Sesión 4: Módulos en C

Hoja de problemas

Programación para Sistemas

Ángel Herranz
aherranz@fi.upm.es
Universidad Politécnica de Madrid

2020-2021

Ejercicio 1. Repasa las transparencias de clase. Presta especial atención a los detalles sintácticos del fichero `Makefile`. Intenta entender el *guión* que lleva a solucionar cada problema que va surgiendo.

Ejercicio 2. En las transparencias puedes encontrar varias referencias, una al K&R, capítulo 4 y otras dos para la herramienta `make`. No dejes de tenerlas a mano y explorarlas:

Chapter 4. Functions and Program Structure
(*The C Programming Language*, K&R 2nd. edition)

GNU Make Manual
The Free Software Foundation (FSF)

`http://makefiletutorial.com/`
Chase Lambert

Ejercicio 3. Asegúrate de realizar todos los ejercicios de las transparencias antes de continuar. Deberías tener en un directorio los siguientes ficheros: `lcg1.c`, `sum1.c`, `lcg2.c`, `generador_lcg.c`, `generador_lcg.h` y un maravilloso `Makefile` con el que puedes *fabricar* los ejecutables `lcg1`, `sum1`, y `lcg2`.

Ejercicio 4. Recuerda que la herramienta `make` tiene algunas reglas por defecto que no es necesario que tú mismo escribas. Dependiendo de la instalación las reglas pueden variar (aunque las relacionadas con el lenguaje C son bastante estables en todas las distribuciones de Unix). Para ver las reglas predefinidas ejecuta:

```
$ make -p -f/dev/null
```

Ejercicio 5. Antes de continuar, vamos a enriquecer el `Makefile` para que con la simple ejecución de `make` se creen todos los ejecutables. Para ello basta con que añadas esta regla justo después de que establezcas la variable `CFLAGS`:

```
CFLAGS=-Wall -g -pedantic

todos: lcg1 lcg2 sum1

...
```

Prueba ahora a ejecutar

```
$ make todos
```

o simplemente

```
$ make todos
```

Ejercicio 6. A veces, después de ejecutar `make`, el directorio en el que estás trabajando se llena de ficheros *feos* o inservibles. Vamos a añadir un par de reglas al final a nuestro el `Makefile` para realizar una limpieza:

```
...

limpio:
        rm -f *.o

muylimpio: limpio
        rm -f lcg1 lcg2 sum1 core
```

Ahora, la ejecución de

```
$ make limpio
```

borrará todos los ficheros `.o` y

```
$ make muylimpio
```

hará lo mismo que `make limpio` y además borrará todos los ejecutables.

Nota: por convención, la mayor parte de los programadores usan `clean` y `veryclean` como objetivos así que es habitual ver:

```
$ make clean
```

y

```
$ make veryclean
```

El contenido final del Makefile debería ser algo parecido a esto:

```
CFLAGS=-Wall -g -pedantic

todos: lcg1 sum1 lcg2 test_lcg

lcg1: lcg1.o
	$(CC) $(CFLAGS) -o lcg1 lcg1.o

sum1: sum1.o
	$(CC) $(CFLAGS) -o sum1 sum1.o

generador_lcg.o: generador_lcg.c generador_lcg.h

lcg2.o: lcg2.c generador_lcg.h

lcg2: lcg2.o generador_lcg.o
	$(CC) $(CFLAGS) -o $@ $^

test_lcg2.o: test_lcg2.c generador_lcg.h

test_lcg: test_lcg.o generador_lcg.o
	$(CC) $(CFLAGS) -o $@ $^

clean:
	rm -f *.o

veryclean: clean
	rm -f lcg1 sum1 lcg2 test_lcg core
```

Ejercicio 7. Ejecución paso a paso: `gdb lcg1`

- Poner en marchar el depurador `gdb`.
- Colocar un *breakpoint* en `main`.
- Ejecutar el programa paso a paso y explorar las variables[1]
- ¿Puedes ver el valor de `anterior` cuando está ejecutando `main`? ¿Y el valor de x?
- ¿Puedes ver el valor de `i`, variable de `main` cuando está ejecutando `generar_aleatorio`?

Ejercicio 8. Ejecución paso a paso: `gdb sum1` (usa un número bajito `;)`)

- Poner en marchar el depurador `gdb`.
- Colocar un *breakpoint* en `main`.

[1] Ver transparencias de la sesión 2

- Ejecutar el programa paso a paso y explorar las variables[2]
- ¿Puedes ver el valor de `n` cuando está ejecutando `main`? ¿Y el valor de `i`?
- ¿Puedes ver el valor de `n`, variable de `main` cuando está ejecutando `sum`?

Ejercicio 9. Tu labor en este ejercicio será modificar el módulo `generador_lcg` y enriquecerlo de la siguiente forma.

- La idea es tener un módulo desde el que poder cambiar en tiempo de ejecución los parámetros a, c y m del generador (ahora no es posible porque se han definido como macros).
- El módulo va a contener tres variables: `lcg_a`, `lcg_c`, y `lcg_m`.
- Además, se expondrá una operación para establecer la *semilla* del generador, es decir, el valor de X_0.

Para facilitar la tarea dispones del *header* aquí:

generador_lcg.h

```
/* Parámetros a, c, y m del generador */
extern int lcg_a = 1;
extern int lcg_c = 1;
extern int lcg_m = 1;

/* Resetea el generador colocando el valor de X_0 al valor de semilla */
extern void lcg_resetear(int semilla);

/* Devuelve el valor de X y genera el siguiente */
extern int lcg_generar();
```

Para que puedas ver si lo que has hecho funciona correctamente puedes usar este *tester*:

test_lcg.h

```
#include <stdio.h>
#include <assert.h>
#include "generador_lcg.h"
int main() {

  lcg_a = 7;
  lcg_c = 1;
  lcg_m = 11;

  lcg_resetear(9);

  assert(lcg_generar() == 9);
```

[2] Ver transparencias de la sesión 2

```
  /* 7 * 9 + 1 % 11 == 9 */
  assert(lcg_generar() == 9);

  lcg_resetear(0);

  assert(lcg_generar() == 0);
  /* 7 * 0 + 1 % 11 == 1 */
  assert(lcg_generar() == 1);
  /* 7 * 1 + 1 % 11 == 8 */
  assert(lcg_generar() == 8);
  /* 7 * 8 + 1 % 11 == 2 */
  assert(lcg_generar() == 2);
}
```