Ejercicio 8.1.

Pruebe a comentar en el archivo fuente main.cpp la directiva de procesamiento “#include ”functions.h”. La línea quedaría así: //#include ”functions.h”. Pruebe a generar ahora el módulo objeto con la orden de compilación mostrada anteriormente. ¿Qué ha ocurrido?

```
juanka1995@juanka1995-VBox:~/modulo2/sesion08/dirprograma1$ g++ -c main.cpp
main.cpp: In function ‘int main()’:
main.cpp:7:17: error: ‘print_hello’ was not declared in this scope
     print_hello();
                 ^
main.cpp:9:52: error: ‘factorial’ was not declared in this scope
     cout << "The factorial of 7 is " << factorial(7) << endl;
                                                    ^
```

Ejercicio 8.2.

Explique por qué el enlazador no ha podido generar el programa archivo ejecutable programa2 del ejemplo anterior y, sin embargo, ¿por qué sí hemos podido generar el módulo main2.o?

No se ha podido generar el programa ejecutable debido a que este requiere del uso de ciertas librerías para su creación, las cuales no hemos especificado previamente.
Podemos generar el modulo main2.o debido a que esto solo realiza las etapas de preproceso y compilación. En otras palabras no existen fallos en el código por lo que se puede generar el modulo.

Ejercicio 8.3.

Explique por qué la orden g++ previa ha fallado. Explique los tipos de errores que ha encontrado.

```
juanka1995@juanka1995-VBox:~/modulo2/sesion08/dirprograma2$ g++ -L./ -o programa
2 main2.cpp factorial.cpp hello.cpp -lmates
main2.cpp:2:23: fatal error: functions.h: No existe el archivo o el directorio
 #include "functions.h"
                       ^
compilation terminated.
factorial.cpp:1:23: fatal error: functions.h: No existe el archivo o el director
io
 #include "functions.h"
                       ^
compilation terminated.
hello.cpp:2:23: fatal error: functions.h: No existe el archivo o el directorio
 #include "functions.h"
                       ^
compilation terminated.
```

El error viene producido porque hemos movido las librerías a otro directorio y ahora con la orden ejecutada, nuestro programa no es capaz de generarse debido a que no encuentra dichas librerías en el directorio especificado en la orden.

Ejercicio 8.4.

Copie el contenido del makefile previo a un archivo llamado makefileE ubicado en el mismo directorio en el que están los archivos de código fuente .cpp. Pruebe a modificar distintos archivos .cpp (puede hacerlo usando la orden touch sobre uno o varios de esos archivos) y compruebe la secuencia de instrucciones que se muestra en el terminal al ejecutarse la orden make. ¿Se genera siempre la misma secuencia de órdenes cuando los archivos han sido modificados que cuando no? ¿A qué cree puede deberse tal comportamiento?

No, no se genera la misma secuencia de ordenes cuando los archivos han sido modificados. Si modificásemos dos archivos solo se ejecutarían dichos archivos para actualizar su contenido.

Se debe a que el fichero makefile se encarga de comprobar que ficheros han sido modificados y en caso de haber alguno ejecutar dicho fichero.

```
juanka1995@juanka1995-VBox:~/modulo2/sesion08/dirprograma2$ make -f makefileE
g++ -I./includes -c main2.cpp
g++ -I./includes -c factorial.cpp
g++ -I./includes -c hello.cpp
g++ -I./includes -c sin.cpp
g++ -I./includes -c cos.cpp
g++ -I./includes -c tan.cpp
ar -rvs libmates.a sin.o cos.o tan.o
ar: creando libmates.a
a - sin.o
a - cos.o
a - tan.o
g++ -L./ -o programa2 main2.o factorial.o hello.o -lmates
juanka1995@juanka1995-VBox:~/modulo2/sesion08/dirprograma2$ touch main2.cpp
juanka1995@juanka1995-VBox:~/modulo2/sesion08/dirprograma2$ make -f makefileE
g++ -I./includes -c main2.cpp
g++ -L./ -o programa2 main2.o factorial.o hello.o -lmates
juanka1995@juanka1995-VBox:~/modulo2/sesion08/dirprograma2$ touch hello.cpp
juanka1995@juanka1995-VBox:~/modulo2/sesion08/dirprograma2$ make -f makefileE
g++ -I./includes -c hello.cpp
g++ -L./ -o programa2 main2.o factorial.o hello.o -lmates
juanka1995@juanka1995-VBox:~/modulo2/sesion08/dirprograma2$ make -f makefileE
make: «programa2» está actualizado.
```

Ejercicio 8.5.

Obtener un nuevo makefileF a partir del makefile del ejercicio anterior que incluya además las dependencias sobre los archivos de cabecera. Pruebe a modificar cualquier archivo de cabecera (usando la orden touch) y compruebe la secuencia de instrucciones que se muestra en el terminal al ejecutarse la orden make.

```
juanka1995@juanka1995-VBox:~/modulo2/sesion08/dirprograma2$ touch includes/funct
ions.h
juanka1995@juanka1995-VBox:~/modulo2/sesion08/dirprograma2$ make -f makefileF
g++ -I./includes -c main2.cpp
g++ -I./includes -c factorial.cpp
g++ -I./includes -c hello.cpp
g++ -L./ -o programa2 main2.o factorial.o hello.o -lmates
juanka1995@juanka1995-VBox:~/modulo2/sesion08/dirprograma2$ touch includes/mates
.h
juanka1995@juanka1995-VBox:~/modulo2/sesion08/dirprograma2$ make -f makefileF
g++ -I./includes -c sin.cpp
g++ -I./includes -c cos.cpp
g++ -I./includes -c tan.cpp
ar -rvs libmates.a sin.o cos.o tan.o
r - sin.o
r - cos.o
r - tan.o
g++ -L./ -o programa2 main2.o factorial.o hello.o -lmates
juanka1995@juanka1995-VBox:~/modulo2/sesion08/dirprograma2$
```

Dependiendo del archivo de cabecera que modifiquemos la secuencia sera una u otra, ejecutando de nuevo los ficheros que dependendan de dicha cabecera.

Ejercicio 8.6.

Usando como base el archivo makefileG, sustituya la línea de orden de la regla cuyo objetivo es programa2 por otra en la que se use alguna de las variables especiales y cuya ejecución sea equivalente.

```
programa2: main2.o factorial.o hello.o libmates.a
	$(CC) -L$(LIB_DIR) -o $@ $^ -lmates
```

Ejercicio 8.7.

Utilizando como base el archivo makefileG y los archivos fuente asociados, realice los cambios que considere oportunos para que, en la construcción de la biblioteca estática libmates.a, este archivo pase a estar en un subdirectorio denominado libs y se pueda enlazar correctamente con el resto de archivos objeto.

```
# Variable que indica el directorio en donde se encuentran las bibliotecas
LIB_DIR= ./libs

programa2: main2.o factorial.o hello.o libmates.a
	$(CC) -L$(LIB_DIR) -o $@ main2.o factorial.o hello.o -lmates

main2.o: main2.cpp
	$(CC) -I$(INCLUDE_DIR) -c main2.cpp

factorial.o: factorial.cpp
	$(CC) -I$(INCLUDE_DIR) -c factorial.cpp

hello.o: hello.cpp
	$(CC) -I$(INCLUDE_DIR) -c hello.cpp

libmates.a: sin.o cos.o tan.o
	ar -rvs $(LIB_DIR)/libmates.a sin.o cos.o tan.o

sin.o: sin.cpp
	$(CC) -I$(INCLUDE_DIR) -c sin.cpp

cos.o: cos.cpp
	$(CC) -I$(INCLUDE_DIR) -c cos.cpp

tan.o: tan.cpp
	$(CC) -I$(INCLUDE_DIR) -c tan.cpp
```

Ejercicio 8.8.

Busque la variable predefinida de make que almacena la utilidad del sistema que permite construir bibliotecas. Recuerde que la orden para construir una biblioteca estática a partir de una serie de archivos objeto es ar (puede usar la orden grep para filtrar el contenido; no vaya a leer línea a línea toda la salida). Usando el archivo makefileG, sustituya la orden ar por su variable correspondiente.

```
libmates.a: sin.o cos.o tan.o
	$(AR) -rvs $(LIB_DIR)/libmates.a sin.o cos.o tan.o
```

Ejercicio 8.9.

Dado el siguiente archivo makefile, explique las dependencias que existen y para qué sirve cada una de las líneas del mismo. Enumere las órdenes que se van a ejecutar a consecuencia de invocar la utilidad make sobre este archivo.

```
# Nombre archivo: makefileH
# Uso: make -f makefileH
# Descripción: Mantiene todas las dependencias entre los módulos que utiliza el
# programa1.

CC=g++
CPPFLAGS=-Wall -I./includes
SOURCE_MODULES=main.cpp factorial.cpp hello.cpp
OBJECT_MODULES=$(SOURCE_MODULES:.cpp=.o)
EXECUTABLE=programa1

all: $(OBJECT_MODULES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECT_MODULES)
	$(CC) $^ -o $@

# Regla para obtener los archivos objeto .o que dependerán de los archivos .cpp
# Aquí, $< y $@ tomarán valores respectivamente main.cpp y main.o y así sucesivamente
.o: .cpp
	$(CC) $(CPPFLAGS) $< -o $@
```

1. La regla **all** creara la variable **OBJECT_MODULES** con los nombres de los objetos necesarios a utilizar los cuales provienen del cambio de extension **.ccp** de los ficheros contenidos en la variable **SOURCE_MODULES** por la extension **.o**

2. Seguidamente la regla **all** llamara a otra regla llamada **programa1**, cuyo nombre esta almacenado en otra variable definida como **EXECUTABLE**.

3. Y posteriormente, esta regla se encargara de crear un ejecutable con el nombre de **programa1** a partir de sus dependencias, las cuales son los **ficheros objet**o conseguido con la ejecución de la regla **all** en el apartado 1, almacenados en la variable **OBJECT_MODULES**.

Ejercicio 8.10.
Con la siguiente especificación de módulos escriba un archivo denominado makefilePolaca que automatice el proceso de compilación del programa final de acuerdo a la siguiente descripción:

Compilador: gcc o g++
Archivos cabecera: calc.h (ubicado en un subdirectorio denominado cabeceras)
Archivos fuente: main.c stack.c getop.c getch.c
Nombre del programa ejecutable: calculadoraPolaca

Además, debe incluir una regla denominada borrar, sin dependencias, cuya funcionalidad sea la de eliminar los archivos objeto y el programa ejecutable.

Nombre archivo: makefilePolaca
Uso: make -f makefilePolaca
Descripción: Mantiene todas las dependencias entre los módulos que utiliza
el calculadoraPolaca.

# Variable que indica el compilador que se va a utilizar
CC=g++

# Variable que indica el directorio en donde se encuentran los archivos de cabecera
INCLUDE_DIR= ./cabeceras

```
calculadoraPolaca: main.o stack.o getop.o getch.o
	$(CC) -o $@ $^

main.o: main.c
	$(CC) -I$(INCLUDE_DIR) -c main.c

stack.o: stack.c
	$(CC) -I$(INCLUDE_DIR) -c stack.c

getop.o: getop.c
	$(CC) -I$(INCLUDE_DIR) -c getop.c

getch.o: getch.c
	$(CC) -I$(INCLUDE_DIR) -c getch.c

borrar:
	rm *.o calculadoraPolaca
```