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 ejecutada 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 siguiente expresiones. Completá los resultados de la tablas 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`	7	5
`y < x *z`	1	0

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

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

$$(\texttt{x} \mapsto \boxed{4},\ \texttt{y} \mapsto \boxed{-4},\ \texttt{z} \mapsto \boxed{8},\ \texttt{b} \mapsto \boxed{T},\ \texttt{w} \mapsto \boxed{T})$$

de manera que las siguientes expresiones tengan el valor indicado:

Expresión	Valor
`x % 4 == 0`	True
`x + y == 0 && y - x == (-1) * z`	True
`not b && w`	False

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

3. **Asignaciones**

   a) 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 = 5	x = 5
1.a ejecución 3	x = 9	x = 5
1.b ejecución 1	x = 6, y = 2	x = 8, y = 4
1.b ejecución 2	x = 5, y = 6	x = 11, y = 12
1.b ejecución 3	x = 1, y = 3	x = 4, y= 6
1.c ejecución 1	x = 4, y = 5	x = 14, y= 10
1.c ejecución 2	x = 3, y = 6	x = 15, y = 12
1.c ejecución 3	x = 2, y = 1	x = 4, y = 2

   b) Utilizar la función `assert` de C para asegurar que se cumpla la precondición σ_0.

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:

$[|\sigma_0 : (\mathtt{x}\mapsto\boxed{5}, \mathtt{y}\mapsto\boxed{4}, \mathtt{z}\mapsto\boxed{8}, \mathtt{m}\mapsto\boxed{0}) \ |]$

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

$[|\sigma_1 : (\mathtt{x}\mapsto\square, \mathtt{y}\mapsto\square, \mathtt{z}\mapsto\square, \mathtt{m}\mapsto\square) \ |]$

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

$[|\sigma_2 : (\mathtt{x}\mapsto\square, \mathtt{y}\mapsto\square, \mathtt{z}\mapsto\square, \mathtt{m}\mapsto\square) \ |]$

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

5. **Ciclos**

a) 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.

b) 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.

1)

$[|\sigma_0: (\mathtt{x}\mapsto\boxed{13}, \mathtt{y}\mapsto\boxed{3}, \mathtt{i}\mapsto\boxed{0}) \ |]$

$i := 0$
`do` $(x \geq y) \rightarrow$
$x := x - y$
$i := i + 1$
$[|\sigma_1^0, \sigma_1^1, \sigma_1^2, \sigma_1^3|]$
`od`

$[|\sigma_1^0: (\mathtt{x}\mapsto\square, \mathtt{y}\mapsto\square, \mathtt{i}\mapsto\square)$, **luego de iter. 1**,
$\sigma_1^1: (\mathtt{x}\mapsto\square, \mathtt{y}\mapsto\square, \mathtt{i}\mapsto\square)$, **luego de iter. 2**
$\sigma_1^2: (\mathtt{x}\mapsto\square, \mathtt{y}\mapsto\square, \mathtt{i}\mapsto\square)$, **luego de iter. 3**
$\sigma_1^3: (\mathtt{x}\mapsto\square, \mathtt{y}\mapsto\square, \mathtt{i}\mapsto\square) \ |]$ **luego de iter. 4**

2)

$[|\sigma_0$: (x↦5, i↦0, res↦False) $|]$
$i := 2$
$res := True$
do $(i < x \wedge res) \rightarrow$
$res := res \wedge (\mathtt{mod}(x, i) \neq 0)$
$i := i + 1$
$[|\sigma_1^0, \sigma_1^1, \sigma_1^2|]$
od

$[|\sigma_1^0$: (x↦□, i↦□, res↦□) , σ_1^1: (x↦□, i↦□, res↦□) ,
luego de iter. 1 **luego de iter. 2**
σ_1^2: (x↦□, i↦□, res↦□) $|]$:
luego de iter. 3

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 pedirEntero(void)
```

- que toma un entero como parámetro y lo imprime:

```
void imprimeEntero(int x)
```

b) Escribí el programa del ejercicio 4*b*, pero utilizando las funciones del ejercicio anterior. ¿Qué ventajas enconarás en esta nueva versión?. ¿Podría escribir alguna otra función en ese ejercicio, cual?. ¿En qué otros ejercicios de ese Proyecto lo podrías utilizar?. Reescribí los programas donde puedas utilizarlas.

c) en un archivo `saludos.c` implementar las funciones siguientes (Además de la `main`):

- una que imprime el string `"hola"`, que no toma ni devuelve parámetros, con prototipo:

```
void imprimeHola(void)
```

- similar a la anterior con la el string `"chau"`:

```
void imprimeChau(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 como corre el flujo de ejecución de este programa leyendo su código fuente.