ESCUELA DE PREPARACIÓN DE OPOSITORES

E. P. O.

C/. La Merced, 8 – Bajo A Telf.: 968 24 85 54

30001 MURCIA

INF11 – SAI11

Organización lógica de los datos. Estructuras estáticas.

Esquema.

1 Introducción.

Los computadores son máquinas para el tratamiento automático de la información. Esta información no se almacena ni se representa al azar, sino que debe organizarse o estructurarse de forma adecuada para obtener un rendimiento razonable en su almacenamiento, tratamiento y recuperación. El tratamiento de la información se debe realizar de un modo sistemático. La resolución de cualquier problema conlleva el encontrar un método de resolución expresado con la suficiente precisión para poder ser descompuesto en acciones realizables por el computador (esto es lo que denominaremos un algoritmo).

Toda constante, variable, expresión o función es de cierto tipo. Este tipo caracteriza esencialmente el conjunto de valores al cual pertenece una constante o bien que puede ser tomado por una variable o expresión o quizá que puede ser generado por una función. El tipo asociado se hace explícito en una declaración de la constante, variable o función y esta declaración precede textualmente a la aplicación de esa constante, variable o bien función.

Las características principales del concepto de tipo de datos son las siguientes:

1. Un tipo de datos determina el conjunto de valores al cual pertenece una constante o el cual puede ser tomado por una variable o por una expresión; o bien que puede ser generado por un operador o por una función.
2. El tipo de un valor representado por una constante, variable o expresión puede derivarse de su forma o de su declaración sin que se necesite ejecutar el proceso de cálculo.
3. Cada operador o función aguarda la llegada de argumentos de un tipo fijo y da como resultado un tipo fijo. Si un operador admite argumentos de varios tipos, entonces el tipo del resultado se puede determinar de las reglas específicas del lenguaje.

En muchos casos nuevos tipos de datos se definen en términos de tipos de datos anteriormente definidos. Los valores de estos tipos generalmente son conglomerados de valores de componentes de los tipos constituyentes y se dice que son estructurados. Si hay solamente un tipo constituyente, es decir, si todas las componentes son del mismo tipo constituyente, entonces éste se conoce como el tipo base.

Ya que los tipos constituyentes pueden volver a ser estructurados, pueden constituirse jerarquías completas de estructuras pero las componentes últimas de una estructura son atómicas. Por lo tanto, se necesita ofrecer una notación que introduzca de igual manera estos tipos primitivos y no estructurados. Un método directo consiste en enumerar los valores que constituirán el tipo. Si existe un ordenamiento entre los valores individuales, entonces se dice que el tipo es ordenado o bien escalar.

De este modo es posible definir tipos primitivos y construir tipos estructurados que se limitan a un grado arbitrario de inserción o anidamiento. En la práctica, no es suficiente tener sólo un método general de combinación de tipos constituyentes en una estructura. Considerando problemas prácticos de representación y uso, un lenguaje de programación de uso general debe ofrecer varios métodos de estructuración. Los métodos básicos de estructuración que se presentan aquí son el array, el registro y la secuencia.

En este tema comenzamos estudiando el concepto de dato desde un enfoque lógico. Describimos los tipos de datos elementales más usuales en informática, analizando algunas estructuras de datos utilizadas en programación, en sistemas operativos, o en el diseño físico de computadores.

2 Tipos de datos.

Se denomina dato a cualquier objeto manipulable por el computador. Un dato puede ser un carácter leído de un teclado, información almacenada en un disco, un número que se encuentra en memoria principal, etc.

Datos son tanto las constantes definidas dentro de los programas, que no alteran su valor durante la ejecución de éstos, como las variables utilizadas en dichos programas. Asimismo son datos la información externa al programa, a la que se puede acceder mediante algún procedimiento, ya esté dicha información grabada en algún medio de memoria masiva o sea generada por algún periférico.

Cuando utilizamos un computador para resolver un problema, debemos hacer una abstracción de éste y de las magnitudes que influyen en él. Dichas magnitudes deben ser representadas por datos. La representación de una magnitud como un dato, se puede entender como una aplicación que hace corresponder un dato a cada valor de la

magnitud. Así, por ejemplo, para resolver un problema en que intervengan distancias, es preciso tener una aplicación, T, del conjunto de valores de la magnitud distancia, d, en un conjunto de datos, D.

Esta transformación es deseable que esté definida sobre todo el conjunto de valores de la magnitud. Es conveniente, además, que sea unívoca, es decir, que a dos valores de magnitud distintos les asocie datos distintos. Para que se pueda operar con los datos es necesario que existan unas operaciones internas en el conjunto de datos, que sean semejantes a las operaciones usuales en el conjunto de magnitudes. Dichas operaciones deben cumplir que: la imagen según la transformación T del resultado de una operación en el conjunto de magnitudes, sea igual al resultado de las operaciones correspondientes en el conjunto de datos sobre las imágenes de los operandos.

Si denotamos por + la suma en el conjunto de magnitudes, d, lo dicho anteriormente implica que debe existir una operación ⊕ sobre el conjunto de datos, D, que cumpla:

$$T(x+y) = T(x) \oplus T(y) = X \oplus Y \qquad \forall x,y \in d$$

Para que los resultados obtenidos en el conjunto de datos, D, puedan ser interpretados, es necesario que exista una transformación, T^{-1}, de éstos al conjunto de magnitudes, d. Esta aplicación T^{-1} hará corresponder a cualquier dato, X, un valor de magnitud, x, cuya imagen por T es el propio dato X.

Se denomina tipo de dato al conjunto de la transformación, T, y de las operaciones y funciones internas y externas definidas sobre el conjunto de datos. Distintas transformaciones darán lugar a distintos tipos de datos, aun cuando el conjunto origen a representar sea el mismo.

Nos encontraremos tipos de datos en la representación de la información, tanto a nivel físico del computador, como en los lenguajes de programación. No todos los tipos de datos existen en todos los lenguajes de programación. Hay lenguajes más ricos que otros en tipos de datos.

3 Tipos de datos primitivos.

3.1 Tipos primitivos estándar.

Los tipos primitivos estándar son aquellos que se tienen a disposición en muchas computadoras como características integradas.

3.1.1 Datos de tipo entero.

El tipo entero comprende un subconjunto de números enteros cuyo tamaño puede variar entre los diversos sistemas de computación. Si una computadora utiliza n bits, para representar un entero en notación de complemento a 2, entonces los valores posibles de x deben cumplir la desigualdad $-2^{n-1} \leq x < 2^{n-1}$. La razón de esta limitación está en utilizar un espacio finito y fijo para cada dato, y en el hecho de que la transformación realizada entre los números enteros y el tipo de datos consiste en representar el número en binario y almacenarlo con un número fijo de bits.

El número de datos distintos de tipo entero que se pueden generar, cardinalidad del tipo, es 2^n (donde n es el número de bits que se utiliza en la representación). Por tanto, si se modifica el número de bits, n, se obtienen distintos tipos enteros. En consecuencia, no todos los números enteros se pueden representar.

Se supone que todas las operaciones con datos de este tipo son exactas y corresponden a las leyes ordinarias de la aritmética. En el caso de un resultado que quede fuera del subconjunto representable se obtendría un resultado no representable en el tipo. Este evento se llama desbordamiento (*overflow*).

Los operadores estándar son las cuatro operaciones aritméticas básicas de adición, sustracción, multiplicación y división (división entera y resto).

En lenguaje C este tipo es `int`, y el programador tiene control sobre él, pudiendo especificar `short`, `long`, `signed` y `unsigned`.

3.1.2 Datos de tipo real.

El tipo real representa un subconjunto de los números reales. Se supone que la aritmética con operandos del tipo entero produce resultados exactos, pero la aritmética sobre valores de tipo real puede ser inexacta dentro de los límites de errores de redondeo ocasionados por operaciones de cálculo con un número finito de cifras. Esta es la razón principal de la distinción explícita entre los tipos entero y real.

En matemáticas el conjunto de los reales es un conjunto infinito que se extiende desde infinito hasta -infinito. Este conjunto a diferencia del conjunto de los enteros es "no numerable", ya que cualquier intervalo siempre contiene infinitos valores. En el terreno de los computadores el tipo real, como es obvio, no representa un conjunto no numerable sino que su dominio estará formado por un conjunto numerable de "representantes" de intervalos del continuo real.

El tipo real está implementado a nivel máquina mediante la técnica de punto flotante. Esta técnica de representación podemos resumirla en los siguientes puntos:

1. Se elige un tamaño (número de bits, `N`) para la representación.
2. Cada valor real, `M`, se representa internamente en formato exponencial, frecuentemente empleado en problemas científicos cuando se quieren expresar magnitudes tales como la masa de la tierra o el número de Avogadro. `M = mantisa x base`$^{\texttt{exponente}}$. La base se elige por convenio: binaria, octal o hexadecimal. Por ejemplo, si la base es binaria será preciso codificar `M` en binario y entonces ponerlo en forma exponencial.
3. `N` se descompone en dos partes. Una de `M` bits y otra de `N-M` bits, destinadas a la codificación de la mantisa y el exponente, respectivamente.
4. El número de bits de la mantisa determina la precisión y el número de bits del exponente el rango.
5. Cuando empleamos la notación exponencial para expresar un valor real, es evidente que a un mismo valor le corresponden múltiples representaciones diferentes. Es por ello que en punto flotante se codifica el valor real en una forma normalizada `(1/base)≤mantisa≤1`.

Esta representación no permite el almacenamiento de números muy grandes o muy pequeños, lo que conlleva que se produzcan desbordamientos (*overflows* o *underflows*). Distinguimos varias zonas:

- Zonas de desbordamiento. No podemos representar valores fuera del rango establecido por el exponente. Si una operación produce un valor en esa zona obtenemos en tiempo de ejecución una indicación de error por desbordamiento.

- Zonas que corresponden a valores que no podemos representar porque 0 es el representante que corresponde a ese intervalo. En este caso no se produce error sino que el valor obtenido se aproxima a 0 lo cual, salvo en cálculos que requieran mucha precisión, es admisible.
- Zonas que corresponden a los representantes que constituyen el dominio del tipo real supuesto. Es interesante notar que la distancia, *d*, entre dos mismos representantes consecutivos no se mantiene constante a lo largo de estas zonas, sino que es función del exponente. Para exponentes altos es mucho más significativo, es decir aumenta la magnitud de la anchura del intervalo real asociado a un representante conforme aumentamos el valor del exponente.

Podemos resumir toda esta discusión presentando un conjunto reducido de los axiomas para el tipo real que propone N. Wirth en su libro "Programación Sistemática":

Axioma 1 El tipo real, TR, es un subconjunto del conjunto de los reales, R.

Axioma 2. A todo número $x \in R$ se le asocia un número $x \in TR$ que llamaremos su representante.

Axioma 3. Todo x representa varios números reales y éstos constituyen un conjunto coherente, es decir si $x_1<x_2$ y $x_1=r$ y $x_2=r$, entonces $x=r$ para todo $x_1<=x<=x_2$. Mas aún $x \in R$ implica que $x=x$. En particular 0 y 1 tienen representaciones exactas.

Axioma 4: Existe un valor *max* tal que no existen representantes para x tal que |x|>max. Se llama rango de desbordamiento al conjunto de los números |x|>max.

Axioma 5: TR es simétrico con respecto al 0. (-x) = - (x)

Axioma 6: La suma y multiplicación de representantes es conmutativa.

Axioma 7: No se cumplen las propiedades asociativa ni distributiva. Por ejemplo, suponiendo una representación con mantisa de 4 dígitos, si x=9.900, y=1.000, z=-0.999 entonces (x+y)+z=9.910 y x+(y+z)=9.901. Por otra parte si x=1.100, y=5.000 z=5.001 entonces (x*y)+(x*z)=1.000 y x*(y+z)=1.100.

De los axiomas 1 al 4 se deduce que $x<y \Rightarrow x<=y$, $x=y \Rightarrow x=y$, y $x>y \Rightarrow x>=y$. Por todo esto se deduce que no es adecuado hacer comprobaciones de igualdad donde intervengan variables reales debido al carácter aproximado de los cálculos. Así puede suceder que 3*(1/3) sea distinto de 1. Conviene, por tanto, en vez de comprobar la igualdad de dos valores reales, comprobar que el valor absoluto de su diferencia es menor que una constante cercana a cero, tanto como lo requiera la magnitud de los valores.

En lenguaje C, este tipo puede ser `float`, `double` o `long double`, dependiendo de la precisión deseada.

3.1.3 Datos de tipo lógico.

Los datos de tipo lógico representan valores lógicos o booleanos. Pueden tomar uno de entre dos valores: verdadero o falso (abreviadamente V, F ó 0, l).

Sobre los valores lógicos pueden actuar los llamados operadores lógicos. Los operadores lógicos fundamentales son: Y, O y NO (en inglés: AND, OR y NOT, respectivamente). La definición de las operaciones se hace indicando su resultado para las cuatro posibles combinaciones de valores de los argumentos. En algunos lenguajes de programación hay definidos sobre los datos de tipo lógico otros operadores booleanos, como son: NO-Y, NO-O y NO-exclusivo (en inglés: NAND, NOR y XOR).

b	C	b Y c	b O c	NO b	b NO-Y c	b NO-O c	b NO-exclusivo c
V	V	V	V	F	F	F	F
V	F	F	V	F	V	F	V
F	V	F	V	V	V	F	V
F	F	F	F	V	V	V	F

Un caso particularmente importante de valor de tipo lógico es el obtenido como resultado de una operación de relación sobre datos de un tipo, para el que existe una relación de orden (tipos entero, real, carácter, enumerado y subrango, por ejemplo). Una relación es una expresión formada por dos operandos pertenecientes a un mismo tipo ordenado y un operador de relación. Son operadores de relación los siguientes: ≥ (mayor o igual), > (mayor), ≤ (menor o igual), < (menor), = (igual), <> (distinto). El resultado de una operación de relación es el valor lógico "verdadero" si la relación expresada es cierta, y "falso" en caso contrario. Una relación es, pues, una operación externa sobre el tipo de datos de los operandos.

En lenguaje C, este tipo no está implementado pero puede utilizarse a través del tipo `int`, ya que cualquier operación se considera falsa cuando es evaluada como 0, y cierta cuando es evaluada con un valor diferente de 0.

3.1.4 Datos de tipo carácter.

Los datos de tipo carácter representan elementos individuales de conjuntos finitos y ordenados de caracteres. No hay ninguna operación interna sobre datos de tipo carácter (salvo la asignación). Normalmente existen funciones de conversión de tipo. Como por ejemplo, la que asocia a cada dato de tipo carácter un valor entero, que indica su posición en el código.

Desafortunadamente, no existe un conjunto de caracteres generalmente aceptado que se use en todos los sistemas de computación. En consecuencia, el uso del atributo "estándar" puede en este caso ser casi engañoso; se entenderá en el sentido de "estándar en el sistema de computación en el cual se ejecute cierto programa".

El conjunto de caracteres definido por la ISO y particularmente su versión estadounidense ASCII, es el conjunto más ampliamente aceptado.

A fin de poder diseñar algoritmos con caracteres que sean independientes del sistema, debemos poder suponer ciertas propiedades de los conjuntos de caracteres como grupo, a saber:

1. El tipo carácter contiene las 26 cifras mayúsculas latinas, las 26 letras minúsculas, los 10 numerales arábigos y otros caracteres gráficos, como los signos de puntuación.
2. Los subconjuntos de letras y dígitos son ordenados y contiguos.
3. El tipo carácter contiene un carácter en blanco que no se imprime y que se puede utilizar como separador.

En C, este tipo es `char`. La disponibilidad de funciones de transferencia de tipo estándar entre los tipos carácter y entero es particularmente importante cuando se quiere escribir programas de forma independiente de la máquina, sin embargo en C, no son necesarias debido a que un carácter es un tipo especial de entero, de forma que dicho entero representa el número ordinal del carácter en el conjunto de caracteres y viceversa.

3.2 Tipos primitivos no estándar.

Al contrario de los tipos vistos anteriormente, el tipo de datos enumerado, como el tipo subrango y los tipos estructurados que veremos posteriormente en las secciones siguientes, no es un tipo normalizado. Puede haber muchos tipos de datos enumerados distintos dentro de un programa en un lenguaje determinado, mientras que no habrá más que un tipo lógico.

Los tipos de datos vistos en las secciones anteriores son usualmente tratados por el computador a nivel hardware. Mientras que el tipo de datos enumerado y el tipo de datos subrango sólo son interpretados por el software (por el traductor del lenguaje).

3.2.1 Datos de tipo enumerado.

Los datos de tipo enumerado se definen explícitamente dando un conjunto finito de valores. En cierto modo podemos considerar al tipo de dato enumerado como una clase de tipos de datos, a la que pertenecerán todos los tipos definidos por enumeración.

Internamente, los datos de tipo enumerado se almacenan como valores enteros. A cada valor del tipo se le asocia un entero consecutivo, comenzando por cero. Existen, como en el tipo carácter, funciones de conversión a entero. Pueden también existir funciones que generen el valor siguiente (sucesor) o anterior (predecesor) a uno dado, según el orden en que éstos se definieron.

En el lenguaje C, su definición tiene la forma

$$\texttt{enum T } \{c_1, c_2, \ldots, c_n\};$$

donde `T` es el nuevo identificador del tipo y c_i son los nuevos identificadores de las constantes. La cardinalidad de `T` es card(T) = n. La definición de estos tipos presenta no sólo un nuevo identificador de tipos, sino también al conjunto de identificadores que representan los valores del nuevo tipo. Estos identificadores pueden, por tanto, utilizarse como constantes en el programa completo y resaltan considerablemente su facilidad de entendimiento.

Ejemplo:

```
enum dias_semana {lunes, martes, miercoles, jueves, viernes, sabado,
                  domingo};
/* Ahora podemos asignar valores a la variable dia, compararlos, etc.:
*/
enum dias_semana dia;
dia = jueves;
```

En C, el ordenador almacena cada valor asociándole un número entero empezando por el 0 y siguiendo consecutivamente, a menos que se le asigne uno definido durante la declaración. En nuestro caso, lunes tendría asociado el valor 0, martes el 1, etc. Si la declaración hubiera sido:

```
enum dias_semana {lunes=1, martes, miercoles, jueves, viernes, sabado,
                  domingo};
```

entonces, lunes tendría asociado el valor 1, martes el 2, etc.

3.2.2 Datos de tipo subrango.

El tipo subrango se define a partir del tipo entero, carácter o de un tipo enumerado. Un dato de tipo subrango puede tomar determinados valores del tipo original, a partir del cual se ha definido el subrango, entre un mínimo y un máximo. Con datos de tipo subrango se pueden realizar las operaciones definidas para el tipo original.

Este tipo no existe en el lenguaje C, pero por ejemplo en Pascal se puede expresar de la forma:

```
TYPE T = mín .. máx
```

donde `mín` y `máx` son expresiones que especifican los límites del intervalo. Cabe hacer notar estas expresiones deben contener constantes sólo como operandos.

4 Estructuras estáticas de datos.

Una estructura de datos o tipo de dato estructurado es un tipo de dato construido a partir de otros tipos de datos. Así, un dato de tipo complejo, que representa al conjunto de los números complejos, es un par ordenado de datos reales, y por tanto un tipo de dato estructurado.

Un dato de tipo estructurado está compuesto por una serie de datos de tipos elementales y alguna relación existente entre ellos. Normalmente la relación suele ser de orden, aunque puede ser de otro tipo.

Una estructura de datos se dice que es homogénea cuando todos los datos elementales que la forman son del mismo tipo. En caso contrario se dice que la estructura es heterogénea. Por ejemplo, el tipo de datos complejo es una estructura homogénea, tanto la parte real como la parte imaginaria se representan con datos reales.

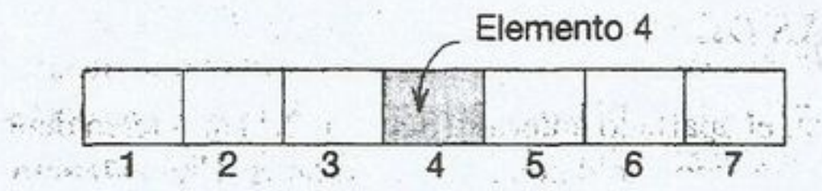
Siempre que se utilice un dato en un programa debe estar determinado su tipo, para que el traductor sepa cómo debe tratarlo y almacenarlo. En el caso de datos de tipos elementales, el tipo del dato determina el espacio que se usa en memoria. Esto puede no ocurrir si el dato es de un tipo estructurado. Algunos tipos estructurados (listas y árboles) se declaran sin especificar el número de componentes que van a tener. En este caso el compilador les reserva el espacio de memoria mínimo que necesitan. Durante la ejecución del programa la estructura de datos puede ir creciendo, es decir, ocupando más memoria. En cualquier caso, el máximo espacio al que pueden llegar está limitado por el espacio libre en el programa. Si se necesitase más memoria de la disponible en el programa, este terminaría por error. Una estructura de datos que es gestionada de esta forma se dice que es dinámica, ya que la memoria que necesita se asigna dinámicamente. Por el contrario, una estructura de datos que siempre ocupa el mismo espacio se dice que es estática.

4.1 Arrays.

El array es la estructura de datos más usual. Existe en la mayoría de los lenguajes de programación y en algunos es de las pocas estructuras de datos existentes (Basic y FORTRAN).

Un array es una estructura de datos formada por una cantidad fija de datos de un mismo tipo, cada uno de los cuales tiene asociado uno o más índices que determinan de forma unívoca la posición del dato en el array. Cada índice es un dato de tipo subrango. Para cada combinación posible de valores de índices existe un y sólo un dato del tipo constituyente, o elemento del array.

Podemos imaginar un array como una estructura de celdas donde se pueden almacenar valores. La siguiente figura representa un array de un solo índice que toma valores de 1 a 7.



El índice será el valor del tipo definido como el tipo de índice del array, y concretamente en lenguaje C son enteros, donde el mínimo es siempre 0.

```
Tipo_base x[expresión];
```

donde `x` es el identificador del array, y `expresión` es una expresión entera positiva que indica el número de elementos del array.

Un componente individual de un array puede ser seleccionado por un índice. Dada una variable de array `x`, representamos un selector de array por medio del nombre de dicho array seguido del índice del componente respectivo `i` y se escribe x_i o bien `x[i]`.

Ejemplos:

```
float x[5];
```

Un cierto valor de los elementos del array `x` pueden visualizarse en la figura.

x[0]	0.5
x[1]	0.25
x[2]	0.125
x[3]	0.0625
x[4]	0.03125

La manera común de operar con arrays, particularmente con elementos grandes, consiste en actualizar selectivamente componentes simples en vez de construir valores estructurados completamente nuevos. Esto se expresa tomando en consideración una variable de array como un array de variables de componente y permitiendo asignaciones a componentes seleccionados, como por ejemplo `x[i]=0.125`. Aunque la actualización selectiva ocasiona solamente que cambie un solo valor del componente, desde un punto de vista conceptual debemos considerar el valor compuesto en su totalidad como si también hubiera cambiado.

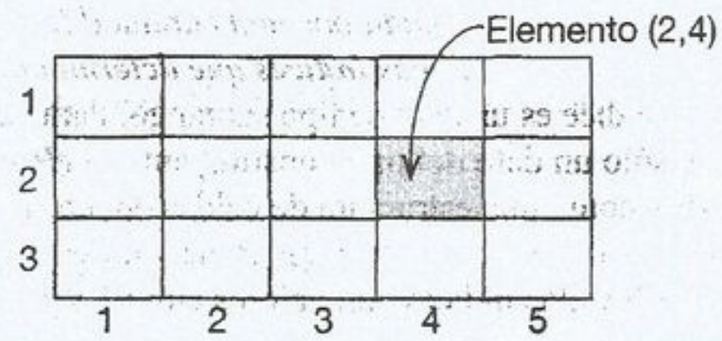
Los constituyentes de los arrays se pueden estructurar por sí solos. Por ejemplo, si se tiene:

```
float x[10][5];
```

es un array que consta de diez componentes, cada uno de los cuales consta a su vez de cinco componentes de tipo real y se llama matriz 10 x 5 con componentes reales. Los selectores pueden eslabonarse siguiendo este modelo, `x[i][j]` ó `x[i,j]`.

Al número de índices de un array se le denomina dimensión. Así, podemos tener arrays unidimensionales, bidimensionales, tridimensionales, etc. Concretamente, a los arrays unidimensionales se les conoce también como vectores, mientras que a los bidimensionales se les conoce como tablas.

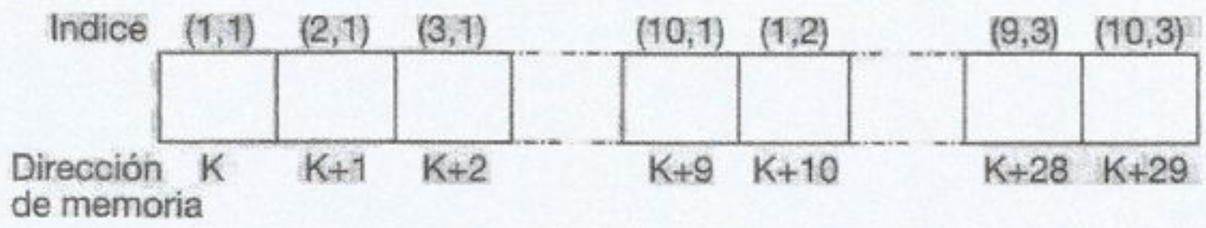
El array de la siguiente figura utiliza dos índices con valores entre 1 y 3, el primero, y entre 1 y 5, el segundo. Cada elemento de este array está especificado por un par ordenado de números, el valor de los dos índices.



Un array puede considerarse como una aplicación de un dominio subrango (o producto cartesiano de subrangos) al tipo de datos del array.

La cardinalidad de un tipo estructurado, es decir, el número de valores que pertenecen a este tipo, es el producto de la cardinalidad de sus componentes.

Un array es una estructura de datos estática. Esto es, al definirla se especifica el número de elementos que la constituyen. Este dato lo utiliza el compilador para reservar el espacio necesario para almacenarla. Los arrays se almacenan en memoria ocupando un área contigua. Cada elemento ocupa el mismo número de palabras, que será el que corresponda al tipo de éstos. Los elementos se colocan en la memoria según un orden prefijado de los índices.



4.2 Cadenas de caracteres.

Una cadena de caracteres es una estructura de datos formada por una secuencia de caracteres en un orden determinado. En una variable de tipo cadena se puede almacenar una palabra, una frase, una matrícula de un coche, una temperatura, etc. Las constantes de este tipo se escriben normalmente entre comillas.

Sobre datos de tipo cadena de caracteres se pueden realizar las siguientes operaciones:

- Concatenación: consiste en formar una cadena a partir de dos ya existentes, yuxtaponiendo los caracteres de ambas.
- Extracción de subcadena: permite formar una cadena (subcadena) a partir de otra ya existente. La subcadena se forma tomando un tramo consecutivo de la cadena inicial.

- Comparación de cadenas: es posible comparar dos cadenas en conjunto. Se considera menor aquella en que el primer carácter en que difieren ambas es menor.
- Obtención de longitud. la longitud de una cadena es un dato de tipo entero, cuyo valor es el número de caracteres que contiene.

Aunque una cadena de caracteres y un array lineal de caracteres puedan contener la misma información, son tipos de datos distintos. Representan objetos distintos y permiten realizar operaciones diferentes. Sin embargo, en determinados lenguajes, como por ejemplo C, no existe este tipo, por lo que se implementa mediante arrays lineales.

4.3 Registros o estructuras.

Un registro es una estructura de datos formada por yuxtaposición de elementos que contienen información relativa a un mismo ente. A los elementos que componen el registro se les denomina campos. Cada campo es de un determinado tipo. Los campos dentro del registro aparecen en un orden determinado, y se identifican por un nombre. Para definir un registro es necesario especificar el nombre y tipo de cada campo. Los campos pueden ser de un tipo estructurado.

Con los registros se pueden realizar asignaciones del registro completo a una variable de tipo registro, definida con los mismos campos en el mismo orden. Se puede realizar, al igual que en arrays, la selección de un campo. Esto se realiza especificando el nombre del campo.

El método más general para obtener tipos estructurados consiste en asociar elementos de tipos arbitrarios, que sean posiblemente tipos estructurados por sí solos, en uno compuesto. Ejemplos de matemáticas son los números complejos, compuestos por dos números reales, y coordenadas de puntos, compuestas por dos o más números de acuerdo con la dimensionalidad del espacio ocupado por el sistema de coordenadas. Un ejemplo de procesamiento de datos es la descripción de personas por medio de unas cuantas características relevantes, como su nombre y apellido, fecha de nacimiento, sexo y estado civil.

En el procesamiento de datos, los tipos compuestos, como las descripciones de personas u objetos, generalmente se presentan en archivos o bancos de datos y registran las características relevantes de una persona u objeto. La palabra registro se ha vuelto por tanto ampliamente aceptada para describir un compuesto de datos de esta naturaleza y adoptamos esta nomenclatura en lugar del término producto Cartesiano usado en matemáticas.

Las estructuras de registro y array tienen la propiedad común de que ambas son estructuras con acceso aleatorio. El registro es más general en el sentido de que no es requisito que todos los tipos constituyentes deban ser idénticos. A su vez, el array ofrece mayor flexibilidad al permitir que sus selectores de componentes sean valores (expresiones) calculables, en tanto que los selectores de componentes de registro son identificadores de campo declarados en la definición del tipo de registro.

Los registros se mapean en la memoria de una computadora por la simple yuxtaposición de sus componentes. La dirección de un componente (campo) s_i relativa a la dirección de origen del registro `r` se llama composición k_i del campo. Esta se calcula como $k_i=s_1+s_2+...+s_{i-1}$, donde s_j es el tamaño (en palabras) del j-ésimo componente. Ahora comprendemos que el hecho de que todos los componentes de un array sean del mismo tipo tiene la bienvenida consecuencia de que $k_i=(i-1)*s$. La

generalidad de la estructura del registro por desgracia no admite dicha función lineal simple para calcular la compensación de la dirección y ésta es por tanto la razón real del requisito de que los componentes del registro sean seleccionables sólo por identificadores fijos. Esta restricción tiene el beneficio deseable de que las compensaciones respectivas se conocen en el momento de la compilación. La mayor eficiencia resultante del acceso del campo de registro es bien conocida.

En lenguaje C, una estructura se define como sigue:

```
struct T
{
   T1 s1;
   T2 s2;
   ...
   Tn sn;
}
```

Ejemplo:

```
struct complejo
{
   float re;
   float im;
}
```

4.4 Secuencias o ficheros.

Todos los elementos de una secuencia son del mismo tipo, llamado tipo base T_0 de la secuencia. Una secuencia s la representaremos con n elementos por medio de

$$s = <s_0, s_1, s_2, \ldots, s_{n-1}>$$

donde a n se le denomina longitud de la secuencia. Esta estructura parece ser muy semejante al array. La diferencia esencial es que, en el caso del array, el número de elementos es fijado por la declaración del array, en tanto que en la secuencia se deja abierto. Esto implica que puede variar durante la ejecución del programa. Aunque toda secuencia tiene en cualquier instante una longitud específica y finita, debemos considerar la cardinalidad de un tipo de secuencia como infinito, ya que no hay límite fijo para la longitud potencial de las variables de una secuencia.

Una secuencia se inspecciona procediendo estrictamente de un elemento a su sucesor inmediato, que se genera agregando repetidamente un elemento en su extremo. La consecuencia inmediata es que los elementos no son directamente accesibles, con excepción del elemento que se encuentra regularmente en espera de inspección. Es esta disciplina de acceso la que distingue en forma fundamental las secuencias de los arrays.

La ventaja de apegarse al acceso secuencial que, después de todo, es una restricción grave, es la relativa simplicidad del manejo de la memoria que se requiere. Pero aún más importante es la posibilidad de utilizar técnicas de manejo por buffer efectivas cuando se desplazan datos hacia o desde dispositivos secundarios de almacenamiento. Dado solamente un acceso secuencial, el mecanismo de manejo por buffer es razonablemente directo para todas las secuencias y todos los medios. Por lo tanto, puede convertirse con toda seguridad en un sistema de uso general y el programador no tiene que verse agobiado por tener que incorporarlo en su programa.

A dicho sistema por lo general se le llama sistema de archivo o fichero, ya que los dispositivos de acceso secuencial de alto volumen se utilizan para el

almacenamiento permanente de datos y los conservan aun cuando la computadora sea desconectada. La unidad de datos en estos medios es la secuencia o archivo secuencial.

La disciplina del acceso secuencial puede reforzarse ofreciendo un conjunto de operadores de secuencia a través de los cuales se puede acceder a las variables de las secuencias exclusivamente. Por tanto, aunque podamos aquí referirnos al i-ésimo elemento de una secuencia s escribiendo s_i, esto no será posible en un programa. Evidentemente, el conjunto de operadores debe contener un operador para generar y uno para inspeccionar una secuencia. Como ya se mencionó, una secuencia se genera agregando elementos en su extremo, y se inspecciona procediendo al siguiente elemento. Por lo tanto, una cierta posición siempre está asociada con toda secuencia. Ahora postularemos y describiremos informalmente el siguiente conjunto de operadores primitivos:

- `open(s)`, define a `s` como la secuencia vacía, es decir, la secuencia de longitud 0.
- `write(s, x)`, agrega un elemento con valor `x` a la secuencia `s`.
- `reset(s)`, coloca la posición actual (indicador de posición) al inicio de `s`.
- `read(s, x)`, asigna el elemento designado por la posición actual a la variable `x` y adelanta la posición hacia el siguiente elemento.

Con mucha frecuencia, los sistemas de archivo admiten alguna relajación de la disciplina del acceso estrictamente secuencial sin penalización grave. En particular, permiten el posicionamiento de la secuencia en un sitio arbitrario en vez de sólo en el inicio. Consecuentemente, si una operación de escritura ocurre cuando la posición del archivo no está en el extremo, el nuevo elemento sustituye la cola anterior en su totalidad, es decir, la secuencia se trunca.

Cuando los datos son transferidos hacia o desde un dispositivo de almacenamiento secundario, los bits individuales se transfieren como un flujo. Por lo general, un dispositivo impone restricciones estrictas de temporización en la transmisión. Por ejemplo, si los datos se escriben sobre una cinta, la cinta se desplaza a una velocidad fija y requiere que los datos sean alimentados a una razón fija. Cuando la fuente se detiene, el movimiento de la cinta se suspende y la velocidad desciende rápida, pero no instantáneamente. Así, se deja un espacio entre los datos transmitidos y los que siguen en un instante posterior. A fin de lograr una alta densidad de datos, el número de espacio tiene que conservarse pequeño y por tanto, los datos se transmiten en bloques relativamente grandes una vez que la cinta se mueve. Condiciones semejantes se cumplen con los discos magnéticos, donde los datos se asignan a pistas con un número fijo de bloques de tamaño específico, el así llamado tamaño del bloque. De hecho, un disco debe considerarse como un array de bloques, con cada bloque siendo leído o escrito como un todo.

Sin embargo, nuestros programas no observan ninguna de estas restricciones de temporización. A fin de permitirles ignorar las restricciones, los datos por transferir son manejados por buffer. Se colectan en una variable llamada buffer (en memoria principal) y se transfieren cuando se acumula una cantidad suficiente de datos para formar un bloque del tamaño que se pide.

El manejo por buffer tiene una ventaja adicional al permitir que el proceso que genera (recibe) datos proceda concurrentemente con el dispositivo que escribe (lee) los datos de (hacia) el buffer. De hecho, conviene considerar al dispositivo como un

proceso en sí que meramente copia bloques de datos. La finalidad del buffer consiste en asignar un cierto grado de desacoplamiento entre los dos procesos, a los que llamaremos productor y consumidor. Si, por ejemplo, el consumidor es lento en un cierto momento, puede encontrarse con el productor más adelante. Este desacoplamiento es a menudo esencial para lograr una buena utilización de dispositivos periféricos, pero sólo tiene efecto si los índices del productor y consumidor son casi iguales en promedio, pero fluctúan en tiempos. El grado de desacoplamiento crece cuando aumenta el tamaño del buffer.

Ahora centramos nuestra atención en la cuestión de cómo representar un buffer y se supondrá, en el tiempo que sea, que los elementos de datos se depositan y extraen de forma individual en vez de en bloques. Un buffer esencialmente constituye una cola de primero en entrar, primero en salir (FIFO). Si se declara como un array, dos variables de índice, por ejemplo, `in` (dentro) y `out` (fuera), marcan las posiciones de la siguiente localidad que se escribirá y que será leída. En teoría, dicho array no debe tener límites de índice. Un array finito es muy adecuado, considerando el hecho de que los elementos una vez traídos ya no son relevantes. Su localidad puede bien volver a usarse. Esto nos conduce a la idea del buffer circular. Este mecanismo implica asimismo una variable n que cuenta el número de elementos que hay regularmente en el buffer.

5 Conclusiones.

Los tipos de datos simples o primitivos significan que no están compuestos de otras estructuras de datos. Los más frecuentes y utilizados por casi todos los lenguajes son: enteros, reales, y caracteres, siendo los tipos lógicos o booleanos, subrango y enumerados propios de lenguajes estructurados. Los tipos de datos estructurados están constituidos por tipos de datos primitivos.

Los tipos de datos primitivos pueden ser organizados en diferentes estructuras de datos: estáticas y dinámicas. Las estructuras de datos estáticas son aquéllas en las que el tamaño ocupado en memoria se define antes de que el programa se ejecute y no puede modificarse dicho tamaño durante la ejecución del programa. Estas estructuras están implementadas en casi todos los lenguajes: arrays (vectores, tablas o matrices), registros, y secuencias o ficheros. Las estructuras de datos dinámicas no tienen las limitaciones o restricciones en el tamaño de memoria ocupada que son propias de las estructuras estáticas.

Una característica importante que diferencia a los tipos de datos es la siguiente: los tipos de datos primitivos tienen como característica común que cada variable representa a un elemento; los tipos de datos estructurados tienen como característica común que un identificador (nombre) puede representar a múltiples datos individuales, pudiendo cada uno de éstos ser referenciados independientemente.