Programación 1
Conceptos generales sobre las variables y la memoria

En matemática, a un objeto o expresión que representa un valor de un determinado tipo le asignamos un nombre que llamamos variable. Las variables tienen pues un valor de un tipo (por ejemplo, si definimos x = 3, el nombre x queda asociado al valor 3 de tipo numérico y, en el contexto de esa definición, son expresiones equivalentes x < 2 y 3 < 2).

En computación, un objeto *tiene algo más*: una representación interna (la secuencia de ceros y unos que representa un valor numérico en representación binaria) y un lugar en la memoria del computador. Un objeto en computación *(data object)* es un contenedor para valores, es decir, un lugar en el computador donde los valores pueden ser depositados y recuperados. Un objeto está caracterizado por un conjunto de atributos, el más importante de los cuales es su tipo. Los atributos determinan la cantidad y el tipo de valores que un objeto puede contener y también determinan la organización de esos valores. Un valor puede ser un carácter, un número, etc. y se representa por una secuencia particular de ceros y unos en la memoria del computador. Pueden existir muchas copias del mismo valor en la memoria, esto significa que la misma secuencia de ceros y unos se repetirá.

Por ejemplo, en la figura se ilustra, a la izquierda, las instrucciones del texto del programa y, a la derecha, el contenedor de valores *(data object)*, cuya dirección de memoria es 23AC4001. Al momento de la declaración de la variable n de tipo entero, el contenedor queda determinado para guardar un valor de ese tipo. Al momento de la asignación n:=17, el valor 17 de la variable n se guarda como secuencia de ceros y unos en el contenedor.

Una de las operaciones que altera el valor de un *data object* es la asignación: por ejemplo en el caso de arriba, luego de la instrucción n := n + 1 el valor del contenedor de valores 23AC4001 será 18 (10010 en representación binaria).

En el curso de la ejecución de un programa, algunos objetos existen desde el inicio, otros son creados dinámicamente durante la ejecución, otros son destruidos, otros persisten hasta el final de la ejecución. O sea que un objeto tiene un *tiempo de vida* durante la ejecución y mientras dure, puede

ser usado para almacenar valores. Un objeto puede ser simple como 23AC4001 arriba o puede ser estructurado, por ejemplo, una secuencia de celdas de memoria para almacenar una lista de valores.

A diferencia de la matemática donde, en determinado contexto, un objeto está asociado a un valor y un tipo, en computación un objeto puede tener *asociaciones* a diferentes atributos: por ejemplo, el objeto arriba, está asociado a un lugar en la memoria (23AC4001), un nombre (n), un valor en el texto del programa (17), su valor binario interno (10001), un tipo (numérico). A su vez, otras asociaciones pueden ser creadas durante la ejecución, por ejemplo, si usamos alguno de los atributos como argumentos (o parámetros) de un sub programa (o programa).

Memoria

Los datos en la memoria se representan en codificación binaria (secuencia de 0 y 1), donde 8 bits constituyen un byte. La memoria se considera como una tabla indizada cuyas celdas son datos de tamaño de un byte (8 bits), que se acceden a través del índice correspondiente. En el ejemplo arriba, si el programa necesita el dato 17, accede al mismo a través del índice 23AC4001 de la tabla memoria. Un dato de un programa puede ser por ejemplo una secuencia de datos, en cuyo caso ocupa varias celdas en la tabla (la memoria). En la memoria no solo se almacenan los datos, sino también *el programa*, que especifica las operaciones a realizar con los datos.

¿Quién realiza las operaciones?

En un computador, además de la memoria, exite una Unidad Central de Proceso (CPU por sus siglas en inglés) que es la que se encarga de ejecutar las operaciones.

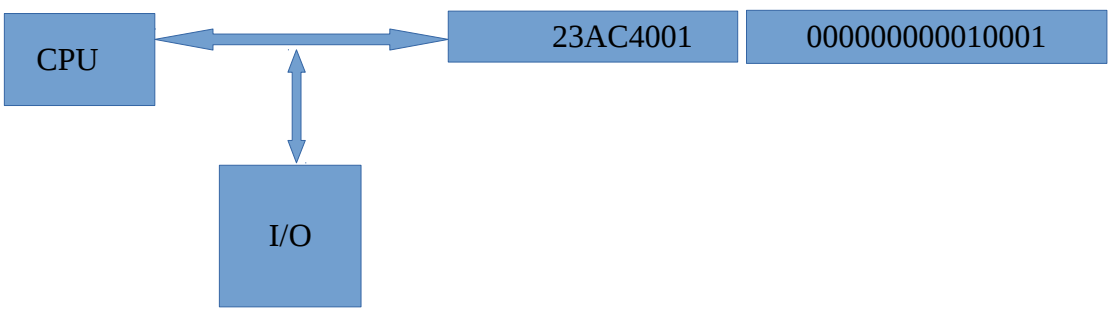
¿Cómo sabemos lo que hace el computador?
Si tenemos un programa como este:

```
program factorial;
var i,  n, fact : integer;
begin
    read (n); fact := 1;
    for i := 2 to n do
       fact := fact * i;
end.
```

Si lo ejecutamos, ¿cómo sabemos el resultado?

Existe también un subsistema de **Entrada/Salida** (**I/O** por sus siglas en inglés) que realiza la comunicación con el programador, el usuario, u otros dispositivos externos, a través de un BUS, y un protocolo que especifica un conjunto de mensajes que son enviados a través del mismo. Las sentencias read y write operan en dicho subsistema.

Memoria con el valor 17 en la celda 23AC4001



Pascal y otros lenguajes

¿Por qué tiene Pascal la parte declarativa?

El resultado de cada expresión tiene un tipo determinado, independiente de su valor, y que puede conocerse aún ignorando los valores de sus elementos componentes, teniendo en cuenta solo sus tipos. Así por ejemplo, pueden existir expresiones numéricas, o bien enteras (como 2 + 3) o reales (como 3.14 * Sqr(2.5)), booleanas (como 2 + 2 = 5) y de caracteres (como Succ('a')).

La información sobre el tipo de una expresión (y el de sus subexpresiones componentes) juega en Pascal un importante papel: una vez escrito un programa, el compilador comprueba que las expresiones que intervienen en él son correctas sintácticamente y que los tipos de las componentes de las mismas son consistentes. Por ejemplo, para una parte declarativa como sigue:

```
Var x, y : Integer;
      z : Real;
      c : Char;
      b : Boolean;
```

se puede verificar el tipo de una expresión como

```
((x + y) * z < Ord (c)) or b
```

de la siguiente manera:

```
( (Z + Z) * R < Ord ( C ) ) or B
( Z * R < Z ) or B
( R < Z ) or B
B or B
B
```

si se asignan los siguientes valores

```
x := 6;
y := 8;
z := 3.14;
c := 'a';
b := True
```

el compilador acepta la sentencia

```
if ((x + y) * z < Ord (c)) or b then …
```

y rechaza en cambio esta otra:

```
if 2 = ((x + y) * z < Ord (c)) or b
```

Variables y constantes

```
Const Pi = 3.14
```

Pi
3.14

```
Var base, altura : real;
```

base
? (real)

altura
? (real)

La declaración de constantes permite asignar un nombre a una constante y referir a su valor a través del mismo. Las variables son objetos cuyo valor no se conoce a priori y puede cambiar a lo largo del programa. En cambio, el tipo de las variables permanece inalterable desde que se establece al principio del programa; los compiladores de Pascal utilizan esta información para determinar la cantidad de espacio reservado en la memoria para cada objeto y la forma en que se hará la representación, así como realizar las verificaciones de tipo.

En algunos compiladores, una variable declarada presenta un valor indefinido al iniciarse el programa; en efecto, se trata de un valor “basura", representado por el contenido de la memoria reservado cuando se declaró la variable. Lógicamente, un programa que depende de valores indefinidos tiene un comportamiento indeterminado; por ello es necesario evitar el operar con tales variables, asignándoles valores iniciales. Algunos compiladores asignan valores por defecto a las variables (para las variables globales, free pascal pone 0, si son numéricas, FALSE si son booleanas, si son Char se asigna un carácter x tal que ord(x) es 0). No es correcto programar asumiendo que el compilador va a asignar valores por defecto. El programador debe controlar siempre el valor de las variables.

Ejercicios

1) Complete la tabla con el efecto de cada instrucción (0, 1, 2, 3,4) sobre a y b.

```
    0. var a, b : Integer;
begin
    1.    a := 6;
    2.    b := 7;
    3.    a := b;
    4.    b := a
end;
```

	a	b
0		
1		
2		
3		
4		

2) Completar los cuadros de la tabla con varios valores de a y b, ingresados con la sentencia 1.

```
var a, b : Integer;
begin
      1.     read (a,b)
      2.     a:= a + b;
      3.     b:= a – b;
      4.     a:= a – b
end;
```

	a	b
1		
2		
3		
4		

3) Sean las variables m y n de tipo integer. Coloque los paréntesis necesarios para que la siguiente expresión sea correcta.

```
m < n or m = n
```

¿Puede ser correcta si m y n son de algún otro tipo y parentizamos de otro modo?