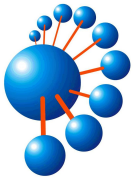
Programación de Computadores
2023-2

Tema 8: Compilación y debugging

Universidad de Concepción

José Fuentes - jfuentess@inf.udec.cl

Departamento de Ingeniería Informática y Ciencias de la Computación



Makefile

- Cuando tenemos proyectos con muchos archivos .c, .h y ejecutables, **make** y **Makefiles** son muy útiles
  - ¿Te ha pasado que realizas un pequeño cambio y debes compilar todo? Ahora ya no será necesario

- Makefiles expresan las dependencias entre distintos archivos, definiendo reglas para generar unos a partir de otros

- Para que funcione, el archivo Makefile (así debe llamarse) debe estar en la misma carpeta que los otros archivos .c y .h

Makefile: Un ejemplo

Consideremos estos archivos

sum_power2.h

```c
// Declaración de la función
unsigned long sum_power2(int[], int);
```

main.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>
#include <time.h>
#include "sum_power2.h"

#define N 1000000

int main() {
  // Variables especiales para medir tiempo de ejecución
  struct timespec stime, etime;
  double t;

  // Declaramos un arreglo de N enteros
  int numbers[N];

  // Llenamos el arreglo con N valores seleccionados al azar en el rango [0,9]
  for(int i=0; i < N; i++)
    numbers[i] = rand() % 10;

  // Función para medir tiempo de ejecución
  if (clock_gettime(CLOCK_THREAD_CPUTIME_ID , &stime)) {
    fprintf(stderr, "clock_gettime failed");
    exit(-1);
  }

  unsigned long total = sum_power2(numbers, N);

  printf("La suma de cuadrados es %lu\n", total);
  if (clock_gettime(CLOCK_THREAD_CPUTIME_ID , &etime)) {
    fprintf(stderr, "clock_gettime failed");
    exit(-1);
  }

  t = (etime.tv_sec - stime.tv_sec) + (etime.tv_nsec - stime.tv_nsec) / 1000000000.0;
  printf("Tiempo de cómputo: %f secs\n", t); // tiempo en segundos

  return 0;
}
```

sum_power2_v2.c

```c
#include <math.h>
#include "sum_power2.h"


unsigned long sum_power2(int A[], int N) {
  unsigned long total = 0;

  for(int i=0; i < N; i++) {
    total += pow(A[i], 2);
  }

  return total
}
```

sum_power2_v1.c

```c
#include "sum_power2.h"


unsigned long sum_power2(int A[], int N) {
  unsigned long total = 0;

  for(int i=0; i < N; i++) {
    total += A[i]*A[i];
  }

  return total;
}
```

Makefile: Un ejemplo

Este Makefile permite compilarlos

Contenido del archivo **Makefile**

```
CC = gcc
CFLAGS = -Wall -Wextra

sum_v1: main.o sum_power2_v1.o
	$(CC) $(CFLAGS) -o sum_v1 main.o sum_power2_v1.o

sum_v2: main.o sum_power2_v2.o
	$(CC) $(CFLAGS) -o sum_v2 main.o sum_power2_v2.o -lm

sum_power_v1.o: sum_power_v1.c
	$(CC) $(CFLAGS) -c sum_power_v1.c

sum_power_v2.o: sum_power_v2.c
	$(CC) $(CFLAGS) -c sum_power_v2.c

clean:
	rm *.o sum_v1 sum_v2
```

En la **consola**

```
make sum_v1
make sum_v2
make clean
```

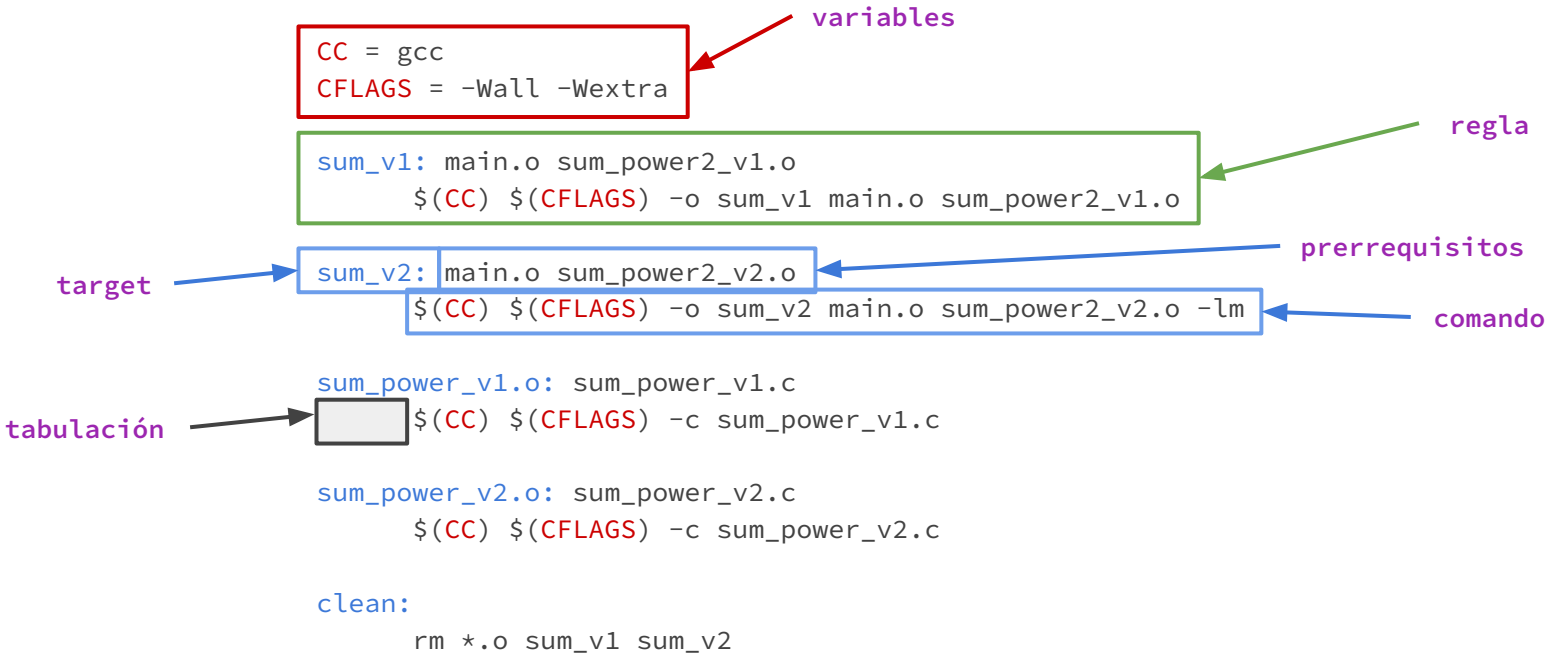
Intenta lo siguiente

Ejecuta **make sum_v1**
¿Cuántos archivos se compilan?

Ahora ejecuta **make sum_v2**
¿Cuántos archivos se compilan?
¿Por qué esa diferencia?

Makefile: Un ejemplo

Un poco de nomenclatura



Nota: Recordar el proceso de compilación de la primera clase

Debugging: GDB

- GDB es el debugger de GNU
- Permite ejecutar código de tal manera que permite
  - Pausar y reanudar la ejecución
  - Imprimir los valores de variables durante la ejecución
  - Buscar el lugar donde errores importantes puede ocurrir (Ej. error como **Segmentation Fault**)

- **Nota:** Para usar GDB (y otros debuggers) necesitas compilar con la opción **-g**

Debugging: GDB - un ejemplo

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void string_reverse(char *str) {
5        const int len = strlen(str);
6        for(int i = 0; i < len; i++) {
7              str[i] = str[len-i-1]; // swap characters
8              str[len-i-1] = str[i];
9        }
10 }
11 int main() {
12       char reverse_me[] = "AAABBB";
13       string_reverse(reverse_me);
14       printf("%s\n", reverse_me);
15       return 0;
16 }
```

Este programa tiene un error ¿logras ver cuál es el error?

En lugar de imprimir "BBBAAA" imprime "BBBBBB"

Ver: ejemplo_gdb.c

Ejemplo obtenido de https://www.cs.jhu.edu/~langmea/resources/lecture_notes/210_gdb.pdf

Debugging: GDB - un ejemplo (cont.)

```c
#include <stdio.h>
#include <string.h>

void string_reverse(char *str) {
      const int len = strlen(str);
      for(int i = 0; i < len; i++) {
            str[i] = str[len-i-1]; // swap characters
            str[len-i-1] = str[i];
      }
}
int main() {
      char reverse_me[] = "AAABBB";
      string_reverse(reverse_me);
      printf("%s\n", reverse_me);
      return 0;
}
```

Ver: ejemplo_gdb.c

Paso 1

```
> gcc -g -o test ejemplo_gdb.c
```

Paso 2

```
> gdb ./test
```

Paso 3: Añadir un *break point* en la función main

```
(gdb) break main
Breakpoint 1 at 0x120c: file ejemplo_gdb.c,
line 11.
```

Paso 4: Ejecutamos el programa, el que se detiene en el break point

```
(gdb) run
Starting program: …
Breakpoint 1, main () at ejemplo_gdb.c:11
11    int main() {
```

Paso 5: Nos movemos a la siguiente línea

```
(gdb) next
12      char reverse_me[] = "AAABBB";
```

Ejemplo obtenido de https://www.cs.jhu.edu/~langmea/resources/lecture_notes/210_gdb.pdf

Debugging: GDB - un ejemplo (cont.)

```c
#include <stdio.h>
#include <string.h>

void string_reverse(char *str) {
      const int len = strlen(str);
      for(int i = 0; i < len; i++) {
            str[i] = str[len-i-1]; // swap characters
            str[len-i-1] = str[i];
      }
}
int main() {
      char reverse_me[] = "AAABBB";
      string_reverse(reverse_me);
      printf("%s\n", reverse_me);
      return 0;
}
```

Ver: ejemplo_gdb.c

Paso 6: Nos movemos a la siguiente línea

```
(gdb) next
13      string_reverse(reverse_me);
```

Paso 7: Entramos a la función string_reverse

```
(gdb) step
string_reverse (str=0x5555555552bd "AAABB")
at ejemplo_gdb.c:4
4     void string_reverse(char *str) {
```

Paso 8: Nos movemos a la siguiente línea (2 veces - *next* y *n* son equivalentes)

```
(gdb) next
(gdb) n
6       for(int i = 0; i < len; i++) {
```

Paso 9: Imprimimos el valor de *len*

```
(gdb) p len
$2 = 6
```

Ejemplo obtenido de https://www.cs.jhu.edu/~langmea/resources/lecture_notes/210_gdb.pdf

Debugging: GDB - un ejemplo (cont.)

```c
#include <stdio.h>
#include <string.h>

void string_reverse(char *str) {
      const int len = strlen(str);
      for(int i = 0; i < len; i++) {
            str[i] = str[len-i-1]; // swap characters
            str[len-i-1] = str[i];
      }
}
int main() {
      char reverse_me[] = "AAABBB";
      string_reverse(reverse_me);
      printf("%s\n", reverse_me);
      return 0;
}
```

Ver: ejemplo_gdb.c

```
Paso 10: Verificamos el primer swap del
ciclo (línea 7)
(gdb) n
7 str[i] = str[len-i-1]; // swap characters

(gdb) p i
$3 = 0

(gdb) p str[i]
$4 = 65 'A'

(gdb) p str[len-i-1]
$5 = 66 'B'

(gdb) n
8 str[len-i-1] = str[i];

(gdb) n
6 for(int i = 0; i < len; i++) {

(gdb) p i
$6 = 0
Completamos un ciclo
```

Ejemplo obtenido de https://www.cs.jhu.edu/~langmea/resources/lecture_notes/210_gdb.pdf

Debugging: GDB - un ejemplo (cont.)

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void string_reverse(char *str) {
5        const int len = strlen(str);
6        for(int i = 0; i < len; i++) {
7              str[i] = str[len-i-1]; // swap characters
8              str[len-i-1] = str[i];
9        }
10 }
11 int main() {
12       char reverse_me[] = "AAABBB";
13       string_reverse(reverse_me);
14       printf("%s\n", reverse_me);
15       return 0;
16 }
```

Ver: ejemplo_gdb.c

```
Verifiquemos si el swap se hizo
correctamente
(gdb) p str[i]
$7 = 66 'B'

(gdb) p str[len-i-1]
$8 = 66 'B'
```

Error! **str[i]** debería contener la letra **'B'** y **str[len-i-1]** la letra **'A'**

¿Cómo lo solucionamos?

Ejemplo obtenido de https://www.cs.jhu.edu/~langmea/resources/lecture_notes/210_gdb.pdf

Debugging: GDB - un ejemplo (cont.)

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void string_reverse(char *str) {
5        const int len = strlen(str);
6        for(int i = 0; i < len; i++) {
7              char tmp = str[i];
8              str[i] = str[len-i-1]; // swap characters
9              str[len-i-1] = tmp;
10       }
11 }
12 int main() {
13       char reverse_me[] = "AAABBB";
14       string_reverse(reverse_me);
15       printf("%s\n", reverse_me);
16       return 0;
17 }
```

Solución

Ver: ejemplo_gdb.c

Ejemplo obtenido de https://www.cs.jhu.edu/~langmea/resources/lecture_notes/210_gdb.pdf

- **Nota 1:** Volveremos a repasar el uso de GDB cuando veamos punteros y memoria dinámica (¡es muy útil!)

- **Nota 2:** GDB no es el único debugger. Recomiendo que revisen **Valgrind** para detectar errores de memoria (te ahorrará muchos dolores de cabeza)