

Algoritmos y Estructuras de Datos I - Laboratorio

Proyecto 3

Programación imperativa - Programas como transformadores de estados

1. Objetivo

El objetivo de este proyecto es introducir

- el concepto de estado y de programas como transformadores de estado;
- el modelo computacional imperativo, y sus diferencias con el modelo funcional;
- la implementación en lenguaje "C" de programas imperativos vistos en el teórico-práctico.

2. Lenguaje "C"

A lo largo de todo el proyecto, se utilizará el lenguaje C, y algunas herramientas como el GDB: The GNU Project Debugger, para ayudar a la comprensión del concepto de estado y del paradigma imperativo.

En el caso del lenguaje "C", para poder ejecutar un programa, lo vamos a tener que "compilar", y de esa manera generamos un archivo binario que podrá ser ejecutado en la computadora.

Cómo compilar en C:

Para compilar un archivo .c escribir en la terminal:

```
$> gcc -Wall -Wextra -std=c99 miarchivo.c -o miprograma
```

Para ejecutar escribir:

```
$> ./miprograma
```

Para compilar para gdb, agregar el flag -g al momento de compilar .c escribir en la terminal:

```
$> gcc -Wall -Wextra -std=c99 -g miarchivo.c -o miprograma
```

3. Ejercicios

1. **Entrada/Salida** Hace un programa en C, que solicite el ingreso de los valores de las variables x,y,z, e imprima el resultado de las siguientes expresiones. Completá los resultados de la tabla para los dos estados dados.

Expresión	(x↦7, y↦3, z↦5)	(x↦1, y↦10, z↦8)
x + y + 1	11	12
z * z + y * 45 - 15 * x	55	499
y - 2 == (x * 3 + 1) % 5	0	0
y / 2 * x	5	25
y < x * z	1	1

¿En la última expresión, que tipo tiene el resultado en lenguaje "C"?

2. **Debugging** Utilizá **GDB** o **printf** como ayuda y encontrá valores para las variables que forman el estado:

$$(x \mapsto \boxed{0}, y \mapsto \boxed{0}, z \mapsto \boxed{0}, b \mapsto \boxed{0}, w \mapsto \boxed{0})$$

de manera que las siguientes expresiones tengan el valor indicado:

Expresión	Valor
$x \% 4 == 0$	True
$x + y == 0 \ \&\& \ y - x == (-1) * z$	True
$\text{not } b \ \&\& \ w$	False

Podés cambiar el programa hecho en el ejercicio anterior, agregando las nuevas expresiones booleanas.

3. Asignaciones

Traducí al lenguaje C los programas 1.a, 1.b y 1.c del práctico que se encuentra en este [enlace](#). Esos programas están escritos en un pseudocódigo de la materia y la traducción a C no siempre es directa.

El estado σ_0 debe solicitarse al usuario utilizando el comando `scanf()`. Luego, ejecute cada programa 3 veces con diferentes valores de las variables solicitadas y escriba los valores del estado final resultante en la siguiente tabla:

programa	usuario ingresa un σ_0	produce una salida σ
1.a ejecución 1	(X->1)	(X->5)
1.a ejecución 2	(X->2)	(X->5)
1.a ejecución 3	(X->3)	(X->5)
1.b ejecución 1	(X->3, Y-> 4)	(X->7, Y-> 8)
1.b ejecución 2	(X->8, Y-> 9)	(X->17, Y-> 18)
1.b ejecución 3	(X->18, Y-> 19)	(X->37, Y-> 38)
1.c ejecución 1	(Y->4, X-> 3)	(Y->8, X-> 7)
1.c ejecución 2	(Y->9, X-> 8)	(Y->18, X-> 17)
1.c ejecución 3	(Y->19, X-> 18)	(Y->38, X-> 37)

4. Condicionales

- a) Traducí al lenguaje C los programas 1.e y 1.f de este [práctico](#).

El estado σ_0 debe solicitarse al usuario, agregando las instrucciones necesarias para que el usuario pueda ingresar los valores de las variables de entrada.

- b) Traducí a lenguaje C los programas que siguen a continuación, agregando las instrucciones necesarias para que el usuario pueda ingresar los valores de las variables de entrada. Luego, completá los estados:

$[[\sigma_0 : (x \mapsto 5, y \mapsto 4, z \mapsto 8, m \mapsto 0) \]]$

```
var x,y,z,m: Int;
if (x < y) → m := x
□ (x ≥ y) → m := y
fi
```

$[[\sigma_1 : (x \mapsto \square, y \mapsto \square, z \mapsto \square, m \mapsto \square) \]]$

```
if (m < z) → skip
□ (m ≥ z) → m := z
fi
```

$[[\sigma_2 : (x \mapsto \square, y \mapsto \square, z \mapsto \square, m \mapsto \square) \]]$

Volvé a ejecutar nuevamente con otros estados iniciales. ¿Qué hace este programa?
¿Cuál es el valor final de la variable m?

5. Ciclos

- Traducí al lenguaje C los programas 1.h y 1.i del [práctico](#). El estado σ_0 debe solicitarse al usuario, agregando las instrucciones necesarias para que el usuario pueda ingresar los valores de las variables de entrada.
- Traducí a lenguaje C los programas que siguen a continuación, agregando las instrucciones necesarias para que el usuario pueda ingresar los valores. Luego, completá los estados, donde el estado a completar es el resultado de realizar 1, 2, 3 o 4 iteraciones del ciclo. Una iteración es la ejecución completa del cuerpo del ciclo.

$[[\sigma_0: (x \mapsto 13, y \mapsto 3, i \mapsto 0) \]]$

$i := 0$

do $(x \geq y) \rightarrow$

$x := x - y$

$i := i + 1$

$[[\sigma_1^0, \sigma_1^1, \sigma_1^2, \sigma_1^3]]$

od

1)

$[[\sigma_1^0: (x \mapsto \square, y \mapsto \square, i \mapsto \square) \ , \ \sigma_1^1: (x \mapsto \square, y \mapsto \square, i \mapsto \square) \ ,$

luego de iter. 1,

luego de iter. 2

$\sigma_1^2: (x \mapsto \square, y \mapsto \square, i \mapsto \square) \ , \ \sigma_1^3: (x \mapsto \square, y \mapsto \square, i \mapsto \square) \]]$

luego de iter. 3

luego de iter. 4

2)
$$\begin{aligned}
& [[\sigma_0: (x \mapsto \boxed{5}, i \mapsto \boxed{0}, res \mapsto \boxed{False}) \]] \\
& \quad i := 2 \\
& \quad res := True \\
& \quad do(i < x \wedge res) \rightarrow \\
& \quad \quad res := res \wedge (\text{mod}(x, i) \neq 0) \\
& \quad \quad i := i + 1 \\
& \quad \quad \quad [[\sigma_1^0, \sigma_1^1, \sigma_1^2]] \\
& \quad od \\
& \quad [[\sigma_1^0: (x \mapsto \boxed{}, i \mapsto \boxed{}, res \mapsto \boxed{}) \ , \quad \sigma_1^1: (x \mapsto \boxed{}, i \mapsto \boxed{}, res \mapsto \boxed{}) \ , \\
& \quad \quad \text{luego de iter. 1} \qquad \qquad \qquad \text{luego de iter. 2} \\
& \quad \sigma_1^2: (x \mapsto \boxed{}, i \mapsto \boxed{}, res \mapsto \boxed{}) \]]: \\
& \quad \quad \text{luego de iter. 3}
\end{aligned}$$

3) Ejecutá los programas con otros estados iniciales para deducir qué hace cada uno.

6. (Funciones en C) Escribí los siguientes programas:

a) `entradas.c` que lee una variable de tipo `int` y la imprime por pantalla. En esta ocasión el programa debe utilizar dos funciones a definir (además de la función `main`):

- una función que le pide un entero al usuario y lo devuelve, con prototipo:

```
int pedir_entero(void)
```

Recordar avisarle al usuario mediante un mensaje que se espera que ingrese un valor.

- que toma un entero como parámetro y lo imprime:

```
void imprimir_entero(int x)
```

- Pensá cómo agregar un parámetro más llamado `name` de tipo `char` a las funciones anteriores y usarlo en el mensaje que se muestra al usuario. Entonces por ejemplo para la función `pedir_entero` si se ejecuta:

```
n = pedir_entero('n')
```

debería indicarse en el mensaje que se está pidiendo un valor para almacenar en la variable de nombre `n`. Notar que no hay forma de impedir que alguien use "mal" la función:

```
x = pedir_entero('n')
```

en ese caso el mensaje indicará que el valor se almacena en `n` aunque en realidad se guarda en una variable llamada `x`.

Escribí de nuevo ambas funciones con este parámetro extra (debe estar como primer parámetro de las funciones).

b) `entradas_bool.c` que lee una variable de tipo `bool` y la imprime por pantalla. El programa debe utilizar dos funciones a definir (además de la función `main`):

- una función que le pide un booleano al usuario y lo devuelve, con prototipo:

```
bool pedir_booleano(void)
```

- que toma un booleano como parámetro e imprime un mensaje "verdadero" o "falso" según sea su valor de verdad:

```
void imprimir_booleano(bool x)
```

- Agregar a las funciones un parámetro llamado `name` de tipo `char` con la misma funcionalidad descrita en el tercer ítem del apartado [6a](#)
- c) Escribí el programa del ejercicio [4b](#), pero utilizando las funciones del ejercicio anterior. ¿Qué ventajas encontras en esta nueva versión?. ¿Podrías escribir alguna otra función en ese ejercicio, cual?. ¿En qué otros ejercicios de ese Proyecto lo podrías utilizar?. Reescribí los programas donde puedas utilizarlas.
- d) En un archivo `saludos.c` implementar las siguientes funciones (además de la `main`):
- una que imprime el string "hola", que no toma ni devuelve parámetros, con prototipo:

```
void imprimir_hola(void)
```
 - similar a la anterior con el string "chau":

```
void imprimir_chau(void)
```

ese programa tiene que imprimir dos veces "hola" seguido de dos veces "chau", llamando a las dos últimas funciones desde el `main`.

Ayuda: Se debe entender cómo corre el flujo de ejecución de este programa leyendo su código fuente.