TEMA 10

REPRESENTACIÓN INTERNA DE LOS DATOS

INDICE:

Realizado por Cayetano Borja Carrillo

Tiempo de escritura: 2 horas y 15 minutos

1. INTRODUCCIÓN

En informática, un dato es una unidad de información u objeto manipulable por un ordenador. Los datos pueden ser de tipo primitivo si almacenan un único valor simple o estructurado si se representan mediante una estructura de datos que puede contener varios valores.

Un ordenador digital funciona con impulsos eléctricos y solo entiende 2 estados: cuando pasa corriente eléctrica y cuando no. Por este motivo, la única forma que tiene un ordenador para representar internamente los datos es utilizando un código que empleé 2 estados, como es el código binario.

En este tema se desarrollan los principales sistemas de codificación para representar internamente los datos. Se trata de un tema de gran importancia en el estudio de la programación ya que, conocer estos conceptos, permite al desarrollador elegir el tipo de datos que mejor se adapte a sus necesidades.

2. DATOS PRIMITIVOS

Los datos primitivos son los más simples y básicos que existen. La cantidad de tipos de datos que se pueden usar depende del lenguaje de programación, siendo los más comunes los siguientes:

2.1. Entero

Un dato de tipo entero es aquel que almacena valores numéricos sin decimales. La cantidad de valores diferentes que puede representar un dato depende de la cantidad de memoria que se le reserva. Por ejemplo, si se le reservan 8 bits, puede representar $2^8 = 256$ valores distintos.

Algunos sistemas que permiten codificar datos de tipo entero son los siguientes:

BCD

BCD permite codificar números naturales, es decir, enteros sin signo. Para convertir un número decimal a BCD, se transforma cada cifra decimal a su equivalente binario empleando 4 bits.

Ejemplo: Codificar $25_{(10}$ a BCD.

$25_{(10} = \underbrace{0010}_{2}\underbrace{0101}_{5}{}_{(2}$

Signo y magnitud

Este sistema emplea 1 bit para representar el signo, donde 0 denota un número positivo y 1 un negativo, y “n-1” bits para representar la magnitud, que es el valor absoluto del número en binario natural.

Ejemplo: Codificar $-25_{(10}$ a signo y magnitud empleando 8 bits.

Signo (1 bit) → Negativo = 1

Magnitud (7 bits) → $25_{(10} = 0011001_{(2}$

$-25_{(10} = 10011001_{(2}$

Complemento a 1 (Ca1)

Para codificar un número a Ca1, hay que diferenciar los números positivos de los negativos. Los positivos se representan en binario natural, mientras que los negativos se representan invirtiendo los dígitos (los ceros por unos y los unos por ceros).

Ejemplo: Codificar $25_{(10}$ y $-25_{(10}$ a Ca1 de 8 bits.

$25_{(10} = 00011001_{(2}$

$-25_{(10} = 11100110_{(2}$

Complemento a 2 (Ca2)

En Ca2, al igual que ocurre con Ca1, los números positivos se representan en binario natural, pero los negativos se codifican sumando 1 al resultado obtenido tras invertir los dígitos.

Ejemplo: Codificar $25_{(10}$ y $-25_{(10}$ a Ca2 de 8 bits.

$25_{(10} = 00011001_{(2}$

$-25_{(10} = 11100110 + 1 = 11100111_{(2}$

2.2. Real

Un dato de tipo real, también llamado flotante, es aquel que almacena valores numéricos con decimales. Como algunos números como π (pi) no se pueden representar de forma exacta, se codifica una aproximación. Cuanto mayor sea la cantidad de memoria que se le reserva al dato, mayor será la precisión con la que se representa.

Existen varios sistemas para codificar números reales, siendo el más extendido el coma flotante con normalización IEEE754, que puede ser de precisión simple o doble. Antes de describir este sistema, se va a definir notación científica ya que es así como se representan los números bajo esta norma.

Cualquier número "N" se puede expresar en notación científica o coma flotante de forma: "$N = m * b^e$", donde "m" es la mantisa, "b" la base y "e" el exponente. Dependiendo de la posición de la coma, un mismo número puede tener varias representaciones. Por ejemplo, algunas representaciones del número $958_{(10}$ son:

$958_{(10} = 95{,}8 \times 10^1$
$958_{(10} = 9{,}58 \times 10^2$
$958_{(10} = 0{,}958 \times 10^3$

IEEE754 de precisión simple

Este estándar utiliza 32 bits para codificar un número, donde 1 bit se reserva al signo (0 positivo y 1 negativo), 8 para el exponente y 23 para la mantisa.

Para convertir un número a este sistema, primero se pasa el valor absoluto del número a binario y luego a notación científica, dejando un solo dígito (un 1) en la parte entera y siendo la parte fraccionaria la mantisa.

Después se suma al exponente el valor "$2^{(n-1)}$-1", donde "n" es el número de bits que se reservan al exponente. Como se usan 8 bits, se le suma 2^{8-1}-1 = 127.

Finalmente, se pasa el exponente a binario y se colocan el signo, el exponente y la mantisa en ese orden.

Ejemplo: Codificar $-25_{(10}$ a IEEE754 de precisión simple.

1º - Se pasa $25_{(10}$ a binario:

$25_{(10} = 11001_{(2}$

2º - Se pasa a notación científica:

$25_{(10} = 1{,}1001 * 2^4$

3º - Se suma 127 al exponente y se pasa a binario:

$e = 4 + 127 = 131_{(10} \rightarrow 10000011_{(2}$

4º - Se colocan el signo, el exponente y la mantisa en ese orden.

$-25_{(10} = \underbrace{1}_{\text{Signo}}\underbrace{10000011}_{\text{Exponente}}\underbrace{10010000000000000000000}_{\text{Mantisa}}{}_{(2}$

IEEE754 de precisión doble

Para codificar un número a este sistema de 64 bits, se realizan las mismas operaciones que con IEEE754 de precisión simple, pero reservando 1 bit al signo, 11 al exponente y 52 a la mantisa.

2.3. Carácter

Un dato de tipo carácter es aquel que almacena un símbolo lingüístico como una letra o un signo de puntuación. Algunos lenguajes de programación como C utilizan una secuencia de caracteres concatenados para formar el tipo de datos cadena.

Algunos de los sistemas más utilizados para codificar caracteres son los siguientes:

ASCII

ASCII utiliza 8 bits para representar cada carácter, donde 7 son para el carácter en sí y el octavo, llamado bit de paridad, se utiliza para detectar errores.

Que se usen 7 bits significa que se pueden representar hasta $2^7 = 128$ caracteres distintos. Esta limitación provoca que solo estén registrados los caracteres del alfabeto inglés, por lo que los símbolos “ñ” o “¿” no tienen representación.

Para codificar un carácter a ASCII no existe ninguna regla de conversión, simplemente cada carácter tiene un valor establecido. Por ejemplo, el carácter “A” corresponde con el carácter número 65, así que su representación interna es:

$A = 65_{(10} = 01000001_{(2}$

Unicode

Unicode fue diseñado con el objetivo de poder representar todos los caracteres existentes, como los caracteres del alfabeto latino, chino, ruso e, incluso, emojis. Para conseguirlo, Unicode utiliza 16 bits para codificar un carácter. Que se usen 16 bits permite representar hasta $2^{16} = 65536$ caracteres distintos.

Los primeros caracteres coinciden con los de cualquier código de 8 bits, como ASCII, para evitar incompatibilidades.

2.4. Booleano

Un dato de tipo booleano o lógico es aquel que toma como valor uno de 2 valores posibles: verdadero y falso. Se utiliza para para representar alternativas a ciertas condiciones como, por ejemplo, ¿la variable “X” es mayor que la variable “Y”?

La representación interna un dato booleano depende del lenguaje de programación utilizado. Por ejemplo, en C++ se usa un byte de ceros (8 ceros) para representar el valor “Falso” y cualquier otro valor para el “Verdadero”. Ejemplo:

$X = 00000000_{(2}$ → Falso $X = 00000001_{(2}$ → Verdadero

2.5. Puntero

Un dato de tipo puntero es aquel que toma como valor la dirección de memoria RAM donde se encuentra almacenado otro dato. Supongamos que una variable “X” de tipo entero se encuentra almacenada en la dirección de memoria 0x10000111. Si otra variable llamada “Y” de tipo puntero apunta a “X”, el valor de “Y” será $10000111_{(2}$.

La cantidad de bits que usa un puntero para su representación interna depende de la arquitectura para la que ha desarrollado el software, siendo lo habitual 64 bits.

3. DATOS ESTRUCTURADOS

Los datos estructurados son aquellos que se representan mediante una estructura de datos, que es una forma de organizar una colección de datos con el fin de facilitar su acceso y manipulación.

Las estructuras de datos se pueden clasificar en estáticas o dinámicas, dependiendo de si el tamaño de memoria que ocupan es fijo o variable.

3.1. Estructuras estáticas

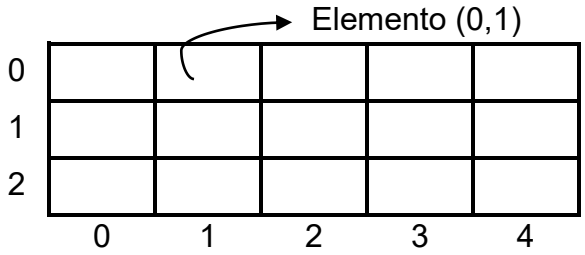
Las estructuras estáticas son aquellas donde el tamaño de memoria que ocupan se define durante el desarrollo del *software* y no cambia durante la ejecución del programa.

Las principales estructuras estáticas son las siguientes:

3.1.1. Array

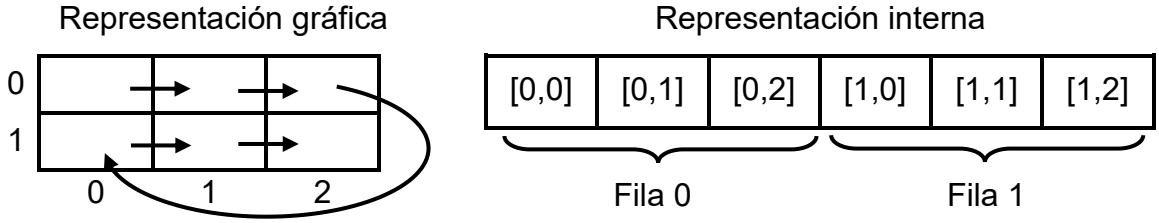
Un *array* es una estructura de datos compuesta por un conjunto determinado de elementos del mismo tipo (todos enteros, todos booleanos, etc.). Cada elemento del *array* tiene asociada una dirección única llamada índice, que determina su posición en el *array*.

Un *array* puede tener varias dimensiones, llamándose vector al *array* de una dimensión y matriz al de varias. Las dimensiones de un *array* determinan el número de índices que se utilizan para acceder a cada elemento. Por ejemplo, en un *array* de 2 dimensiones, cada celda es accedida mediante 2 índices: uno para indicar la fila y otro para la columna.

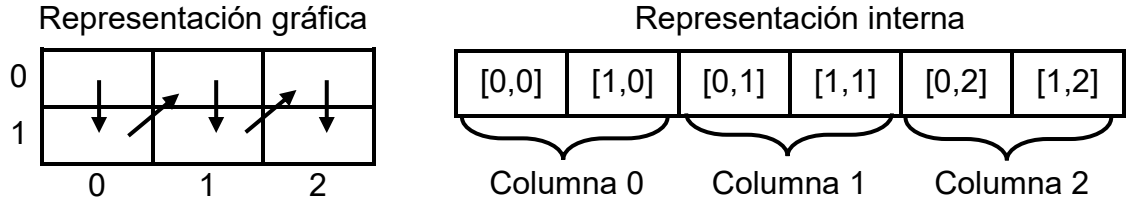


Para su representación interna, los elementos del *array* se almacenan de manera lineal y secuencial, independientemente de sus dimensiones. La organización lógica puede realizarse de 2 formas:

- Por orden de fila mayor: Los elementos del array se almacenan por filas consecutivas donde, después del último elemento de una fila está el primer elemento de la siguiente fila. Ejemplo:



- Por orden de columna mayor: Los elementos del array se almacenan por columnas consecutivas donde, después del último elemento de una columna está el primer elemento de la siguiente columna. Ejemplo:



Si el *array* tiene, por ejemplo, 3 dimensiones, primero se almacenan los elementos de una página usando una de las 2 ordenaciones anteriores y, a continuación, los elementos de la siguiente página.

3.1.2. Cadena

La estructura de datos cadena o *string* es un tipo de datos compuesto de una secuencia de caracteres alfanuméricos que se almacenan de forma contigua en memoria. En realidad, una cadena se trata de un *array* de caracteres.

Las cadenas pueden representarse internamente de 2 maneras: de forma implícita y explícita. El uso de una forma u otra depende del lenguaje de programación utilizado.

- Organización implícita: Un carácter especial indica el fin de la cadena. En la mayoría de los lenguajes como C se usa el carácter nulo "\0". Ejemplo de cómo se representa la cadena "Hola" en un *array* de 7 elementos en C.

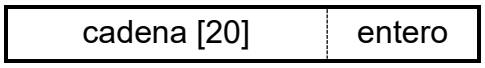
'H'	'o'	'l'	'a'	\0	\0	\0

- Organización explícita: En este caso, el primer Byte de la cadena sirve para indicar la longitud del texto. Pascal usa este método. Ejemplo de cómo se representa la cadena "Hola" bajo esta organización:

4	'H'	'o'	'l'	'a'	''	''

3.1.3. Registro

Un registro es una estructura de datos que puede contener datos de distinto tipo. Por ejemplo, un registro puede estar compuesto de una cadena de texto de tamaño 20 y un entero.

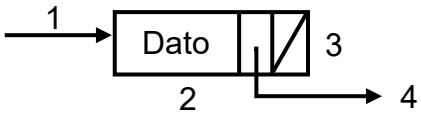


Debido a que cada registro puede tener una estructura distinta que depende de las necesidades del programador, no se pueden declarar tal cual y hay que diseñarlas.

Los registros creados a partir de estructuras estáticas se representan internamente en la memoria RAM de forma secuencial y con un tamaño fijo. En el caso del ejemplo anterior, quedaría representado como 20 caracteres más un entero.

3.2. Estructuras dinámicas

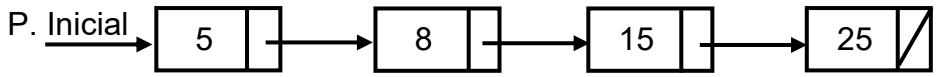
Una estructura dinámica es aquella donde el tamaño que ocupa en memoria puede cambiar durante la ejecución del programa. Esto es posible porque se basa en nodos que almacenan los datos y una o varias referencias a otros nodos (punteros). Ejemplo de nodo:



1 – Puntero que contiene la dirección de memoria donde se encuentra el nodo.
2 – Nodo que almacena 3 campos: un dato y dos punteros.
3 – Puntero con valor nulo (NULL). No apunta a ningún lugar.
4 – Puntero que contiene la posición de memoria de otro nodo.

3.2.1. Lista enlazada

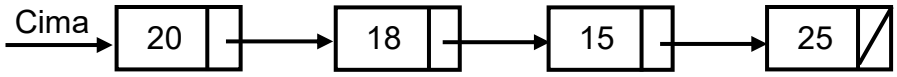
Una lista enlazada es una estructura de datos compuesta de un conjunto de nodos enlazados que mantienen una relación lineal (cada nodo tiene un antecesor y un sucesor, excepto el primero y el último) y secuencial (va pasando de un elemento a otro). Ejemplo:



Existen varios tipos de listas enlazadas como, por ejemplo, las listas circulares, las doblemente enlazadas y las listas multinivel.

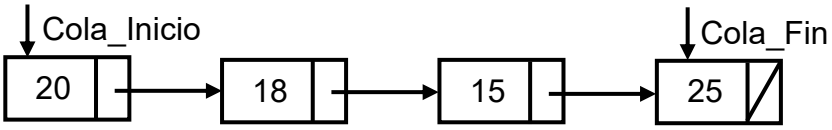
3.2.2. Pila

Una pila es una estructura de datos de tipo LIFO (*Last In, First Out*) o último en entrar, primero en salir. Esto quiere decir que el primer elemento que se introduce siempre estará depositado en el fondo de la pila (último lugar) y el último elemento que se introduce estará en la cima o cabecera (primer lugar). Ejemplo:



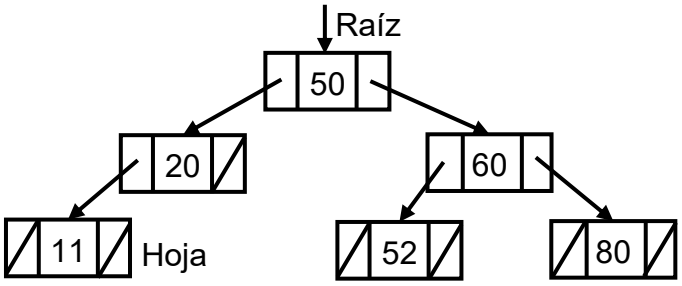
3.2.3. Cola

Una cola es una estructura de tipo FIFO (*First In, First Out*) o primero en entrar, primero en salir. Las colas utilizan 2 punteros iniciales, uno para apuntar al primer elemento y otro para apuntar al último. De esta forma, las inserciones se realizan por un extremo, mientras que las extracciones se hacen por el opuesto. Ejemplo:



3.2.4. Árbol

Un árbol es una estructura no lineal cuyos nodos se organizan de forma jerárquica, como en un árbol genealógico. Cada elemento del árbol tiene un único antecesor (padre), pero puede tener varios sucesores (hijos). El primer elemento del árbol se llama “raíz”, los nodos sin descendencia se llaman “hojas” y a cada generación en el árbol se le denomina “nivel”. Ejemplo:

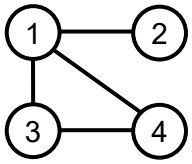


Existen distintos tipos de árboles, destacando los siguientes:

- Árbol binario de búsqueda (ABB): Árbol cuyos nodos pueden tener como máximo 2 descendientes.
- Árbol Addelson-Velskii y Landis (AVL): Es un ABB equilibrado, es decir, es un ABB cuyo factor de equilibro (Altura subárbol derecho – Altura subárbol Izquierdo) está entre -1 y 1.
- Árbol B+: Es un árbol cuyos nodos pueden tener más de 2 descendientes.
- Árbol rojinegro: Es un ABB equilibrado como el árbol AVL, pero obtiene su equilibrio de una forma distinta y menos rígida.

3.2.5. Grafo

Un grafo se define como una estructura G = (V, A) donde “V” es un conjunto de nodos llamados vértices y “A” son las aristas que unen los vértices. Ejemplo:



Los grafos se pueden representar internamente mediante:

- Una matriz de adyacencias: Se crea a partir de *arrays*. Útil si el número de vértices del grafo es fijo.
- Lista o multilista de adyacencias: Se crea a partir de nodos. Útil si el número de nodos puede variar.

4. CONCLUSIÓN

Durante el desarrollo de un *software*, el programador va declarando variables según van surgiendo necesidades. Estas variables pueden ser de tipo primitivo si almacenan un valor simple (entero, real, carácter, booleano o puntero) o de tipo estructurado si pueden agrupar varios valores dentro de la misma variable.

Las estructuras de datos pueden ser estáticas (*array*, cadena o registro) o dinámicas (lista, cola, pila, árbol o grafo), dependiendo de si el tamaño que ocupan en memoria durante la ejecución del programa es fijo o variable.

Independientemente de su tipo, todos los datos se representan internamente en binario. Para cada tipo existen diferentes sistemas de codificación como el Ca2 para los números enteros, el IEEE754 para los números reales o el ASCII para los caracteres, entre otros. El uso de un sistema u otro depende de varios factores como, por ejemplo, del compilador que se utilice, del lenguaje de programación, de la arquitectura a la que va dirigida el software, etc.

Conocer cada una de estos datos y su representación interna es fundamental para saber cuál elegir y poder desarrollar así un *software* de calidad, eficiente y sin errores.

5. BIBLIOGRAFÍA

- López Ureña, L. A. et al. (1997). *Fundamentos de Informática (1ª ed.)*. Ra-ma.
- Prieto Espinosa, A. et al. (2006). *Introducción a la informática (4ª ed.)*. McGraw-Hill.
- Brookshear, J. G. (2012). *Introducción a la computación (11ª ed.)*. Pearson Educación.
- Joyanes Aguilar, L. (2020). *Fundamentos de programación (5ª ed.)*. McGraw-Hill.

6. NORMATIVA

Para el desarrollo de este tema, se ha tenido en cuenta la siguiente normativa, donde se especifican los contenidos, competencias y criterios de evaluación de los Ciclos Formativos y Bachillerato en Andalucía:

- Orden 7 de julio de 2009 (SMR). La parte correspondiente al módulo “Sistemas Operativos Monopuesto”.
- Orden 19 de julio de 2010 (ASIR). La parte correspondiente al módulo “Implantación de Sistemas Operativos”.
- Orden 16 de junio de 2011 (DAW/DAM). La parte correspondiente a los módulos “Entornos de Desarrollo” y “Programación”.
- Instrucción 13/2022 (Bachillerato). La parte correspondiente a la asignatura “Tecnologías de la Información y Comunicación”