auf amsel) vertraut.

Verwenden Sie für alle Übersetzungsschritte den g++ mit folgenden Optionen:

-O3	- optimiere maximal
-Wall	- verwarne alles
-ansi	- lasse nur Standard C++ zu (keine gnu-Erweiterungen)
-pedantic	- sei so kleinlich wie möglich

Folgende Quelltexte (die Namen sind verbindlich) sollen zum Projekt gehören:

main.cc	-	ruft die Funktion **`void helloworld();`**
helloworld.cc	-	implementiert die Funktion **`void helloworld();`** diese ruft die Funktionen **`void H();`** und **`void elloworld();`**
elloworld.cc	-	implementiert die Funktion **`void elloworld();`** diese ruft die Funktionen **`void e();`** und **`void lloworld();`**
lloworld.cc	-	implementiert die Funktion **`void lloworld();`** diese ruft die Funktionen **`void l();`** (zweimal) und **`void oworld();`**
oworld.cc	-	implementiert die Funktion **`void oworld();`** diese ruft die Funktionen **`void o();`** , **`void ks();`** und **`void world();`**
world.cc	-	implementiert die Funktion **`void world();`** diese ruft die Funktionen **`void W();`** und **`void orld();`**
orld.cc	-	implementiert die Funktion **`void orld();`** diese ruft die Funktionen **`void o();`** und **`void rld();`**
rld.cc	-	implementiert die Funktion **`void ld();`** diese ruft die Funktionen **`void r();`** und **`void ld();`**
ld.cc	-	implementiert die Funktion **`void ld();`** diese ruft die Funktionen **`void l();`** , **`void d();`** und **`void fini();`**
H.cc	-	implementiert die Funktion **`void H();`** diese gibt den Buchstaben 'H' aus (**`std::cout<<'H';`**)
e.cc	-	implementiert die Funktion **`void e();`** diese gibt den Buchstaben 'e' aus
l.cc	-	implementiert die Funktion **`void l();`** diese gibt den Buchstaben 'l' aus
o.cc	-	implementiert die Funktion **`void o();`** diese gibt den Buchstaben 'o' aus
ks.cc	-	implementiert die Funktion **`void ks();`** diese gibt ein Komma und ein Leerzeichen aus
W.cc	-	implementiert die Funktion **`void W();`** diese gibt den Buchstaben 'W' aus
r.cc	-	implementiert die Funktion **`void r();`** diese gibt den Buchstaben 'r' aus
d.cc	-	implemetiert die Funktion **`void d();`** diese gibt den Buchstaben 'd' aus
fini.cc	-	implementiert die Funktion **`void fini();`** diese gibt ein Ausrufungszeichen und ein Zeilenendezeichen aus

Zu jeder *.cc-Datei (außer `main.cc`) gehört ein Headerfile, welches den Prototyp der jeweils implementierten Funktion zur Verfügung stellt. Für aufgerufene Funktionen wird der Prototyp durch Einschluss des/der entsprechenden Header bereitgestellt.

Erstellen Sie ein **`Makefile`**, welches sämtliche Abhängigkeiten innerhalb des Projektes enthält, so dass

`make`	-	ein jeweils aktuelles (d.h. alle Quelltextänderungen berücksichtigendes) ausführbares Programm namens `main` erstellt und
`make clean`	-	alle abhängigen (d.h. durch Übersetzung entstandenen) Dateien löscht.

Lesen Sie dazu ggf. unter `man makedepend` nach, wie dies mit Werkzeugunterstützung erfolgen kann.

Erstellen Sie zwei weitere Make-Files **`makefile.static`** und **`makefile.dynamic`**, mit welchen aus sämtlichen *.cc-Dateien (außer `main.cc`) jeweils eine statische bzw. dynamische Bibliothek erstellt wird, so dass das ausführbare Programm jeweils durch direkte Verwendung der Bibliotheken erstellt werden kann:

```
g++ … -o main main.cc  …… nur noch mit der statischen Bibliothek libhw.a …
```

bzw.

```
g++ … -o main main.cc  …… nur noch mit der dynamischen Bibliothek libhw.so …
```

(Die tatsächlich notwendigen Aufrufe sollen Sie selbst ermitteln!)

Vergleichen Sie die dabei jeweils entstehenden Binärfiles hinsichtlich ihrer Größe. Machen Sie sich die Ausgaben des Kommandos

```
nm main | c++filt
```

angewendet auf die jeweiligen Binärfiles klar.

Refaktorisieren Sie ihr Projekt in einem Unterverzeichnis `inline` derart, dass alle gerufenen Funktionen komplett im jeweiligen Headerfile als inline-Funktionen implementiert werden (z.B. `inline void H(){ …/*implementation*/ }`). Dabei entfallen (bis auf `main.cc`) die *.cc-Dateien. `main.cc` selbst soll dabei unverändert bleiben!

Machen Sie sich wiederum die Ausgaben des Kommandos

```
nm main | c++filt
```

angewendet auf das entstehende Binärfile klar. Erzeugen Sie für die (statische) Bibliotheksversion und die inline-Version von main die jeweilige Assembler-Ausgaben des `g++` (Schalter –S) und versuchen Sie die strukturellen Unterschiede zu verstehen.

Lesen Sie ggf. die Abschnitte 3.5 und 6.1 des Buches ‚C++ Entwicklung mit Linux' von Thomas Wieland (ISBN 3-89864-307-7, online unter http://www.cpp-entwicklung.de) und ggf. natürlich die Online-Manuals der erforderlichen Kommandos (Solaris ist eben **kein** Linux: vieles funktioniert ähnlich, aber nicht 100% genauso wie im Buch beschrieben)