TP CONCEPTUAL NRO 1

Objetivo del TP:

Desarrollo del Teorema del programa estructurado propuesto por Corrado Böhm y Giuseppe Jacopini. Identificar y ejemplificar a partir de enunciados desarrollados por usted mismo las tres estructuras propuestas por los autores.

El trabajo práctico constará de 3 partes:

1) Definiciones teóricas de los autores sobre las tres estructuras de la programación estructurada (Secuencia, selección e iteración).
2) Un enunciado desarrollado por usted mismo (Problema similar a los que utilizamos para practicar los ejercicios) para cada tipo de estructura identificada en el punto 1.
3) Diagramación lógica en diagrama conceptual y pseudocódigo (Puede utilizar Pseint) de cada solución del punto 2 y el código en C desarrollado para solucionar cada enunciado.

Aclaraciones:

- Puede desarrollar un solo enunciado donde tenga que aplicar las tres estructuras juntas o bien cada una por separado.
- El TP se entrega en formato Word o pdf con los detalles teóricos, diagramas y código (Formato imagen) en el mismo Word o pdf.

Alumno: Gerardo Tordoya

Docente: Gastón Matías Weingand

Fecha entrega: 01/10/2020

DESARROLLO

Febrero tiene 28 o 29 días (este último solo en los años bisiestos). Sin embargo, dos veces a lo largo de la historia, y solo en determinados países, ha habido un "**30 de febrero**".

La primera vez fue en Suecia en 1712. En lugar de cambiar del calendario juliano al calendario gregoriano omitiendo un bloque de días consecutivos (como se había hecho en otros países) el Imperio sueco planeó cambiar gradualmente omitiendo todos los días bisiestos desde 1700 hasta 1740, inclusive. Aunque el día bisiesto se omitió en febrero de 1700, la Gran Guerra del Norte comenzó más tarde ese año, desviando la atención de los suecos de su calendario para que no omitieran los días bisiestos en las dos siguientes ocasiones: 1704 y 1708 siguieron siendo años bisiestos.

Para evitar confusiones y más errores, el calendario juliano se restauró en 1712 agregando un día bisiesto adicional, lo que le dio a ese año el único uso real conocido del 30 de febrero en un calendario.

Ahora bien. Supongamos que contamos con un programa que calcula el día del año y que (por la razón que fuere) tuviera que ser modificado para contemplar una singularidad así. ¿Cuál preferirías modificar?

¿Éste?

```
void structured_code(int year, int date)
{
    int months[2][12] = {{31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
                         {31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}};
    int leap, days, month;
    if ((year % 400 == 0) | ((year % 100) != 0 && (year % 4) == 0))
        leap = 1;
    else
        leap = 0;
    if (year < 1753 || year > 3999 || date <= 0 || date > 365 + leap)
        printf("bad year, date\n");
    else
    {
        days = 0;
        month = 0;
        while (date > days + months[leap][month])
        {
            days = days + months[leap][month];
            month = month + 1;
        }
    }
    printf("\nday = %d/month = %d", date - days, month + 1);
}
```

¿O éste?

```
int spaghetti_code(int year, int date)
{
    int day, month;
    int L;
    if (date < 1 || date > 366 || year < 0)
        return 0;
    if (date <= 31)
        goto JAN;
    L = 1;
    if (year % 400 == 0)
        goto LEAP;
    if (year % 100 == 0)
        goto NOLEAP;
    if (year % 4 == 0)
        goto LEAP;
NOLEAP:
    L = 0;
    if (date > 365)
        return 0;
LEAP:
    if (date > (181 + L))
        goto G181;
    if (date > (90 + L))
        goto G90;
    if (date > (59 + L))
        goto G59;
    month = 2;
    day = date - 31;
    goto OUT;
G59:
    month = 3;
    day = date - (59 + L);
    goto OUT;
G90:
    if (date > 120 + L)
        goto G120;
    month = 4;
    day = date - (90 + L);
    goto OUT;
G120:
    if (date > 151 + L)
        goto G151;
    month = 5;
    day = date - (120 + L);
    goto OUT;
G151:
    month = 6;
    day = date - (151 + L);
    goto OUT;
G181:
```

```
    if (date > 273 + L)
        goto G273;
    if (date > 243 + L)
        goto G243;
    if (date > 212 + L)
        goto G212;
    month = 7;
    day = date - (181 + L);
    goto OUT;
G212:
    month = 8;
    day = date - (212 + L);
    goto OUT;
G243:
    month = 9;
    day = date - (243 + L);
    goto OUT;
G273:
    if (date > 334 + L)
        goto G334;
    if (date > 304 + L)
        goto G304;
    month = 10;
    day = date - (273 + L);
    goto OUT;
G304:
    month = 11;
    day = date - (304 + L);
    goto OUT;
G334:
    month = 12;
    day = date - (334 + L);
    goto OUT;
JAN:
    month = 1;
    day = date;
    goto OUT;
OUT:
    printf("day=%d/month=%d", day, month);
    return 0;
}
```

Ésta es la adaptación de un ejemplo de la vida real de un programa que se usa en una entidad financiera. En ambos casos, hace exactamente lo mismo: dado un número del año, calcula su fecha. Ejemplo: el día 255 del 2020 cae en 11 de septiembre.

Originalmente, fue codificado en estilo espagueti (2do ejemplo). Luego, fue refactorizado en estilo estructurado (1er ejemplo).

El teorema de la programación estructurada de Böhm–Jacopini

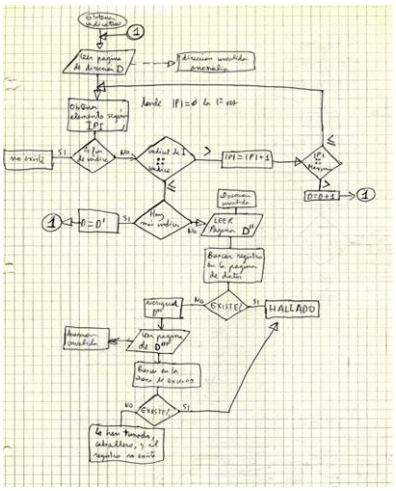
El relato del libro "Memorias de un viejo informático" es más que claro:

«

Cobol es un lenguaje realmente "verbose": seguramente para hacer las mismas tonterías de las sentencias anteriores[1]*, en C (con o sin los signos de la suma) o en Java se requerirían quizá tres líneas cortitas en total… pero esa "verbosidad" redunda en un buen entendimiento de lo que hace un programa Cobol cuando lo mira un programador distinto de aquél que lo escribió… veinte años después. Y eso, en una instalación informática de verdad, con decenas de miles de programas, y millones de líneas de código en Producción, cotiza muy, pero que muy caro.*

Ya sabéis que en los cincuentas, sesentas y primeros setentas, los informáticos de la época (en muchos casos, ingenieros metidos a informáticos), programaban sus rutinas como buenamente sabían y podían. Fueron inventando las diversas maneras de programar conforme programaban, con lo que los más avispados (o afortunados, quién sabe) fueron encontrando maneras de solucionar problemas diversos sobre la marcha.

Pero, en definitiva, la forma aceptada de programar era, mayormente, a la "mecagüendiez", traducción castiza del no menos castizo "cada maestrillo tiene su librillo".



En una palabra, en los primeros setenta, ésa era toda la cera que ardía…

Claro que, si nos damos cuenta, el problema no era todavía excesivamente importante: Había pocos ordenadores funcionando, y no había muchos programas en Producción, y, sobre todo, éstos eran muy pequeños. No había otro remedio, con 8, 16 ó 32Kb de memoria, no se podían hacer programas muy complejos, simplemente porque no cabían.

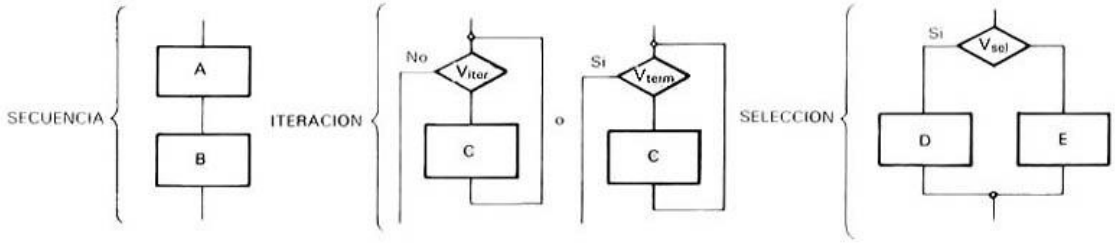
…Y sin embargo, alguien estaba ya dándose cuenta de que esa no era la manera correcta de programar de cara al futuro. Cuando el uso de las máquinas se fuera generalizando, éstas fueran cada vez más y más potentes, y más y más

[1] Se refiere a una codificación como la del segundo ejemplo.

programadores se fueran incorporando a la profesión, los programas spaghetti no serían adecuados, debido no sólo a la dificultad de diseñarlos, programarlos y probarlos, sino sobre todo, de mantenerlos[2].

Conforme más y más aplicaciones se ponían en marcha, más y más aplicaciones había que mantener, modificar, cambiar… y la programación "personal" constituía un problema serio, sobre todo cuando el autor original del engendro, eeeh, digo, del programa, ya no estaba en la empresa para desentrañar lo que ahí dentro había (…y a veces, ni así).

*En 1966, **Bohm y Jacopini** publicaron un artículo en el que demostraron formalmente que, usando exclusivamente tres estructuras de control era posible programar cualquier función computable. Las tres estructuras eran, claro está, la secuencial, la repetitiva y la alternativa. Combinándolas recursivamente es factible realizar cualquier programa.*



***Una Secuencia** consiste en una serie de elementos que ocurren secuencialmente, uno detrás de otro, y siempre en el mismo orden. Esta serie de elementos podría estar vacía, es decir, no contener elemento alguno.*

***Una Repetición** (o iteración) consiste en un único elemento que se repite una y otra vez, o ninguna, hasta que cierta condición sea satisfecha. O lo que es lo mismo, de 0 a n veces, mientras se satisfaga una cierta condición.*

***Una Alternativa** (o selección) consiste en dos elementos, de los que sólo ocurre uno u otro, dependiendo de cierta condición. Aunque en principio se definen sólo dos elementos para definir la selección, no hay ningún problema en permitir más de dos ramas, siempre que sólo ocurra una de ellas, dependiendo siempre de dicha condición (en realidad es el resultado de des-anidar varias selecciones anidadas).*

*No obstante su importancia para la ciencia de la computación, el artículo de Bohm y Jacopini fue poco menos que ignorado, ya que ellos se limitaron a demostrar matemáticamente que esto era así (lo que hoy en día se conoce como **Teorema de Bohm y Jacopini**), pero no rompieron lanza alguna para defender que programar en base al uso de las tres estructuras fuera bueno y recomendable. Y todo el mundo siguió a su bola…*

»

Esos fueron los inicios, y el ejemplo que di, cuaja perfectamente en lo que este autor relata. Obviamente, la historia sigue, pero solo quería dejar en claro los puntos que rodean al nacimiento de la primera forma de ordenamiento de código. El teorema data de 1966, pero el auge estructurado tuvo que esperar hasta los '80 (es decir, dentro de esa historia, hace falta el desarrollo gradual merced al aporte, más o menos uno detrás de otro, de teóricos como Edsger W. Dijkstra, Jean Dominique Warnier, Ken Orr, Marie Thérèse Bertini, para llegar finalmente, en 1975, a Michael A. Jackson, considerado el máximo teórico de la estructurada y que, luego de él, no hubo nadie más que siguiera haciendo aportes teóricos).

[2] Es decir: actualizarlos, modificarlos, etc.

Solo una cosa más: al comparar ambos ejemplos, ¿no les maravilla que, merced a una estructuración del código, conseguir lo mismo con un poco de abstracción y muchas menos líneas de código? A mí, sinceramente, me resulta asombroso.

PSeInt

Usemos el código propuesto como ejemplo (el 1er ejemplo, el estructurado) para completar la consigna del diagrama conceptual (diagrama de flujo, etc.) a partir del seudocódigo escrito en PSeInt.

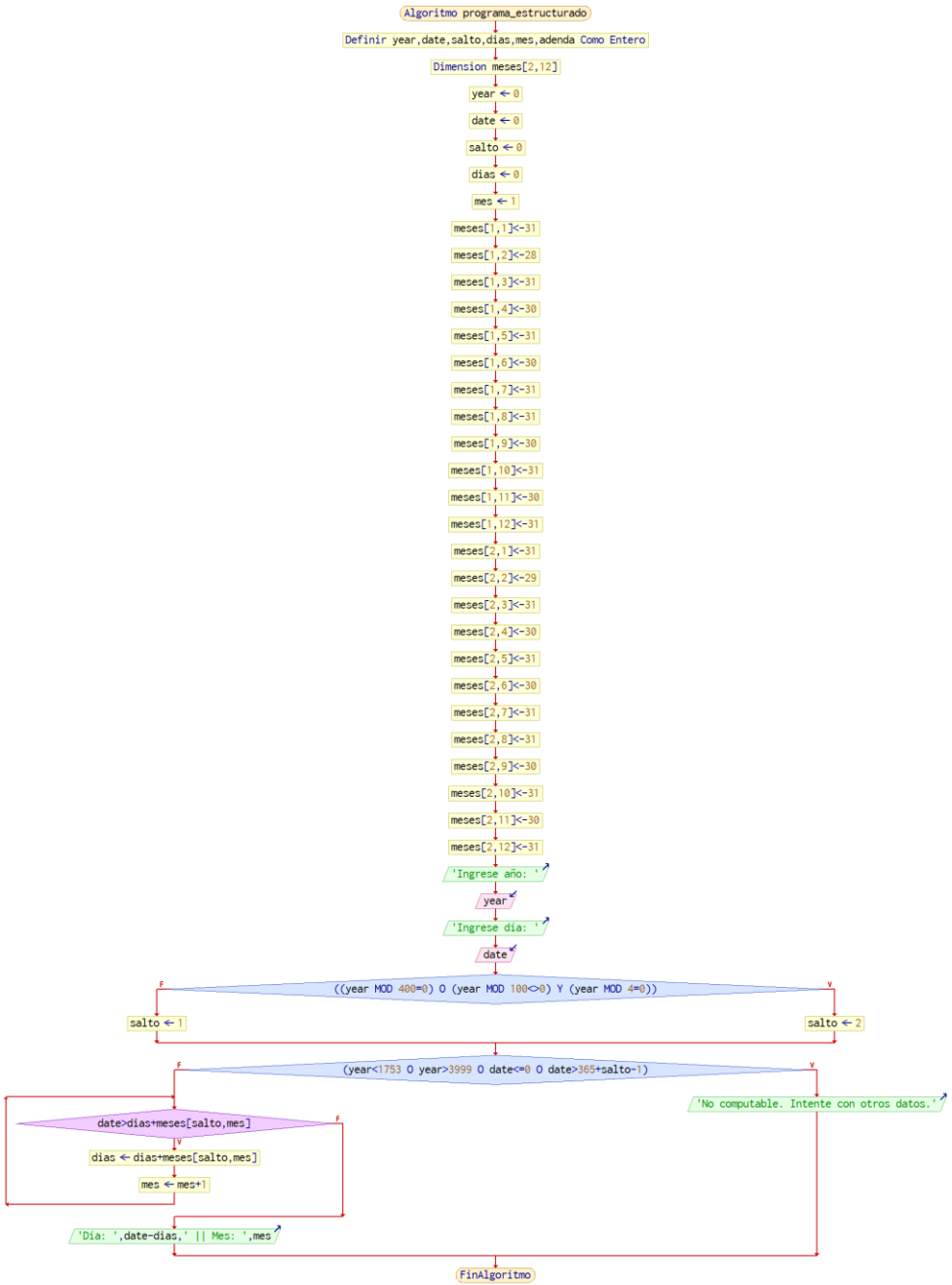
En primer lugar, el seudocódigo:

```
// UAI
// Programación Estructurada
// -------------------------
// TP 1 OBLIGATORIO
// -------------------------
// GERARDO TORDOYA
// sep-2020
// ------------------------

////////////////////////////////////////////////////////////////////////////////

// Este programa, dado un año y un número de día,
// calcula en qué fecha cae.
//  EJEMPLO (si fuera una función):
//    function(2020, 255);
//    (retornará 11/sep/2020 como respuesta)
//
// @param {*} year Año a considerar.
// @param {*} date Día del año del que se desea saber la fecha.

// NOTA:
// El tipo de estructura de control usada
// se señala entre paréntesis como comentarios.

Algoritmo programa_estructurado

      // VARIABLES
      Definir year, date, salto, dias, mes, adenda Como Entero;
      Dimension meses[2, 12];

      // INICIALIZAR
      year = 0;
      date = 0;
      salto = 0;
      dias = 0;
      mes = 1;

      // Inicializar vector.
```

```
	// (en PSeInt no hay una forma económica de hacerlo).
	// (Secuencia)
	meses[1, 1] <- 31;
	meses[1, 2] <- 28;
	meses[1, 3] <- 31;
	meses[1, 4] <- 30;
	meses[1, 5] <- 31;
	meses[1, 6] <- 30;
	meses[1, 7] <- 31;
	meses[1, 8] <- 31;
	meses[1, 9] <- 30;
	meses[1, 10] <- 31;
	meses[1, 11] <- 30;
	meses[1, 12] <- 31;

	meses[2, 1] <- 31;
	meses[2, 2] <- 29;
	meses[2, 3] <- 31;
	meses[2, 4] <- 30;
	meses[2, 5] <- 31;
	meses[2, 6] <- 30;
	meses[2, 7] <- 31;
	meses[2, 8] <- 31;
	meses[2, 9] <- 30;
	meses[2, 10] <- 31;
	meses[2, 11] <- 30;
	meses[2, 12] <- 31;

	// INPUT
	// (Secuencia)
	Escribir Sin Saltar "Ingrese año: ";
	Leer year;
	Escribir Sin Saltar "Ingrese día: ";
	Leer date;

	// DETERMINAR SET
	// (Alternativa/Selección)
	Si ( (year Mod 400 = 0) O (year Mod 100 <> 0) Y (year Mod 4 = 0) ) Entonces
		salto <- 2;
	SiNo
		salto <- 1;
	FinSi

	// VALIDAR RANGOS COMPUTABLES.
	// (Alternativa/Selección)
	Si (year < 1753 O year > 3999 O date <= 0 O date > 365 + salto - 1) Entonces
		Escribir "No computable. Intente con otros datos.";
	SiNo
		// (Repetición/Iteración)
		Mientras date > dias + meses[salto, mes] Hacer
			dias <- dias + meses[salto, mes];
			mes <- mes + 1;
		FinMientras

		// MOSTRAR RESULTADO EN PANTALLA (OUTPUT)
```

```
            Escribir "Día: ", date - dias, " || Mes: ", mes;

      FinSi

FinAlgoritmo
```

Y finalmente, el diagrama resultante:



Finalmente, un par de capturas de pantalla para ver las salidas por pantalla: