# Tablas de ayuda - Convención C

Organización del Computador II

## 1 Esquema de resolución para programar en C/ASM

Estas listas son una referencia, lo importante es que empiecen a desarrollar una metodología con la cual atacar el programa que van a escribir.

### 1.1 En alto nivel (C)

Para escribir el código:

- Escribir un planteo informal de resolución e identificar aquellos elementos que les generen dudas.
- Escribir un pseudocódigo de la solución.
- Identificar qué parte de la información va a estar en el `heap` y qué parte en el `stack,.data`.
- Identificar cuándo y de qué forma se van a inicializar los datos.
- Si se manejan punteros hacer diagrama de referencias.
- Para los datos que se ubiquen en el `heap` (usando `malloc`) identificar los tamaños que hagan falta (`sizeof`).
- Escribir un primer borrador de la función en `C`.
- Correr pruebas y ajustar código hasta pasar tests.

Para diagnosticar problemas con la ejecución:

- Revisar que se refieran a los encabezados necesarios (archivos `.h`) y las funciones están declaradas allí con el tipo correcto.
- Revisar que el `Makefile` esté actualizado (si no entienden alguna reglar recuerden que hay videos en la sección de **Recursos** del campus).
- Revisar que los tipos sean los correctos, por ejemplo
  `uint32_t i; for(i = 5; i >= 0; i--)printf(",");` no termina nunca. ¿Por qué motivo (piensen en los casos bordes antes y después de aplicar el decremento `--`)?
- Revisar que los tamaños de datos e inicializaciones sean correctos.
- Revisar que todos los `malloc` tengan su `free`.
- Siempre asociar el indicador de puntero a la variable y no al tipo. `uint32_t* ptr, ptr2;` es distinto de `uint32_t *ptr, *ptr2;`
- Revisar que no haya accesos fuera de rango, puede ser porque esté mal asignado el índice o porque el tipo de la declaración no es consistente con el tamaño.
  ```
  uint64_t *ptr = malloc(sizeof(uint8_t) * 10); uint32_t i;
  for(i = 0; i < 10; i++)printf("%ld", ptr[i]);
  ```

# 2 En bajo nivel (ASM)

Para escribir el código:

- Escribir un planteo informal de resolución e identificar aquellos elementos que les generen dudas.
- Escribir un pseudocódigo de la solución.
- Tener a mano el cheatsheet de convención de llamada y la lista de registros y alias por tamaño de dato.
- Escribir los registros/posición en pila de cada parámetro pasado a una función.
- Escribir prólogo y epílogo inicial completo preservando todos los registros no volátiles.
- Revisar alineación de la pila (a 16 bytes).
- Si se trabaja con estructuras identificar tamaños, tipo de alineado y ubicación de atributos, agregar `EQU` de tamaño y offset necesarios.
- Escribir pseudocódigo del cuerpo de la función.
- Identificar instrucciones a usar y leer del manual el uso de aquellas instrucciones que no manejamos.
- Identificar tamaño de los datos en los parámetros a pasar a la instrucción y revisar mnemónicos.
- Escribir boceto de código en `ASM`, revisar disponibilidad de registros, si hace falta preservar registros temporales antes de una llamada identificarlos y recordar hacer `PUSH RXX` antes de una llamada y `POP RXX` al regresar, revisar alineación de la pila en cada caso.
- Correr pruebas y ajustar código hasta pasar tests.

Para el diagnóstico:

- Revisar balanceo de pila, que cada `PUSH/SUB` tenga su `POP/ADD`.
- Revisar alineación de la pila a 16 bytes si hay llamadas a otras funciones.
- Revisar que los desplazamientos relativos a `RBP` estén bien calculados.
- Recordar que se apila `RIP` y potencialmente un conjunto de parámetros antes de hacer una llamada a función.
- Revisar que las etiquetas locales estén antepuestas con un `.` (de este modo el ensamblador sabe que se trata de una etiqueta local a la función).
- Si nos encontramos con un `operation size not specified` es porque el compilador no pudo inferir el tamaño del dato automáticamente por lo general porque hicimos un `MOV [dir], 12` sin especificar el tamaño del dato a escribir.

# 3 Interacción (C/ASM)

Para diagnosticar:

- Revisar que cada función exportada, por ejemplo `func` figure con `global func` en la sección `.text`.
- Toda función o dato de C accedido desde ASM debe ser declarado como extern, para que el ensamblador sepa que el dato existe, pero en otra unidad de código.

# 4 Alineación de datos

Para datos alineados:

- **La alineación de una estructura es la del atributo que requiere más alineación**
  Si el atributo que más alineación necesita está alineado a 4 bytes, la estructura debe ubicarse en una dirección múltiplo de cuatro
- **El tamaño de una estructura debe de ser múltiplo de su alineación**
- **Cada atributo primitivo está alineado según su tamaño**
  `char` a un byte, `short` a dos, `int` a cuatro y `long` a ocho

Para datos empaquetados (`packed`):

- **El tamaño de la estructura es la suma del tamaño de los atributos**.
- **Los atributos se encuentran contiguos en memoria**.

# 5 Convención de llamada

Para que tengan a mano en la **primera parte de la materia (64 bits)**:

| | |
|---|---|
| **No volátiles:** | `RBX, RBP, R12, R13, R14` y `R15` |
| **Valor de retorno:** | `RAX` enteros/punteros,`XMM0` flotantes |
| **Entero,puntero:** | `RDI, RSI, RDX, RCX, R8, R9`(izq. a der.) |
| **Flotantes:** | `XMM0, XMM1, ..., XMM7`(izq. a der.) |
| **¿No hay registros?** | `PUSH` a la pila(der. a izq.) |
| **Inv. de pila:** | Todo `PUSH/SUB` debe tener su `POP/ADD` |
| **Llamada func. `C`:** | pila alineada a 16 bytes (`SUB RSP, X`) |

Para que tengan a mano en la **segunda parte de la materia (32 bits)**:

| | |
|---|---|
| **No volátiles:** | `EBX, EBP, ESI` y `EDI` |
| **Valor de retorno:** | `EAX` |
| **Parámetros:** | `PUSH` a la pila(der. a izq.) |
| **Inv. de pila:** | Todo `PUSH/SUB` debe tener su `POP/ADD` |
| **Llamada func. `C`:** | pila alineada a 16 bytes (`SUB RSP, X`) |