Makefile: Automatización de procedimientos

Para automatizar el proceso de desarrollo se entrega el archivo `Makefile` que consiste en un conjunto de reglas para la utilidad `make`.
Cada regla consiste en un `objetivo`, las `acciones` para conseguir el objetivo y las `dependencias` del objetivo. Cuando el objetivo y las dependencias son archivos, las acciones se ejecutan cuando el objetivo no está actualizado respecto a las dependencias (o sea, es un archivo que no existe o su fecha de modificación es anterior a la de alguna de las dependencias). Por más información ver el manual de `make`: https://www.gnu.org/software/make/manual/.

En el *Makefile* entregado las reglas incluidas son:

- principal: para compilar y enlazar.
- clean: para borrar archivos.
- testing: para hacer pruebas.
- entrega: para generar el archivo a entregar.

make principal La regla `principal` compila `principal.cpp`, `lista.cpp`, `pila.cpp`, `cola.cpp` y `mapping.cpp` y luego genera el ejecutable *principal*. Esta regla es la predeterminada, o sea que es la que se invoca si no se especifica ninguna.

En la siguiente secuencia se ve una ejecución exitosa de *make* junto con el estado de los directorios antes y después, donde *ls* es un comando que nos muestra los archivos y directorios dentro del directorio actual.

```
$ ls
include  Makefile  obj  principal.cpp  src  test

$ ls obj/

$ make
Compilando principal.cpp
Compilando src/lista.cpp
Compilando src/pila.cpp
Compilando src/cola.cpp
Compilando src/mapping.cpp
Compilando y enlazando principal

$ ls
include  Makefile  obj  principal  principal.cpp  src  test

$ ls obj/
cola.o lista.o  mapping.o  pila.o  principal.o
```

Si no se hacen cambios y se vuelve a correr `make` no se hace nada.

```
$ make
make: No se hace nada para «all».
```

Si hay errores de compilación se muestran en la salida estándar. En el siguiente ejemplo se muestra que en la funcion `remover` de `lista.cpp` no hay instrucción `return` ya que por error se habría comentado la línea que lo contenía.

```
$ make
Compilando src/lista.cpp
src/lista.cpp: In function ‘_rep_lista* remover(nat, TLista)’:
src/lista.cpp:62:1: error: no return statement in function returning non-void [-Werror=return-type]
```

```
 }
 ^
cc1plus: all warnings being treated as errors
Makefile:70: recipe for target 'obj/lista.o' failed
make: *** [obj/lista.o] Error 1
```

Al volver a ejecutar make después de corregir el error solo se compila la que es necesario:

```
$ make
Compilando src/lista.cpp
Compilando y enlazando principal
```

es decir, no se vuelven a compilar `pila.cpp` ni `cola.cpp` ni `mapping.cpp` porque ya estaban correctamente compilados.

make testing Con la regla `testing` se ejecuta el programa con los casos de entrada (*.in*) generando archivos con la extensión *.sal* y estos se comparan con las salidas esperadas (*.out*), obteniendo archivos con extensión *.diff*.
Si no existe el ejecutable, se proceda a la compilación para obtenerlo. Esto es lo que se ve en el siguiente ejemplo (suponiendo que los casos de prueba son 00, 01, 02 y 03):

```
$ make testing
Compilando principal.cpp
Compilando src/lista.cpp
Compilando src/pila.cpp
Compilando src/cola.cpp
Compilando src/mapping.cpp
Compilando y enlazando principal
./principal < test/00.in > test/salidas/00.sal
./principal < test/01.in > test/salidas/01.sal
./principal < test/02.in > test/salidas/02.sal
./principal < test/03.in > test/salidas/03.sal
-- RESULTADO DE CADA CASO --
1111
```

Al final se imprime una secuencia de ceros y unos. Cada elemento de la secuencia corresponde a un caso. Un 1 significa que el resultado es correcto y un 0 significa que hay un error.
Si los archivos a comparar no son iguales se indica en la salida estándar.

```
$ make testing
./principal < test/00.in > test/salidas/00.sal
./principal < test/01.in > test/salidas/01.sal
./principal < test/02.in > test/salidas/02.sal
./principal < test/03.in > test/salidas/03.sal
---- ERROR en caso test/salida/03.diff ----
-- CASOS CON ERRORES --
03
-- RESULTADO DE CADA CASO --
1110
```

Si se vuelve a correr sólo se muestran los casos con errores:

```
-- CASOS CON ERRORES --
03
-- RESULTADO DE CADA CASO --
1110
```

Se puede ver que los archivos `.diff` de los casos con errores no tienen tamaño nulo.

```
$ stat --print="%n  %s \n" test/*.diff
test/00.diff 0
test/01.diff 0
test/02.diff 0
test/03.diff 388
```

make clean En ocasiones puede ser útil borrar los archivos generados. Esto se hace con `make clean`. Si solo se desea borrar los archivos generados por la compilación (`lista.o`, `pila.o`, `cola.o`, `mapping.o`, `principal.o` y `principal`) se debe invocar `make clean_bin`. Para borrar solo los archivos generados por la ejecución de *principal* se debe invocar `make clean_test`.

make entrega Mediante `make entrega` se obtiene el archivo a entregar, *Entrega1.tar.gz*.

```
$ make entrega
shasum src/pila.cpp src/cola.cpp src/mapping.cpp > claves.txt
tar zcf Entrega1.tar.gz claves.txt -C src pila.cpp cola.cpp mapping.cpp
-- El directorio y archivo a entregar es:
  /tarea1/Entrega1.tar.gz
```

El archivo a entregar contiene los módulos implementados y el archivo *claves.txt* que consiste en una clave por cada uno de esos módulos. Este último archivo puede generarase de manera independiente mediante la regla **claves** y puede usarse para comprobar que el archivo que se subió al receptor es el correcto.