Sistemas y Aplicaciones Informáticas

Tema 10. Representación Interna de los Datos.

1. Ámbito de docencia.

- Sistemas informáticos monousuario y multiusuario (ASI 1).
- Sistemas informáticos multiusuario y en red (DAI 1).
- Sistemas operativos en entornos monousuario y multiusuario (ESI 1).

2. Representación interna de los datos.

2.1. Necesidad de abstracción. Datos.

- La abstracción es la capacidad de manejar un objeto o un hecho como un concepto general mediante sucesivos niveles de detalle. Implica una simplificación del problema tratado.
- Un dato representa una propiedad asociada al objeto o al hecho, y corresponde al nivel de abstracción más bajo que puede procesar un ordenador.

2.2. Procesamiento de datos. Tipos de dato y palabras.

- Los datos que un ordenador debe procesar pueden ser captados directamente por el sistema o pueden ser introducidos por cualquier medio. Para ello debe realizarse lo siguiente:
  - Transformación de los datos a código binario.
  - Almacenamiento de los datos en la memoria.
  - Estructuración de los datos para su manejo adecuado.
- Un tipo de dato define el conjunto de valores que un determinado dato puede tomar, y por tanto la cantidad de memoria que ocupa (aunque ésta depende del sistema), así como las operaciones básicas que se pueden realizar sobre dicho conjunto. Al especificar que un dato es de un determinado tipo, implícitamente se le asignan estas tres propiedades.
- Una palabra es el tamaño máximo de los datos que podrán ser manejados por el ordenador en cada acceso a la memoria para efectuar operaciones de lectura y escritura. Constituye la limitación física que posee un ordenador para manipular los datos, y varía con cada sistema.

3. Clasificación de los tipos de dato.

3.1. Introducción.

- Existen diferentes tipos de dato en función de las magnitudes o conceptos que se quieren representar, y cada uno de ellos tiene una representación interna distinta en el ordenador.
- Se pueden clasificar en:
  - *Simples*. Son aquellos que no pueden descomponerse en otros tipos de dato. Están incorporados en los lenguajes de alto nivel.
    - Numéricos. Representan magnitudes numéricas.
    - No numéricos. Representan conceptos no numéricos.
  - *Estructurados*. Son aquellos que están construidos a partir de otros tipos de dato.
    - Estáticos. El número de elementos que lo componen es fijo. Suelen estar incorporados en los lenguajes de alto nivel.
    - Dinámicos. El número de elementos que lo componen es variable. No están incorporados en los lenguajes de alto nivel y es necesario implementarlos. Pueden ser **lineales** (los datos están relacionados entre sí de forma secuencial) y **no lineales** (los datos pueden estar relacionados con cero o más datos).

3.2. Simples.

3.2.1. Numéricos.

3.2.1.1. Entero. Rango y operaciones.

- El tipo entero es una representación del conjunto de los números enteros. Se suele codificar internamente en formato binario puro para los enteros sin signo y en complemento a dos para los enteros con signo.
- Su rango de representación depende de su formato de representación interna:
  - *Enteros cortos (1 byte).*
    - Sin signo, desde 0 hasta 255 (2^8-1).
    - Con signo, desde -128 (-2^7) hasta 127 (2^7-1).
  - *Enteros (2 bytes).*
    - Sin signo, desde 0 hasta 65535 (2^{16}-1).
    - Con signo, desde -32768 (-2^{15}) hasta 32767 (2^{15}-1).
  - *Enteros largos (4 bytes).*
    - Sin signo, desde 0 hasta 4294967295 (2^{32}-1).
    - Con signo, desde -2147483648 (-2^{31}) hasta 2147483647 (2^{31}-1).
- Las operaciones que se pueden realizar entre datos de tipo entero son las aritméticas y las relacionales, con la restricción que impone el rango de representación.
- Si se pretende utilizar, en un rango determinado, cualquier número que sobrepase su límite máximo, se obtiene un resultado erróneo por desbordamiento de memoria (overflow).

3.2.1.2. Real. Rango, operaciones y precisión.

- El tipo real es una representación del conjunto de los números reales. Se suele codificar internamente en formato IEEE 754 (coma flotante).
- Su rango de representación depende de su formato de representación interna:
  - *Reales precisión simple (4 bytes)*. Desde 3.4E-38 hasta 3.4E+38.
  - *Reales precisión doble (8 bytes)*. Desde 1.7E-308 hasta 1.7E+308.
  - *Reales largos precisión doble (10 bytes)*. Desde 3.4E-4932 hasta 3.4E-4932.
- Las operaciones que se pueden realizar entre datos de tipo real son las aritméticas y las relacionales, con la restricción que impone el rango de representación.
- La precisión está limitada al número de bits utilizados para representar la mantisa. La sucesión de operaciones aritméticas provoca errores de redondeo que se acumulan durante todo el cálculo.
- Si se pretende utilizar, en un rango determinado, cualquier número que sobrepase su límite máximo, se obtiene un resultado erróneo por desbordamiento de memoria (overflow). Cuando el número es menor que el límite inferior del rango, de modo que la mantisa sólo contiene ceros, se produce un error por desbordamiento inferior (underflow).

3.2.2. No numéricos.

3.2.2.1. Carácter. Operaciones.

- El tipo carácter representa a uno de los símbolos que forman parte de un código estándar de caracteres. Normalmente se utiliza la serie ASCII (código estándar americano para intercambio de información) extendida de 8 bits, y por tanto se almacenan en un byte.

- Pueden contener caracteres alfanuméricos, especiales y de control. Suelen expresarse encerrados entre caracteres especiales delimitadores, por ejemplo apóstrofes.
- Pueden utilizarse en operaciones relacionales teniendo en cuenta que cuando en una expresión se comparan caracteres se hace con relación al valor de su código.

3.2.2.2. Booleano. Operaciones.

- El tipo booleano sólo puede tener uno de dos valores posibles: verdadero o falso. Se almacena en memoria como un byte de ceros para el valor falso, y cualquier otro para el verdadero.
- Está especialmente diseñado para ser utilizado en operaciones lógicas.

3.2.2.3. Puntero. Operaciones.

- El tipo puntero sólo puede almacenar la dirección de memoria de otro dato. Su tamaño depende del número de direcciones de memoria a las que puede acceder el sistema, normalmente 4 bytes.
- Sólo es posible la suma y la resta de punteros con enteros, nunca entre punteros. También son posibles las expresiones relacionales entre punteros.

3.3. Estructurados.

3.3.1. Estáticos.

3.3.1.1. Vectores. Operaciones.

- Son estructuras que agrupan un número fijo y finito de datos del mismo tipo en posiciones contiguas de memoria bajo un nombre común. Son estructuras de una sola dimensión.
- Cada uno de los elementos del vector se referencia mediante el nombre común y un índice, que es un número entero que determina su posición relativa en la memoria.
- Las operaciones que pueden realizarse son las siguientes:
  - *Recorrido*. Proceso total o parcial de los elementos del vector.
  - *Búsqueda*. Localización de la posición de un elemento con una determinada clave.
  - *Ordenación*. Organización de los elementos conforme a un determinado criterio.
  - *Inserción*. Adición lógica de un nuevo elemento.
  - *Borrado*. Eliminación lógica de un elemento.

3.3.1.2. Cadenas. Semejanza con vectores.

- Son estructuras que agrupan un número fijo y finito de caracteres en posiciones contiguas de memoria bajo un nombre común, que suelen manejarse en conjunto.
- En algunos lenguajes existen como un tipo de dato más y en otros, como C, las cadenas son vectores de caracteres que terminan con un carácter nulo. La mayoría de los lenguajes implementa funciones que permiten la manipulación de cadenas.

3.3.1.3. Matrices. Semejanza con vectores. Multidimensión.

- Son estructuras que agrupan un número fijo y finito de vectores del mismo tamaño y tipo de datos en posiciones contiguas de memoria bajo un nombre común, como un vector de vectores.
- Tienen dos dimensiones, y cada uno de los elementos de la matriz se referencia mediante el nombre común, el índice del vector contenedor (fila) y el índice del elemento dentro del vector contenido (columna).
- Extendiendo este concepto es posible realizar matrices de más de dos dimensiones, aunque no se suelen utilizar más de tres dimensiones debido a su complejidad de manejo.

3.3.1.4. Registros.

- Son estructuras que agrupan un número fijo y finito de datos de distinto tipo y tamaño bajo un nombre común. Cada uno de los elementos del registro se denomina campo y se representa mediante un identificador propio. Los campos suelen mantener una relación lógica entre sí.
- La declaración de un registro crea un nuevo tipo de dato que puede asignarse a una variable. Cada uno de los campos de un registro se referencia mediante el nombre de la variable a la que se ha asignado el registro, un operador (punto) y el nombre del campo.
- Los campos de un registro pueden ser de tipo simple o de tipo estructurado estático.

3.3.1.5. Uniones.

- Representan una misma posición de memoria compartida por dos o más datos diferentes. La cantidad de memoria ocupada es igual a la del dato de mayor tamaño declarado en la unión.
- El acceso a uno de los datos de la unión es similar a la de los registros. Las uniones permiten relacionar una misma posición de memoria con diferentes tipos de dato. Esto es útil para las conversiones de tipos, ya que es posible referirse a los datos de la unión de diferentes formas.

3.3.2. Dinámicos.

3.3.2.1. Lineales.

3.3.2.1.1. Pilas. Operaciones. Implementaciones.

- Son estructuras de datos que se caracterizan porque las operaciones de inserción y eliminación de elementos se realizan solamente en un extremo de la estructura. El extremo donde se realizan estas operaciones se denomina habitualmente cima. Por esta razón, se dice que una pila es una estructura lineal de tipo LIFO (Last In First Out, el último que entra es el primero que sale).
- Existen una serie de operaciones necesarias para la manipulación de las pilas, que son:
  - *Crear la pila*. La operación de creación inicia la pila como vacía.
  - *Añadir elementos*. Coloca un nuevo elemento en la cima de la pila.
  - *Eliminar elementos*. Devuelve el elemento situado en la cima y lo elimina de la pila.
  - *Comprobar si la pila está vacía*. Para decidir si es posible eliminar elementos de la pila.
  - *Comprobar si la pila está llena*. Para decidir si es posible añadir elementos a la pila.
- Existen dos posibles implementaciones en los lenguajes de alto nivel:
  - *Estática*. Es la mejor opción, por rapidez y sencillez, cuando se conoce el número máximo de datos que deben ser almacenados. De esta manera, la pila se representa mediante un registro con un campo formado por un vector que almacena los elementos de la pila, y un campo cima que almacena el índice del último elemento introducido en la pila.
  - *Dinámica*. Es la única opción cuando no se conoce el número máximo de datos que deben ser almacenados. De esta manera, la pila se representa como una sucesión de nodos creados en memoria dinámicamente. Cada nodo es un registro con dos campos, uno de ellos almacena uno de los elementos de la pila y el otro es un puntero al siguiente nodo. El puntero del último nodo señala un valor nulo, y existe un puntero externo que señala al primer nodo de la pila.

3.3.2.1.2. Colas. Operaciones. Implementaciones.

- Son estructuras de datos que se caracterizan porque las operaciones de inserción y eliminación de elementos se realizan en los extremos opuestos de la estructura. La inserción se produce en el extremo derecho o final de la estructura, mientras que la eliminación se realiza en el extremo izquierdo o inicio. Por esta razón, se dice que una cola es una estructura lineal de tipo FIFO (First In First Out, el primero que entra es el primero que sale).
- Existen una serie de operaciones necesarias para la manipulación de las colas, que son:
  - *Crear la cola*. La operación de creación inicia la cola como vacía.
  - *Añadir elementos*. Coloca un nuevo elemento en el final de la cola.
  - *Eliminar elementos*. Devuelve el elemento situado en el inicio y lo elimina de la cola.
  - *Comprobar si la cola está vacía*. Para decidir si es posible eliminar elementos de la cola.
  - *Comprobar si la cola está llena*. Para decidir si es posible añadir elementos a la cola.
- Existen dos posibles implementaciones en los lenguajes de alto nivel:
  - *Estática*. La representación más eficiente se realiza mediante una cola circular. De esta manera, la cola se representa mediante un registro con un campo formado por un vector que almacena los elementos de la cola, un campo inicio que almacena el índice anterior al primer elemento de la cola y un campo final que almacena el índice del último elemento.
  - *Dinámica*. De esta manera, la cola se representa como una sucesión de nodos creados en memoria dinámicamente. Cada nodo es un registro con dos campos, uno de ellos almacena uno de los elementos de la cola y el otro es un puntero al siguiente nodo. El puntero del último nodo señala un valor nulo, y existen dos punteros externos: uno señala al nodo de inicio de la cola, y otro señala al nodo final.

3.3.2.1.3. Listas enlazadas. Operaciones. Implementación. Tipos.

- Son estructuras de datos que se caracterizan porque las operaciones de inserción y eliminación de elementos se realizan en cualquier punto según un determinado criterio de ordenación. Se trata de una generalización del concepto de pilas y colas.
- Existen una serie de operaciones necesarias para la manipulación de las listas enlazadas, que son:
  - *Crear la lista*. La operación de creación inicia la lista como vacía.
  - *Insertar elementos*. Coloca un nuevo elemento dentro de una lista ordenada.
  - *Eliminar elementos*. Busca un elemento situado en cualquier posición y lo elimina.
  - *Recorrer la lista*. Procesa todos los elementos de la lista empezando por el primero.
  - *Comprobar si la lista está vacía*. Para decidir si es posible eliminar elementos de la lista.
- En pilas y colas cada elemento de la estructura de datos (estructura lógica) ocupa la posición de memoria (estructura física) contigua a la del elemento que le precede, lo cual favorece la implementación estática. En las listas enlazadas esto no es así; por tanto para evitar el movimiento de elementos cada vez que se realiza una operación de inserción o eliminación, la lista se representa como una sucesión de nodos creados en memoria dinámicamente:
  - Cada nodo es un registro con dos campos, uno de ellos almacena uno de los elementos de la lista y el otro es un puntero al siguiente nodo.
  - El puntero del último nodo señala un valor nulo, y existen dos punteros externos: uno señala al nodo de inicio de la lista, y otro señala a un punto de interés.

- Otros tipos de listas enlazadas son las siguientes:
  - *Listas circulares*. Son listas enlazadas en las que el puntero del último nodo señala al nodo de inicio de la lista. Presentan la ventaja de que partiendo de cualquier elemento se puede hacer un recorrido completo de la lista.
  - *Listas doblemente enlazadas*. Son listas enlazadas en las que cada nodo apunta a su sucesor y a su predecesor. Presentan la ventaja de que se puede hacer un recorrido de la lista en ambos sentidos, pero implican una mayor reserva de memoria por cada nodo.

3.3.2.2. No lineales.

3.3.2.2.1. Árboles. Tipos. Operaciones. Implementación.

- Un árbol es un conjunto finito de cero o más nodos, tal que existe un nodo especial, llamado nodo raíz, y donde los restantes nodos están separados en $n \geq 0$ conjuntos disjuntos, cada uno de los cuales es a su vez un árbol llamado subárbol del nodo raíz. Esto implica que cada nodo del árbol es raíz de algún subárbol contenido en el árbol principal.
- Algunos tipos de árboles son los siguientes:
  - *Árboles binarios*. Constituyen un tipo particular de árboles de gran aplicación, y se caracterizan porque cada nodo tendrá como máximo dos subárboles, llamados subárbol izquierdo y subárbol derecho. Cualquier árbol puede transformarse en un árbol binario.
  - *Árboles binarios enlazados*. Son árboles binarios en los que los enlaces nulos situados en el subárbol derecho de un nodo apuntan al sucesor de ese nodo en un determinado recorrido del árbol, mientras que los enlaces nulos en el subárbol izquierdo apuntan al predecesor del nodo en el mismo tipo de recorrido.
  - *Árboles binarios de búsqueda (ABB)*. Son árboles binarios utilizados para encontrar un determinado nodo sin ser necesario recorrer todos los nodos que le preceden:
    - Todos los nodos están identificados por una clave única.
    - Las claves de los nodos del subárbol izquierdo son menores que la clave del nodo raíz.
    - Las claves de los nodos del subárbol derecho son mayores que la clave del nodo raíz.
    - Los subárboles izquierdo y derecho son también árboles binarios de búsqueda.
  - *Montículos*. Un montículo de máximos (mínimos) es un árbol binario completo tal que, el valor de la clave de cada nodo es mayor (menor) o igual que las claves de sus nodos hijos, si los tiene. Se utilizan para el mantenimiento de las colas de prioridad.
- Existen una serie de operaciones necesarias para la manipulación de árboles binarios, que son:
  - *Recorrer el árbol*. Es el proceso que permite acceder una sola vez a cada uno de los nodos del árbol. El recorrido completo de un árbol produce un orden lineal en la información del árbol. Adoptando el convenio de que siempre se recorre el subárbol izquierdo antes que el derecho, puede hacerse en amplitud (recorriendo todos los nodos de cada nivel empezando por el raíz) o en profundidad, de la siguiente manera:
    - **Pre-orden**. Nodo, subárbol izquierdo y subárbol derecho.
    - **In-orden**. Subárbol izquierdo, nodo y subárbol derecho.
    - **Post-orden**. Subárbol izquierdo, subárbol derecho y nodo.

- *Buscar un nodo en un ABB*. Es una búsqueda binaria de manera recursiva o iterativa. En un ABB los nodos están ordenados por su clave recorriendo el árbol según el criterio in-orden.
- *Insertar un nodo en ABB*. Siempre será un nodo hoja, y la posición se determina mediante una búsqueda binaria en función de la clave del nodo a insertar.
- *Eliminar un nodo en un ABB*. Existen tres casos posibles:
  - **Nodo sin hijos**. El nodo se elimina directamente.
  - **Nodo con un hijo**. El nodo se elimina y se sustituye por su hijo.
  - **Nodo con dos hijos**. El nodo se elimina y se sustituye por el nodo hoja que tenga la clave menor de todos los nodos del subárbol derecho, o bien por el nodo hoja que tenga la clave mayor de todos los nodos del subárbol izquierdo.

– Un árbol binario se representa como una sucesión de nodos creados en memoria dinámicamente. Cada nodo es un registro con tres campos, uno de ellos almacena uno de los elementos del árbol y los otros dos son punteros que señalan al primer nodo del subárbol izquierdo y al primer nodo del subárbol derecho respectivamente. Los punteros de los nodos sin subárbol izquierdo o sin subárbol izquierdo señalan un valor nulo.

3.3.2.2.2. Grafos. Grafos dirigidos y ponderados. Representación.

– Un grafo es un conjunto finito de nodos unidos entre sí por arcos. Un nodo puede tener cero o más arcos, pero todo arco debe unir exactamente dos nodos. Los grafos representan conjuntos de objetos que no tienen restricción jerárquica entre ellos, son un superconjunto de los árboles.
– Un grafo es dirigido si los arcos tienen una única dirección, y es ponderado si tienen un peso.
– Los grafos pueden representarse mediante:
  - *Matrices de adyacencia*. Consiste en una matriz cuadrada en la que el número de filas y columnas corresponde al número de nodos en el grafo, de manera que los arcos entre los nodos se ven como relaciones entre los índices de la matriz. Un par de índices contiene el peso del arco si los nodos correspondientes son adyacentes, y cero en caso contrario.
  - *Listas de adyacencia*. Consiste en almacenar por cada nodo una lista dinámica con los nodos a los que se puede acceder desde él. No consume tanta memoria como las matrices de adyacencia, pero presenta la dificultad de obtener las relaciones inversas.

4. Manejo de los datos.

4.1. Variables y constantes. Atributos.

– Las variables y las constantes son referencias internas de los programas a las direcciones de memoria ocupadas por los datos. Se diferencian en que durante la ejecución del programa el valor de una constante permanece inalterado y el valor de una variable puede modificarse.
– Tienen los siguientes atributos:
  - *Identificador*. Nombre único dentro del programa formado por caracteres alfanuméricos.
  - *Tipo*. Puesto que hacen referencia a datos, deben tener asignado un tipo de datos.
  - *Valor*. Contenido del dato expresado según su tipo de dato.
  - *Ámbito*. Parte del programa en el que son conocidas.
  - *Tiempo de vida*. Parte del programa en el que están asociadas a la posición de memoria.

4.2. Tipos de dato de usuario.

- La ventaja de los lenguajes de programación de alto nivel es que permiten la creación de tipos de dato definidos por el usuario, agrupando datos de tipo simple, de tipo estructurado estático o de otro tipo de dato definido por el usuario. En realidad no se crea un tipo de dato nuevo, sino que se asigna un nombre a una agrupación de tipos de dato ya existente.
- Se utilizan para parametrizar los programas contra los problemas de portabilidad. Si un tipo de dato concreto depende del sistema, bastará con cambiar su definición sin modificar nada más.

4.3. Equivalencia y conversión entre tipos de dato.

- Hay ciertos tipos de operaciones que sólo se pueden realizar entre tipos de dato que sean equivalentes. Por ello resulta necesario definir el criterio de equivalencia de tipos de dato que sigue un lenguaje determinado.
- Normalmente, los criterios de equivalencia utilizados son dos:
  - *Equivalencia estructural*. Dos variables son equivalentes cuando la estructura de la información que contienen es la misma. Este criterio es el utilizado en el lenguaje C.
  - *Equivalencia por nombre*. Dos variables son equivalentes únicamente cuando se han definido utilizando el mismo nombre de tipo de dato.
- La conversión entre tipos de dato es necesaria cuando se mezclan en una expresión variables de un tipo con variables de otro tipo distinto. Los lenguajes de programación suelen realizar las siguientes conversiones automáticas de tipos:
  - En las expresiones se convierten todos los operandos al tipo del mayor operando.
  - En una asignación el valor del lado derecho se convierte al tipo de datos del lado izquierdo.
- Para evitar que se produzca pérdida de información por truncamiento o redondeo, o para asegurar la compatibilidad de tipos, los lenguajes de programación suelen permitir el forzamiento (cast) del tipo de una variable en una expresión.